

Syncro SVN Client 4.2 User Manual

SyncRO Soft Ltd.

Syncro SVN Client 4.2 User Manual

SyncRO Soft Ltd.

Copyright © 2002-2009 SyncRO Soft Ltd. All Rights Reserved.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and SyncRO Soft Ltd., was aware of a trademark claim, the designations have been printed in caps or initial caps. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Third party software components are distributed in the Syncro SVN Client installation packages, including the Java Runtime Environment (JRE). This product includes software developed by the Apache Software Foundation (<http://www.apache.org> [<http://www.apache.org>]): Xerces XML Parser . These products are not the property of SyncRO Soft Ltd.. To the best knowledge of SyncRO Soft Ltd. owners of the aforesaid products granted permission to copy, distribute and/or modify the software and its documents under the terms of the Apache Software License, Version 1.1. Other packages are used under the GNU Lesser General Public License. Users are advised that the JRE is provided as a free software, but in accordance with the licensing requirements of Sun Microsystems. Users are advised that SyncRO Soft Ltd. assumes no responsibility for errors or omissions, or for damages resulting from the use of Syncro SVN Client and the aforesaid third party software. Nor does SyncRO Soft Ltd. assume any responsibility for licensing of the aforesaid software, should the relevant vendors change their terms. By using Syncro SVN Client the user accepts responsibility to maintain any licenses required by SyncRO Soft Ltd. or third party vendors. Unless SyncRO Soft Ltd. declares in writing that the Syncro SVN Client license is inclusive of third party licensing.

Table of Contents

1. Introduction	1
2. Installation	2
Installation Requirements	2
Platform Requirements	2
Operating System, Tools and Environment Requirements	2
Operating System	2
Tools	2
Environment Prerequisites	2
Installation Instructions	3
Unattended installation	5
Setting a parameter in the startup script	6
Starting the application	6
Obtaining and registering a license key	7
Named User license registration	7
License registration with a registration code	8
Unregistering the license key	8
Upgrading the Syncro SVN Client application	9
Checking for new versions	9
Uninstalling the application	10
Unattended uninstall	10
Performance problems	10
Large documents	10
Display problems on Linux/Solaris	11
3. The Syncro SVN Client	12
Introduction	12
What is Syncro SVN Client	12
Quick start guide and reference	12
Main window	13
Starting Syncro SVN Client	13
Views	14
Main menu	14
Getting started	18
Define a repository location	18
Add / Edit / Remove repository locations	18
Authentication	19
Defining a working copy	21
Check out a working copy	21
Depth	22
Revision	22
Use an existing working copy	24
Manage working copy resources	24
Edit files	24
Add resources to version control	25
Ignore resources not under version control	25
Delete resources	26
Copy / Move / Rename resources	26
Lock / Unlock resources	27
Scanning for locks	28
Locked items	28
Locking a file	28
Unlocking a file	29

Synchronize with the repository	29
Presentation modes	30
View differences	32
Resolve conflicts	33
Real conflicts vs mergeable conflicts	33
Content conflicts vs Property conflicts	34
Edit real content conflicts	34
Revert your changes	36
Merge conflicted resources	37
Drop incoming modifications	37
Tree conflicts	38
Update the working copy	39
Send your changes to the repository	40
Integration with Bug Tracking Tools	41
Obtain information for a resource	42
Request status information for a resource	42
Request history for a resource	43
Using the resource history view	44
History actions available in the popup menu displayed by a right click in the view when a single resource is selected:	44
History actions available on the popup menu for double selection:	45
Directory Change Set View	45
Management of SVN properties	46
Add / Edit / Remove SVN properties	47
Creation and management of Branches/Tags	48
Create a Branch/Tag	48
Merging	49
Merge revisions	50
Reintegrate a branch	52
Merge two different trees	52
Merge Options	53
Resolve merge conflicts	55
Switch the Repository Location	56
Relocate a Working Copy	56
Create Patches	56
Create a patch from working copy	57
Include unversioned files in the patch	59
Create patch from repository revision	60
Working with repositories	62
Import / Export resources	62
Import resources into the repository	62
Export resources from the repository	62
Copy / Move / Delete resources from the repository	62
Sparse checkouts	63
Repository View	64
General description	64
Toolbar	64
Contextual menu actions	64
Working Copy View	66
General description	66
Toolbar	67
Contextual menu actions	68
Drag and drop operations	71
Icons	72

Synchronize View	73
General description	73
Synchronize trees	73
Toolbar	73
Contextual menu actions	74
Icons	75
Compare View	76
Description	76
Toolbar	76
Compare images view	77
Editor	78
Description	78
Image preview	79
Description	79
History View	79
Description	79
History Filters	80
The History filter dialog	80
The History filter field	81
Features	81
Annotations View	82
Description	82
Properties View	83
Description	83
The <i>svn:externals</i> property	84
Toolbar / Contextual menu	85
Console View	85
Description	85
Help View	85
Description	85
The Revision Graph of a SVN Resource	85
Command line interface cross reference	89
Actions commands reference	89
Checkout	89
Update	89
Commit	89
Diff	89
Show History	90
Refresh	90
Synchronize	90
Import	90
Export	90
Information	90
Add	90
Add to svn:ignore	91
Delete	91
Copy	91
Move / Rename	91
Mark resolved	91
Revert	91
Cleanup	91
Show / Refresh Properties	92
Branch / Tag	92
Merge	92

Scan for locks	92
Lock	92
Unlock	93
Mark as merged	93
Override and update	93
Override and commit	93
Add / Edit property	93
Remove property	93
Revert changes from this revision	93
Revert changes from these revisions	93
4. Text editor specific actions	95
Undoing and redoing user actions	95
Copying and pasting text	95
Finding and replacing text in the current file	95
The Find/Replace dialog	95
Keyboard shortcuts for finding the next and previous match	97
VI editor actions	97
Dragging and dropping the selected text	98
Exiting the application	98
5. Configuring the application	99
Importing/Exporting Global Options	99
Preferences	99
Global	99
Fonts	101
Encoding	101
Diff Appearance	102
Menu Shortcut Keys	103
SVN File Editors	104
HTTP / Proxy Configuration	105
Messages	107
SVN	107
SVN Diff	109
Reset Global Options	110
6. Common problems	111
Index	112

Chapter 1. Introduction

Welcome to the User Manual of the Syncro SVN Client 4.2 ! This book explains how to use the 4.2 version of the Syncro SVN Client effectively to access Subversion repositories and manage their local working copies. Please note that this manual assumes that you are familiar with the basic concepts of a version control system.

The Syncro SVN Client is a cross-platform application for managing the history of a set of files that change over time and are stored in a central repository using a version control system.

Chapter 2. Installation

This section explains platform requirements and installation procedures. It also provides instructions on how to obtain and register a license key, how to perform upgrades and uninstall the application if required.

If you need help at any point during these procedures please send email to <support@syncrosvnclient.com>

Installation Requirements

Platform Requirements

Minimum run-time requirements are listed below.

- Pentium Class Platform
- 256 MB of RAM
- 300 MB free disk space

Operating System, Tools and Environment Requirements

Operating System

Windows	Windows 98 or later.
Mac OS	minimum Mac OS X 10.4
UNIX/Linux	All versions/flavors

Tools

Installation packages are supplied in compressed archives. Ensure you have installed a suitable archive extraction utility with which to extract the archive. The MD5 sum is available on the Download page [<http://www.syncrosvnClient.com/download.html>] for every archive. You should check the MD5 sum of the downloaded archive with a MD5 checking tool available on your platform.

Environment Prerequisites

Prior to installation ensure that your Operating System environment complies with the following:

- Syncro SVN Client supports only official and stable Java virtual machine versions 1.5.0 and later from Sun Microsystems (available at <http://java.sun.com>) and from Apple Computer (pre-installed on Mac OS X). For Mac OS X, Java VM updates are available at <http://www.apple.com/macosx/features/java/>. Syncro SVN Client may work very well with JVM implementations from other vendors but the eventual incompatibilities will not be solved in further Syncro SVN Client releases. Syncro SVN Client does not work with the GNU libgcj Java virtual machine [<http://www.oxygenxml.com/forum/ftopic1887.html>]. We recommend using Java virtual machine 1.6.0 or later for better performance.
- The PATH environment variable is set to the most current Java VM installation.
- References to older Java VM installations are removed from the PATH.

Installation Instructions

Prior to proceeding with the following instructions, please ensure that your system complies with the prerequisites detailed in the installation requirements.

Note

The following instructions assume that a Java Runtime Environment (JRE) is installed. If you have downloaded an installation package that contains the JRE, please note that the package will automatically install a JRE prior to execution of the application but this JRE will be used on your computer only for running Syncro SVN Client, it will be invisible to other applications.

Note

The installation kits and the executable files packaged inside the installation kits were checked before publication with an antivirus program to make sure they are not infected with viruses, trojan horses or other malicious software.

Procedure 2.1. Windows Installation

1. Download the `syncroSVNClient.exe` installation kit and run it.
2. Follow the instructions presented in the installation program. The user preferences are stored in the subfolder `com.syncrosvnclient` of the folder that is the value of the APPDATA Windows variable for the user that starts the application.

Note

In order to specify another Java virtual machine to be used by Syncro SVN Client you have to set the home folder of the desired JVM in the Windows variable `JAVA_HOME` or in the Windows variable `JDK_HOME`. If `JAVA_HOME` and `JDK_HOME` are not set the application launcher will try to detect a JVM installed in a standard location on the computer and use it for running the application. If you installed the kit which includes a Java virtual machine you have to rename or remove the `jre` subfolder of the install folder in order for the variable `JAVA_HOME` or `JDK_HOME` to have an effect.

Procedure 2.2. Mac OS X Installation

1. Create a folder called `syncroSVNClient` on your local disk.
2. Within the `syncroSVNClient` folder, create child folder named in accordance with the version number of the application. The directory structure looks as follows: `../syncroSVNClient/4.2/`
3. Download the Mac OS X Installation package (`syncroSVNClient.tar.gz`) to this folder.
4. Extract the archive to the same folder.
5. Execute the file named `syncroSVNClient`

Note

Syncro SVN Client uses the first JVM from the list of preferred JVM versions set on your Mac computer that has the version number not less than 1.5.0. To change the version of the Java virtual machine that runs the application you must move your desired JVM version up in the preferred list by dragging it with the

mouse on the first position in the list of JVMs available from Applications -> Utilities -> Java -> Java Preferences.

Procedure 2.3. Linux Installation

1. Download the `syncroSVNClient.sh` installation kit and run it.
2. Follow the instructions presented in the installation program.

Note

In order to specify another Java virtual machine to be used by Syncro SVN Client you have to set the home folder of the desired JVM in the environment variable `JAVA_HOME` or in the environment variable `JDK_HOME`. If `JAVA_HOME` and `JDK_HOME` are not set the application launcher will try to detect a JVM installed in a standard location on the computer and use it for running the application.

Procedure 2.4. All Platforms Installation

1. Create a folder called `syncroSVNClient` on your local disk.
2. Within the `syncroSVNClient` folder, create child folder named in accordance with the application version number. The directory structure looks as follows: `../syncroSVNClient/4.2/`
3. Download the All Platforms Installation package (`syncroSVNClient.tar.gz`) to this folder.
4. Extract the archive to the same folder.
5. Run from a command line the script `syncroSVNClient.bat` on Windows, `syncroSVNClientMac.sh` on Mac OS X, `syncroSVNClient.sh` on Unix/Linux.

Note

To change the version of the Java virtual machine that runs the application you have to specify the full path to the java executable of the desired JVM version in the Java command at the end of the script file, for example:

```
"C:\Program Files\Java\jre1.5.0_13\bin\java" -Xmx256m  
-Dsun.java2d.noddraw=true ...
```

on Windows,

```
/System/Library/Frameworks/JavaVM.framework/Versions/  
1.5.0/Home/bin/java "-Xdock:name=SyncroSVNClient" ...
```

on Mac OS X.

Procedure 2.5. Windows NT Terminal Server

1. Install the application on the server, making its shortcuts available to all users.
2. Edit the `syncroSVNClient.vmoptions` file located in the install folder, adding the parameter `-Dcom.oxygenxml.MultipleInstances=true` so that the file content looks like:

```
-Xmx256m  
-Dcom.oxygenxml.MultipleInstances=true
```

The "-Xmx" value represents the maximum memory for each application instance. Please make sure you tune them in a way that the multiple editor instances won't use all the available physical memory.

Procedure 2.6. Unix/Linux Server

1. Install the editor on the server, making sure the `syncroSVNClient.sh` script is executable and the installation directory is in the PATH of the users that need to use the editor.
2. Create a file called `syncroSVNClient.vmoptions` in the Syncro SVN Client install folder where the `syncroSVNClient4.2` file is located. The content of the file must be:

```
-Xmx256m -Dcom.oxygenxml.MultipleInstances=true
```

The "-Xmx" value represents the maximum memory for each editor instance. Please make sure you tune it in a way that the multiple editor instances won't use all the available physical memory.

3. Make sure the X server processes located on the workstations allow connections from the server host. For this use the `xhost` command.
4. Telnet (or ssh) on the server host.
5. Start an xterm process, with display on the workstation. Ex: **xterm -display workstationip:0.0**
6. Start the application by typing **syncroSVNClient.sh**

Unattended installation

Unattended installation is possible only on Windows and Linux by running the installer executable from command line and passing the `-q` parameter. The installer executable is called `syncroSVNClient.exe` on Windows and `syncroSVNClient.sh` on Linux

In unattended mode the installer does not overwrite files with the same name if a previous version of the application is installed in the same folder. The `-overwrite` parameter added after `-q` forces overwriting these files.

If the installer is executed in silent (unattended) mode and `-console` is passed as a second parameter after `-q` a console will be allocated on Windows that displays the output of the installer. The command for running the installer is in this case:

```
start /wait syncroSVNClient.exe -q -console
```

By default an unattended installation applies the default settings of the installer. If you want to install the application on a large number of computers but you need to change the default values of some settings (like the install folder on disk, whether a desktop icon or a quick launch shortcut are created, the file associations created in the operating system, the name of the program group on the Start menu, etc.) then you should use a special settings file which specifies the new values for these settings. To generate the settings file you have to run the installer in normal attended mode once on a test computer and specify the exact options that you want for the unattended installation. When the installation is completed a file called `response.varfile` and containing your selected options is created in the `.install4j` subfolder of the installation folder, by default `C:\Program Files\Syncro SVN Client 4.2\.install4j` on Windows. This is a one time process. After that for applying these options on all the computers where an unattended installation is performed you have to specify this file in the command line, for example copy the file in the same location as the installer program and use the command:

- on Windows: `syncroSVNClient.exe -q -varfile response.varfile`

- on Linux: `syncroSVNClient.sh -q -varfile response.varfile`

Setting a parameter in the startup script

On the Windows platform if you start the application by double-clicking on the Start menu shortcut/Desktop shortcut in order to set a startup parameter you have to add a line with the parameter to the file `syncroSVNClient.vmoptions` located in the installation directory together with the launcher file called `syncroSVNClient.exe`. If the file `syncroSVNClient.vmoptions` does not exist yet in the folder of the launcher file you have to create it there. For example for setting the maximum amount of Java memory to 600 MB the content of the file `syncroSVNClient.vmoptions` must be:

```
-Xmx600m
```

If you start the application with the script `syncroSVNClient.bat` you have to add or modify the parameter to the java command at the end of the script. For example for setting the maximum amount of Java memory to 600 MB the java command should start with:

```
java -Xmx600m -Dsun.java2d.noddraw=true ...
```

On the Mac OS X platform to add or modify a startup parameter you have to right-click on the Syncro SVN Client application icon in Finder, in the pop-up menu select *Show Package Contents*, then in the *Contents* directory you edit the file `Info.plist`: in the key *VMOptions* you modify the parameter if it already exists in that key or you add it after the model of the existing parameters inside that key.

On the Linux platform you have to create a file called `syncroSVNClient.vmoptions` if it does not exist already and specify the parameter exactly as in the case of the `.vmoptions` file on the Windows platform.

If you use the *All platforms* distribution you have to add or modify the startup parameter that you want to set in the java command line at the end of the startup script `syncroSVNClient.bat` on Windows, `syncroSVNClient-Mac.sh` on Mac OS X and `syncroSVNClient.sh` on Linux. All these files are located in the installation directory. For example for setting the maximum amount of Java memory to 600 MB on Windows the `-Xmx` parameter must be modified in the java command at the end of `syncroSVNClient.bat` like this:

```
java -Xmx600m -Dsun.java2d.noddraw=true ...
```

on Mac OS X the java command at the end of `syncroSVNClientMac.sh` should look like:

```
java "-Xdock:name=SyncroSVNClient"\  
-Dcom.oxygenxml.editor.plugins.dir="$SYNCRO_SVN_CLIENT_HOME/plugins"\  
-Xmx600m\  

```

and on Linux the java command at the end of `syncroSVNClient.sh` should look like:

```
java -Xmx600m\  
"-Dcom.oxygenxml.editor.plugins.dir=$SYNCRO_SVN_CLIENT_HOME/plugins"\  

```

Starting the application

As a Java based application, the Syncro SVN Client can run on all Operating Systems that support the Java Runtime Environment (JRE version 1.5.0 or later). The following instructions assume that JRE and the appropriate Syncro SVN Client distribution package for your Operating System are installed.

To start the application follow the instructions for the installed package:

Procedure 2.7. Windows

- From the Windows Explorer double-click `syncroSVNClient.exe`.

Procedure 2.8. Linux

- At the prompt type: `sh syncroSVNClient.sh`.

Procedure 2.9. Mac OS X

- Double-click the `syncroSVNClient` icon.

Procedure 2.10. All Platforms

- On Windows run `syncroSVNClient.bat`. On Mac OS X run `syncroSVNClientMac.sh`. On Linux/Unix run `syncroSVNClient.sh`

Obtaining and registering a license key

The Syncro SVN Client is not free software and requires a license in order to enable the application.

For demonstration and evaluation purposes a time limited license is available upon request from the Syncro SVN Client [<http://www.syncrosvnClient.com/register.html>] web site. This license is supplied at no cost for a period of 30 days from date of issue. During this period the Syncro SVN Client is fully functional enabling you to test all aspects of the application. Thereafter, the application is disabled and a permanent license must be purchased in order to use the application. For special circumstances, if a trial period of greater than 30 days is required, please contact support@syncrosvnClient.com. All licenses are obtained from the Syncro SVN Client web site [<http://www.syncrosvnClient.com>].

For definitions and legal details of the license types available for Syncro SVN Client you should consult the End User License Agreement received with the license key and available also on the Syncro SVN Client website at <http://www.syncrosvnClient.com/eula.html>

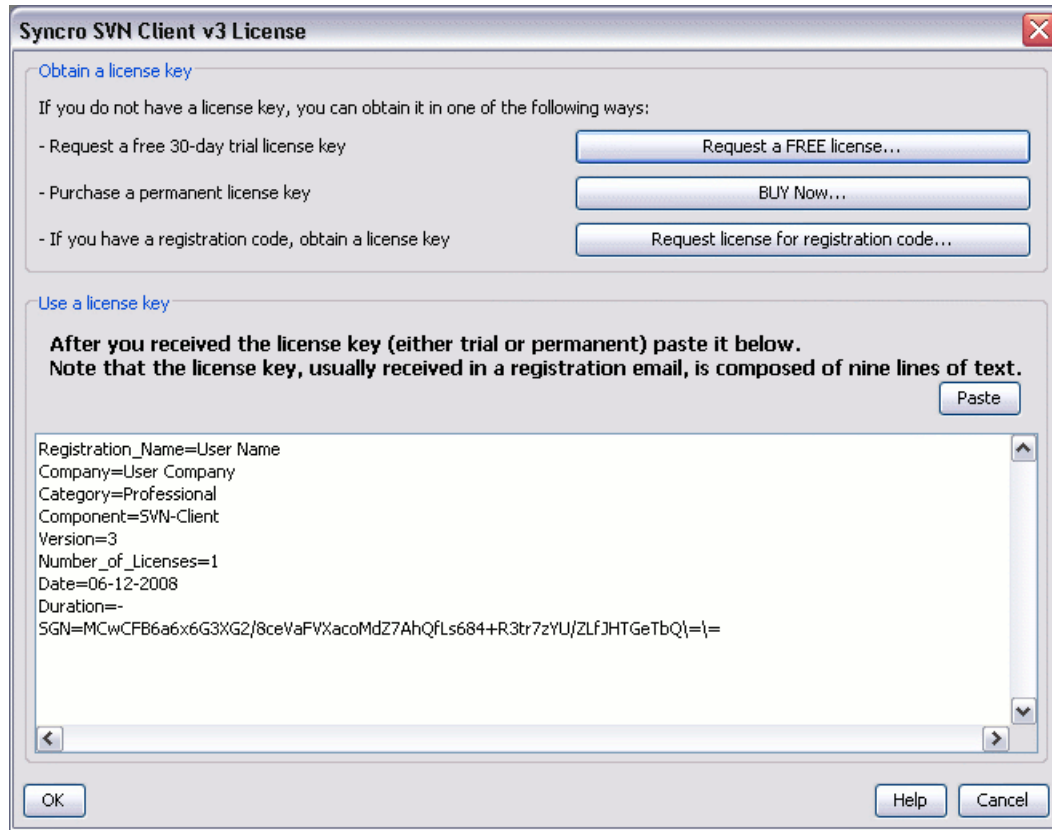
Note

Starting with version 10.0 <oxygen> accepts a license key for a newer version in the license registration dialog, e.g. version 10.0 accepts a license key for version 11 or a license key for version 12.

Once you have obtained a license key the installation procedure is described below.

Named User license registration

1. Save a backup copy of the message containing the new license key.
2. Start the application.
3. Copy to the clipboard the license text as explained in the message.
4. If this is a new install of the application then it will display automatically the registration dialog when it is started. In the case you already used the application and obtained a new license, use the menu option Help/Register to make the registration dialog appear.

Figure 2.1. Registration Dialog

5. Paste the license text in the registration dialog, and press OK.

You have the following alternative for the procedure of license install:

1. Save the license key in a file named `licensekey.txt`.
2. Copy the file in the application folder. In that way the license will not be asked when the application will start.
- 3.

License registration with a registration code

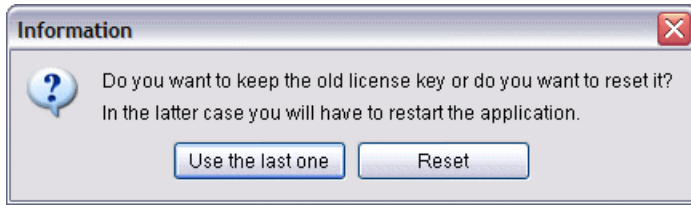
If you have only a registration code and you want to register the associated license key you must request this license key by filling the registration code and other details associated with your license in a request form on the Syncro SVN Client website. The button **Request license for registration code** in the registration dialog available from menu Help → Register opens this request form in the default Web browser on your computer.

Unregistering the license key

Sometimes you need to unregister your license key, for example to transfer your license key to other computer before other user starts using your current computer. This is done by going to Help → Register to display the license registration dialog, making sure the text area for the license key is empty and the checkbox *Use a license server* is unchecked, and pressing the OK button of the dialog. This brings up a confirmation dialog in which you select between falling back

to the license key entered previously (for the case of releasing a floating license and reverting to the individual license entered previously in the *Register* dialog) and removing your license key from your user account of the computer.

Figure 2.2. Unregister a license key



Upgrading the Syncro SVN Client application

From time to time, upgrade and patch versions of Syncro SVN Client are released to provide enhancements that rectify problems, improve functionality and the general efficiency of the application.

This section explains the procedure for upgrading Syncro SVN Client while preserving any personal configuration settings and customizations.

Unless otherwise stated by instructions supplied with a patch or upgrade kit, the following procedure is recommended:

Procedure 2.11. Upgrade Procedure

1. Create a new folder under `../syncrosvnClient` e.g. `../syncrosvnClient/4.2`
2. Download and extract the upgrade to the new folder.
3. If you have defined Syncro SVN Client in the system PATH, modify it to the new installation folder.
4. Start Syncro SVN Client to ensure that the application can start and that your license is recognized by the upgrade installation.
5. If you are upgrading to a major version, for example from 8.3 to 9.0, then you will need to enter the new license text into the registration dialog that is shown when the application is started.
6. Select Help → About to determine the version number. If the previous version was 8.3, the About dialog should now show version 9.0.

Checking for new versions

Syncro SVN Client offers the option of checking for new versions at the <http://www.syncrosvnClient.com> [<http://www.syncrosvnClient.com/>] site when the application is started. If this option is enabled a message dialog will notify the user when new versions are released.

You can check for new versions manually at any time by going to menu Help → Check for New Versions

Uninstalling the application

Caution

The following procedure will remove Syncro SVN Client from your system. *Please ensure that all valuable data is saved to another location prior to performing this procedure.*

Procedure 2.12. Uninstall Procedure

1. Backup all valuable data from the Syncro SVN Client installation folder.
2. On Windows use the appropriate uninstaller shortcut provided with your OS.

On Mac OS X and Unix manually delete the installation folder and all its contents.

3. If you wish to completely remove the application directory and any work saved in it, you will have to delete this directory manually. To remove the application configuration and any personal customizations remove the `%APPDATA%\com.syncrosvnClient` directory on Windows (usually `%APPDATA%` has the value `[user-home-dir]\Application Data`) / `.com.syncrosvnClient` on Linux / `Library/Preferences/com.syncrosvnClient` on Mac OS X from the user home directory.

Unattended uninstall

If you want to run an unattended uninstall this is possible only on Windows and Linux by running the uninstaller executable from command line and passing the `-q` parameter. The uninstaller executable is called `uninstall.exe` on Windows and `uninstall` on Linux and is located in the `install` folder of the application.

Performance problems

Large documents

When started from the icon created on the Start menu or the Desktop on Windows and from the shortcut created on the Linux desktop the maximum memory available to Syncro SVN Client is set by default to 128 MB. When started from the command line scripts the maximum memory is 256 MB. If large documents (more than 10 MB) are edited in Syncro SVN Client and you see that performance slows down considerably after some time then a possible cause is that it needs more memory in order to run properly. You can increase the maximum amount of memory available to Syncro SVN Client by setting the `-Xmx` parameter in a configuration file specific to the platform that runs the application. For example if your file has a size of 50 MB setting a parameter `-Xmx800m` should be enough for opening and editing the file in Syncro SVN Client

Warning

The maximum amount of memory should not be equal to the physical amount of memory available on the machine because in that case the operating system and other applications will have no memory available.

When installed on a multi-user environment such as Windows Terminal Server or Unix/Linux, to each instance of Syncro SVN Client will be allocated the amount stipulated in the memory value. To avoid depreciating the general performance of the host system, please ensure that the amount of memory available is optimally apportioned for each of the expected instances.

Display problems on Linux/Solaris

Display problems like screen freeze or momentary menu pop-ups during mouse movements over screen on Linux or Solaris can be solved by specifying the parameter

```
-Dsun.java2d.pmosffscreen=false
```

for the Java virtual machine. This parameter disables off-screen pixmap support and must be added to the Java command line which starts the Java virtual machine at the end of the file *svnClient4.2* located in the install directory.

Chapter 3. The Syncro SVN Client

Introduction

What is Syncro SVN Client

Syncro SVN is a client for the Subversion version control system compatible with Subversion 1.6 servers. It manages files and directories that change over time and are stored in a central repository. The version control repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to access older versions of your files and examine the history of how and when your data changed.

Quick start guide and reference

The *Main window* section will provide a short description of the application main window layout, general functions, views and menus.

A *Getting started* chapter will take you through the basic operations, such as:

- Define a repository location
- Define a working copy
- Manage working copy resources
- Synchronize with a repository
- Obtain information for a resource
- Using the log history of a resource
- Adding and changing the properties of a resource
- Creating and maintaining branches and tags
- Some more advanced repository operations

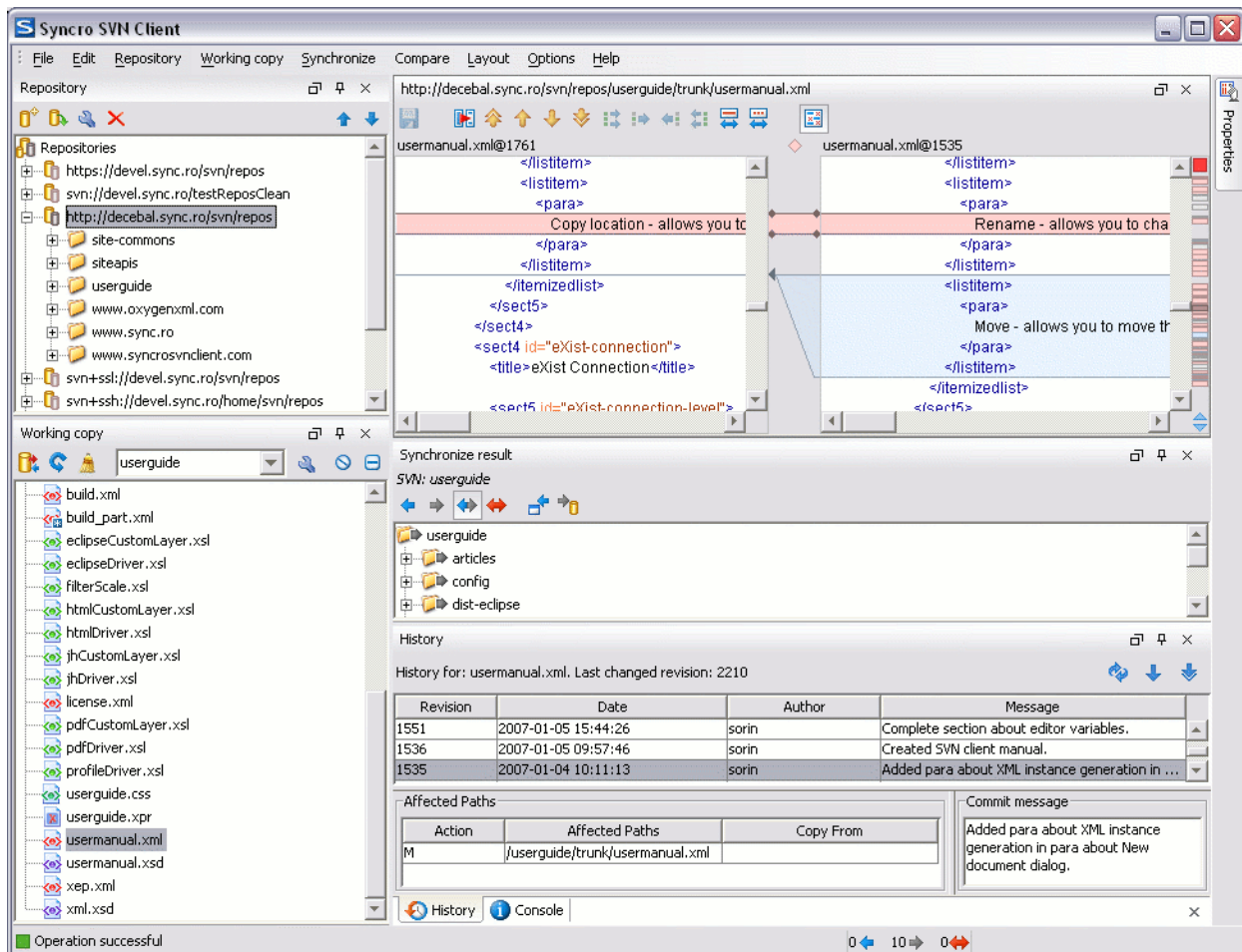
The next few chapters refer to the views of the application:

- Repository view
- Working copy view
- Synchronize view
- Compare resources view
- Editor
- Image preview
- History view
- Properties view

- Console view
- Help view
- Preferences dialog

Main window

Figure 3.1. The Syncro SVN Client main window



Starting Syncro SVN Client

The Syncro SVN Client can be used as a standalone application. To start the client follow the instructions for the installed package:

Procedure 3.1. Windows

- From the Windows Explorer double-click `syncroSVNClient.exe`.

Procedure 3.2. Linux

- At the prompt type: `sh syncroSVNClient.sh`.

Procedure 3.3. Mac OS X

- Double-click `syncroSVNClient`.

Procedure 3.4. All Platforms

- On Windows run `syncroSVNClient.bat`. On Mac OS X run `syncroSVNClient.sh`. On Linux/Unix run `syncroSVNClient.sh`.

Views


The main window consists of the following views:

















- Repository view allows you to define and manage Subversion repository locations.
- Working copy view allows you to manage with ease the content of the working copy.
- Synchronize view displays the modified resources from your working copy (outgoing) and from the repository (incoming).
- Compare view displays the differences between two revisions of a text file.
- Compare images view displays the compared images side by side.
- Editor view allows you to modify and save a file from the working copy.
- Image preview allows you to view the image files from the *Synchronize view*, *Working copy view*, *Synchronize view* or from the *History view*.
- History view displays the log messages for a given resource.
- Properties view displays the SVN properties for the currently selected resource from the *Synchronize view* or from the *Working copy view*.
- Console view shows the start and progress of an operation as if a Subversion command was run from the shell.
- Help view dynamically shows the help for the currently selected view.













The main window's *Status bar* presents in the left side the operation in progress or the final result of the last performed action. In the right side there is a progress bar for the running operation and a stop button to cancel the operation.






Main menu

The main menu of the Syncro SVN Client is composed of the following sub menus:

- File
 -  Save - Saves the local file currently opened in the *Editor view* or the *Compare view*.
 - Exit - Exits the Subversion client.

- Edit
 -  Undo - undo edit changes in the local file currently opened in the *Editor view* or the *Compare view*.
 -  Redo - redo edit changes in the local file currently opened in the *Editor view* or the *Compare view*.
 -  Cut - cut selection to clipboard from the local file currently opened in the *Editor view* or the *Compare view*.
 -  Copy - copy selection to clipboard from the local file currently opened in the *Editor view* or the *Compare view*.
 -  Paste - paste selection from clipboard in the local file currently opened in the *Editor view* or the *Compare view*.
 -  Find/Replace - perform find/replace operations in the local file currently opened in the *Editor view* or the *Compare view*.
 -  Find Next - go to the next find match using the same find options of the last find operation. The action runs in the editor panel and in any non editable text area, for example the *Console view*.
 -  Find Previous - go to the previous find match using the same find options of the last find operation. The action runs in the editor panel and in any non editable text area, for example the *Console view*.
- Repository - operations from the *Repository view*:
 -  New Repository Location - allows you to enter a new repository location by means of the *Add SVN Repository* dialog.
 -  Edit Repository Location - context dependent, allows you to edit the selected repository location by means of the *Edit SVN Repository* dialog. It is active only when a repository location root is selected.
 -  Remove Repository Location - allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.
 -  Move Up - move the selected repository up with one position in the list of repositories in the *Repository view*.
 -  Move Down - move the selected repository down with one position in the list of repositories in the *Repository view*.
- Working copy - operations from the *Working copy view*:
 -  Add/Remove Working Copy - opens the Working copies list dialog which displays the working copies the Subversion client is aware of. In this dialog you can add existing or remove no longer needed working copies.
 -  Synchronize - contacts the repository and determines the changes made by you to the working copy and by others to the repository. The synchronize result will be displayed in the *Synchronize view*. The action performs a synchronize operation on the root of the working copy.
 -  Refresh - refreshes (re scans) the content of the working copy. The action performs a refresh operation on the root of the working copy.

-  Cleanup - performs a maintenance cleanup operation on the working copy.
-  Show History ... - brings up the History view and displays the log history for the selected resource from the working copy.
- Edit conflict - opens a *Compare* view for editing the selected conflict.
-  Show SVN Properties - brings up the Properties view and displays the SVN properties for the selected file.
- Synchronize - operations from the *Synchronize view*:
 -  Update all - updates all resources with incoming changes. It is disabled when *Outgoing* mode is selected or the synchronization result does not contain resources with incoming changes. It will perform a recursive update on the synchronized resources.
 -  Commit all - commits all resources with outgoing changes. It is disabled when *Incoming* mode is selected or the synchronization result does not contain resources with outgoing changes. It will perform a recursive commit on the synchronized resources.
 - Add - it is enabled for unversioned resources and performs a *svn add* command which adds the resources to version control.
 - Commit ... - it is enabled for outgoing changes and commits all selected resources, recursively in the case of directories, to the repository. This action collects the outgoing changes from the selected resources and presents them in a dialog.
 - Update - it is enabled for resources with incoming changes. Updates all selected resources to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.
 - Override and Commit ... - it is enabled on conflicting resources. The action will drop any incoming changes and will send your local version of the resource to the repository. See also Drop incoming modifications.
 - Override and Update ... - it is enabled on resources with outgoing changes including the conflicting ones. It is used for dropping any outgoing change and replacing the local resource with the HEAD revision. See Revert your changes section.
 - Mark Resolved - it is enabled on resources with real content conflicts. Its function is to tell the Subversion system that you resolved the conflict and the resource can be committed. See also Merge conflicts part.
 -  Expand All - expands the selected directories to leaf level.
 -  Collapse All - collapses all child nodes of the selected tree node.
- Compare - operations from the *Compare view*:
 -  Perform Files Differencing - used to perform file differencing on request.
 -  Go to First Modification - used to navigate to the first difference.
 -  Go to Previous Modification - used to navigate to the previous difference.
 -  Go to Next Modification - used to navigate to the next difference.
 -  Go to Last Modification - used to navigate to the last difference.

-  Copy All Non-Conflicting Changes from Right to Left - this action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.
-  Copy Change from Right to Left - this action copies the selected change from the right editor to the left editor.
-  Show Modification Details at Word Level - because the differences are computed using a line differencing algorithm sometimes is useful to see exactly what words are different in a changed section.
-  Show Modification Details at Character Level - useful when you want to find out exactly what characters are different between the two analyzed sections.
-  Ignore Whitespaces - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before the strings are compared they are first normalized and then the whitespace at the beginning and the end of the strings is trimmed.
- Layout - layout control actions:
 - Reset Layout - resets all the views to their default position.
 - Show View - Brings to front the specified view.
- Options
 - Preferences - opens the preferences dialog.
 - Reset Options - resets all your options to the default ones.
 - Import Options - allows you to import options you have previously exported.
 - Export Options - allows you to export the current options to a file.
 - Reset Authentication - resets the Subversion authentication information.
- Help
 - Dynamic Help - shows the Dynamic Help dialog.
 - Help - opens the Help dialog.
 - Check for New Versions - checks the availability of new Syncro SVN Client versions.
 - Browse Syncro SVN Client website - opens the Syncro SVN Client website in a browser.
 - Register - opens the registration dialog.
 - Support center - opens the Syncro SVN Client Support Center web page in a browser.
 - About - opens the About dialog.

 **Note**

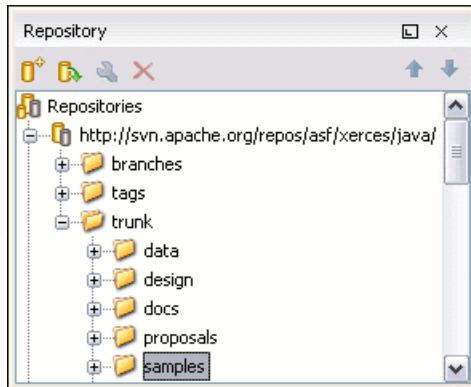
In order to avoid unusual situations you can currently execute only one action that involves operations with the working copy or with the repository at a time.

Getting started

Define a repository location

Usually team members do all of their work separately, in their own working copies and need to share their work. This is done via a Subversion repository. Syncro SVN Client supports the versions 1.3, 1.4 and 1.5 of the SVN repository format.

Figure 3.2. Repository View



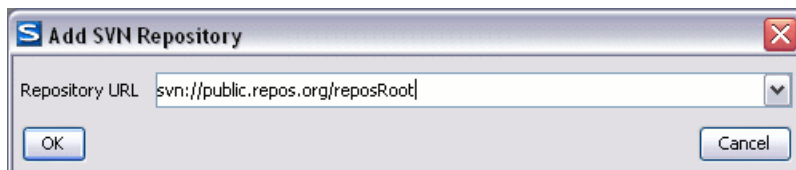
Add / Edit / Remove repository locations

Before you can begin working with a Subversion repository, you must define a repository location in the Repository View.

To create a new repository location, click the *New Repository Location* toolbar button or right click inside the view and select *New Repository Location...* from the popup menu. On Windows the context menu can be displayed with the mouse on a right click or with the keyboard by pressing the special context menu key available on Windows keyboards.

The *Add SVN Repository* dialog will prompt you for the URL of the repository you want to connect to. No authentication information is requested at the time the location is defined; it is left to the Subversion client to request the user and password information when it is needed. The main benefit of allowing Subversion to manage your password in this way is that it will prompt you for a new password only when your password changes.

Figure 3.3. The Add SVN Repository dialog



To edit a repository location, click the *Edit Repository Location* toolbar button or right click inside the view on a repository entry and select *Edit Repository Location...* from the popup menu.

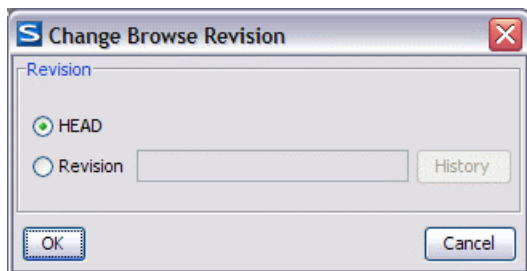
The *Edit SVN Repository* dialog works in the same way as the *Add SVN Repository* dialog. It will show the previously defined repositories URL and it will allow you to change them.

To remove a repository location, click the *Remove Repository Location* toolbar button or right click inside the view on a repository entry and select *Remove Repository Location...* from the popup menu. A confirmation dialog will appear in order to make sure you don't accidentally remove locations.

The order of the repositories can be changed in the Repository view at any time with the two buttons on the toolbar of the view, the up arrow and the down arrow. For example pressing the up arrow once moves the selected repository up in the list with one position.

To set the reference revision number of a SVN repository right-click on the repository in the list displayed in the Repository View and select the action *Change Browse Revision*.

Figure 3.4. The Change Browse Revision dialog



The revision number of the repository set with this dialog will be used for displaying the contents of the repository when it is viewed in the Repository View: only the files and folders that were present in the repository at the moment when this revision number was generated on the repository are displayed as contents of the repository tree. Also this revision number is used and for all the file open operations executed directly from the Repository View.

Authentication

Five protocols are supported: *HTTP*, *SVN*, *HTTPS*, *SVN + SSH* and *FILE*. If the repository that you are trying to access is password protected, the *Enter authentication data* dialog will request a username and a password. If the *Store authentication data* checkbox is checked the credentials will be stored in Subversion's default directory:

- Windows - %HOME%\Application Data\Subversion\auth. Example: C:\Documents and Settings\John\Application Data\Subversion\auth
- Linux & Mac OS X - \$HOME/.subversion/auth. Example: /home/John/.subversion/auth

There will be one file for each server that you access. If you want to make Subversion forget your credentials, you can use the *Reset authentication* command from the Options menu. This will cause Subversion to forget all your credentials.



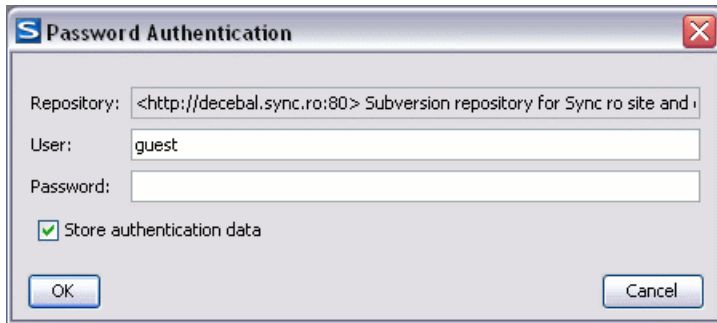
Note

When you reset the authentication data, you will have to restart the application in order for the change to take effect.

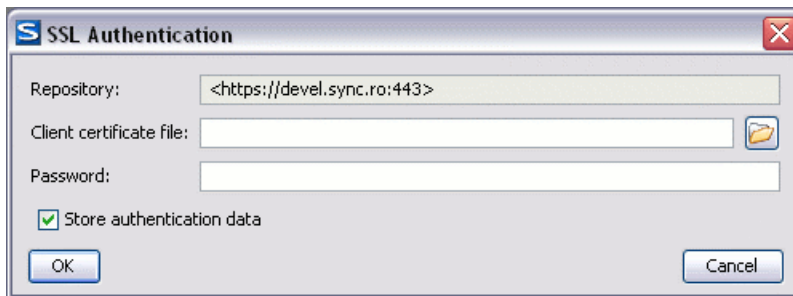


Tip

The *FILE* protocol is recommended if the *SVN* server and Syncro *SVN* Client are located on the same computer as it ensures faster access to the *SVN* server than the other protocols.

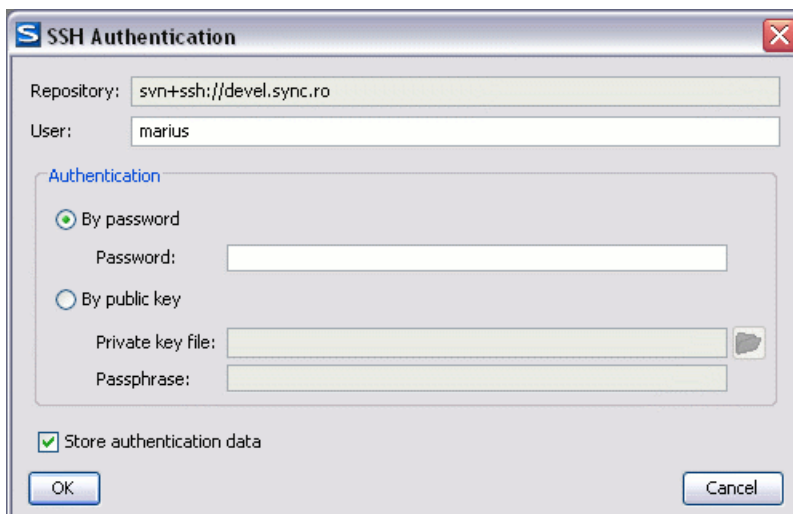
Figure 3.5. User & Password authentication dialog

For https connections where client authentication is required by your SSL server, you have to choose the Certificate File and enter the corresponding Certificate Password which is used to protect your certificate.

Figure 3.6. SSL authentication dialog

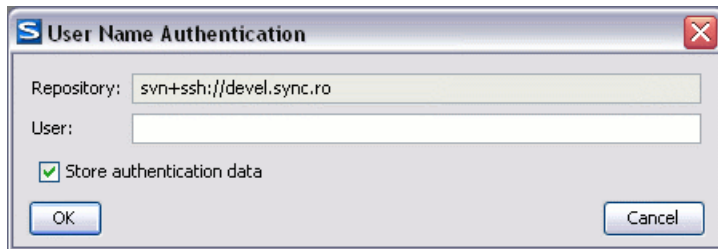
When using a secure http (https) protocol for accessing a repository, a *Certificate information* dialog will pop up and ask you whether you accept the certificate permanently, temporarily or simply deny it.

If the repository used has svn+ssh protocol the SSH authentication can also be made with a private key and a pass phrase.

Figure 3.7. User & Private key authentication dialog

After the SSH authentication dialog another dialog will pop up for entering the SVN user name that will access the SVN repository and will be recorded as the committer in SVN operations.

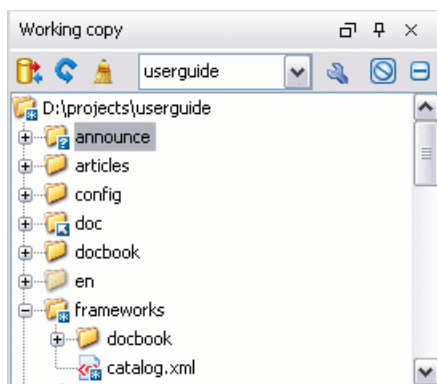
Figure 3.8. SVN user authentication dialog



Defining a working copy

A Subversion working copy is an ordinary directory tree on your local system, containing a collection of files. You can edit these files however you wish, your working copy being your private work area. In order to make your own changes available to others or incorporate other people's changes, you must explicitly tell Subversion to do so. You can even have multiple working copies of the same project.

Figure 3.9. Working Copy View



A Subversion working copy also contains some extra files, created and maintained by Subversion, to help it keep track of your files. In particular, each directory in your working copy contains a subdirectory named `.svn`, also known as the working copy *administrative directory*. This administrative directory contains an unaltered copy of the last updated files from the repository. This copy is usually referred to as the *pristine copy* or the *BASE revision* of the working copy. These files help Subversion recognize which files contain unpublished changes, and which files are out-of-date with respect to others' work.

A typical Subversion repository often holds the files (or source code) for several projects; usually, each project is a subdirectory in the repository's file system tree. In this arrangement, a user's working copy will usually correspond to a particular subtree of the repository.

Check out a working copy

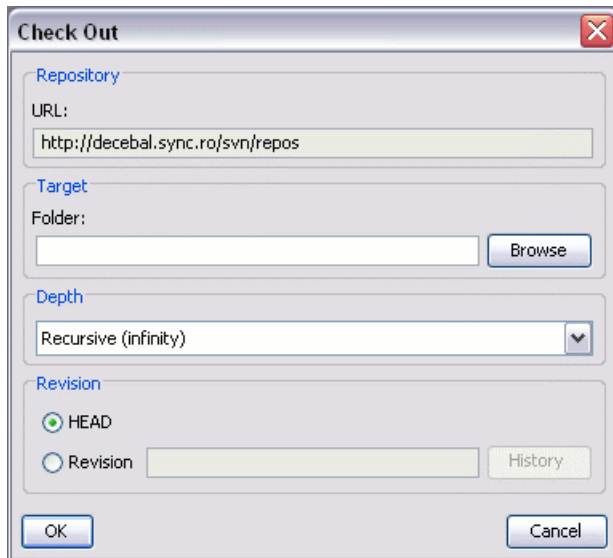
Check out is the term used to describe the process of making a copy of a project from a repository into your local file system. This checked out copy is called a working copy. A Subversion working copy is a specially formatted folder structure which contains additional `.svn` folders that store Subversion information, as well as a pristine copy of each item that is checked out.

You check out a working copy from the Repository View. If you have not yet defined a connection to your repository, you will need to add a new repository location.

To check out a new working copy, navigate inside the repository to the desired directory, right click on it and select *Check Out...* from the popup menu.

In the *Check out* dialog click on the *Browse* button and choose the location where the working copy will be checked out.

Figure 3.10. Check out dialog



After a check out, the new working copy will be added to the list in the Working Copy view and its content will be displayed in that view.

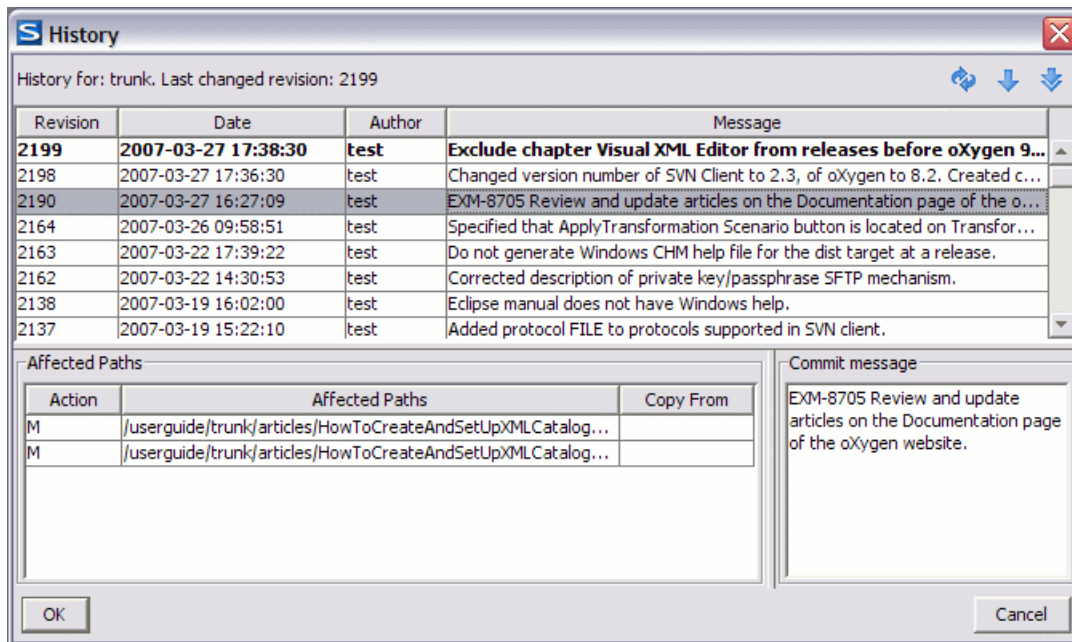
Depth

You can choose the *depth* for the checkout folder. This allows you to specify the recursion level into children. This is used if you want to check out only a portion of an working copy and then bring in a future update operation previously ignored files and subdirectories. You can find out more about checkout depth in the sparse checkouts section.

Revision

By default the last (HEAD) revision will be checked out. If you need another revision you can select the *Revision* radio button and then click on the History button and choose a desired revision from the new dialog. Or you can simply type the revision number in the corresponding text field.

Figure 3.11. History dialog



The *History dialog* presents a list of revisions for a resource. There are presented information about revision, commit date, author and commit comment. The initial number of entries in the list is 50. Additional revisions can be added to the list using the Get next 50 and Get all buttons. The list of revisions can be refreshed at any time with the Refresh button.

The *Affected Paths* area displays all paths affected by the commit of the revision selected in history. On a revision selected in the *Affected Paths* area the contextual menu contains the actions:

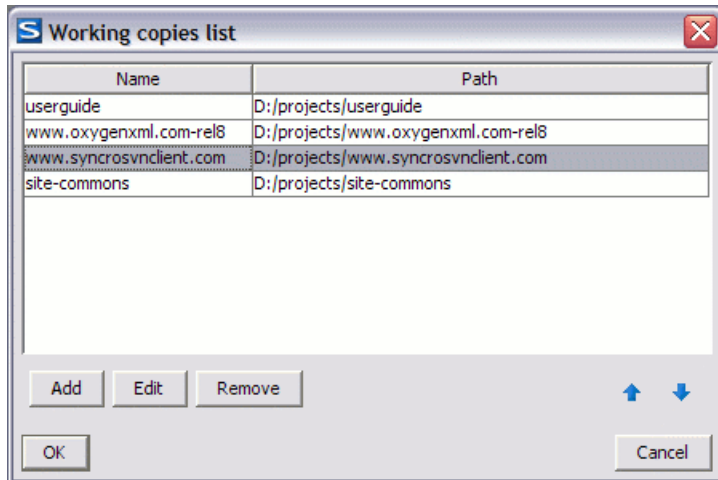
Open	Opens the revision in the editor panel.
Save revision to ...	Save the revision to a new file.
Compare with previous version	Make a diff between the selected revision and the previous one. If there is no external application specified for executing diff operations the built-in diff tool is applied. This is the action also executed on double clicking on a file in the <i>Affected Paths</i> area.
Update to revision	Make the selected revision the current revision in the working copy.
Revert changes from this revision	The changes committed by the selected revision are reverted in the current version of the file in the working copy. If the committed changes were in fact a SVN delete operation the result is restoring the deleted file in the working copy.
Show History	Display the history of the SVN resource of the selected revision.
Show Annotation	Open the Annotations View for the selected revision.

Use an existing working copy

This is the process of taking an working copy that exists on your file system and connecting it to Subversion. If you have a brand new project that you want to import into your repository, then see the section [Import resources into the repository](#)

This assumes that you have an existing valid working copy on your file system. In the Working Copy View click on the *Add/Remove Working Copy* toolbar button.

Figure 3.12. Add/Remove Working Copy dialog



In the *Working copies list* dialog press the Add button and choose the working folder copy from the file system.

Select the new working copy from the list and press the OK button. The selected working copy will be loaded and presented in the Working Copy View.

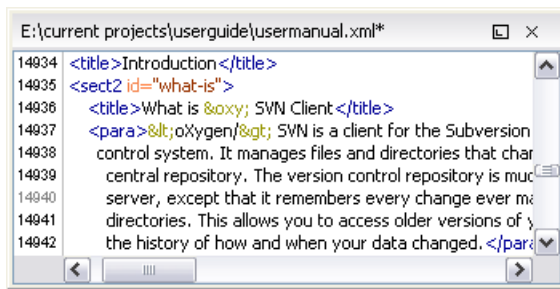
The Edit button allows changing the name of the working copy. The name is useful to differentiate between working copies located in folders with the same name. The default name is the name of the root folder of the working copy.

The order of the working copies can be changed in the list using the two arrow buttons which move the selected working copy with one position up or down.

Manage working copy resources

Edit files

You can edit files from the Working Copy View by double clicking them or by right clicking them and choosing *Open* from the popup menu, or from the Synchronize View by using *Open* from the popup menu. Please note that only one file can be edited at a time; if you try to open another file it will be opened in the same editor window. The editor has syntax highlighting for known file types, meaning that a different color will be used for each type of recognized token in the file. If the selected file is an image, then it will be previewed in the editor, with no access to modifying it.

Figure 3.13. Editor View

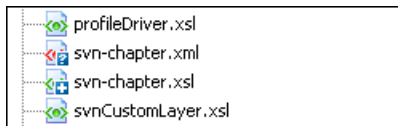
When you edit a file from your working copy, you will notice that after modifying and saving it, a modified marker - an asterisk(*) - will appear on the file's icon in the Working Copy View.

Add resources to version control

The new file(s) or folder(s) you create during your development process must be added to Version Control, using the *Add* command from the context menu in Working Copy View or Synchronize View. If you do not do this, the resource will be marked with a question mark (?), meaning that it is unversioned (unknown). After you have added it to version control, the resource will be marked as added(+) which means you first have to commit your working copy to make those resources available to other developers. Adding a resource to version control does not affect the repository.

If you try to add to version control an unversioned directory the entire subtree starting with that directory will be added.

When you commit your changes, if you forgot to add a resource, it will still be presented in the commit dialog, but will be de-selected by default. When you commit the unversioned resource, it will be automatically added to version control before being committed and the marking will also be removed.

Figure 3.14. Unversioned / Added

Ignore resources not under version control

Sometimes you will have files and folders inside your working copy that should not be subject to version control. These might include files created by the compiler, *.obj, *.class, *.lst, maybe an output directory used to store the executable. Whenever you commit changes, Subversion shows your modified files but also the unversioned files, which fills up the file list in the commit dialog. Though the unversioned files will not be committed unless otherwise specified, it is difficult to see exactly what you are committing.

The best way to avoid these problems is to add the derived files to the Subversion's ignore list. That way they will never show up in the commit dialog and only genuine unversioned files which must be committed will be shown.

You can choose to ignore a resource by using the *Add to svn:ignore* action in the context menu from Working Copy View or Synchronize View.

In the *Add to svn:ignore* dialog you can specify the resource to be ignored by name or by a custom pattern. The custom pattern can contain wildcard characters such as:

- * - Matches any string of characters of any size, including the empty string.

- ? - Matches any single character.

For example you may choose to ignore all text documents by using the pattern: **.txt*

The action adds a predefined Subversion property called *svn:ignore* to the parent directory of the specified resource. In this property there are specified all the child resources of that directory that must be ignored. The result will be visible in the *Working Copy view*. The ignored resources will be represented with grayed icons.

Delete resources

The delete command can be found in the *Edit* submenu of the context menu from the Working Copy View.

When you delete a resource from the Subversion working copy it will be removed from the file system and it will be also marked as deleted. If unversioned, added or modified resources will be encountered, a dialog will prompt you to confirm their deletion.

The delete command will not delete from the file system the directories under version control, it will only mark them as deleted. This is because the directories also contain the pristine copy of that directory content. In the Working Copy View this is transparent as all resources will have the deleted mark(-). The directories will be removed from the file system when you commit them to the repository. You can also change your mind completely and revert the deleted files to their initial, pristine state.

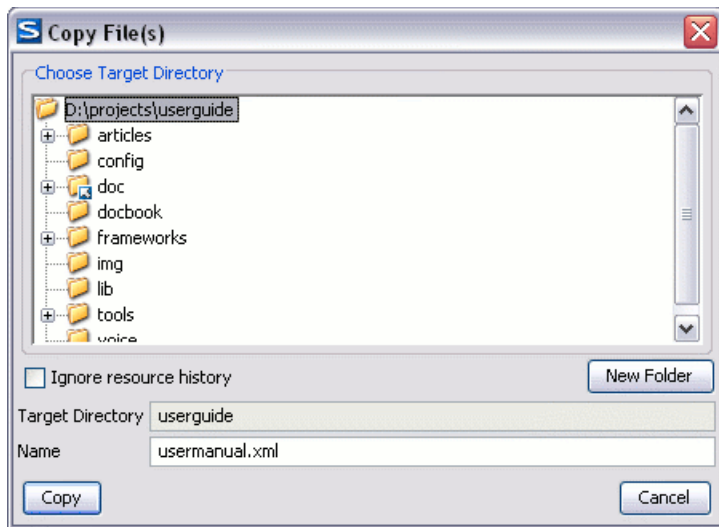
If a resource is deleted from the file system without Subversion's knowledge, your working copy will be in an inconsistent state. The resource will be considered and marked as missing (!). If a file was deleted, it will be treated in the same way as if it was deleted by Subversion. However if a directory is missing you will be unable to commit. If you update your working copy, Subversion will replace the missing directory with the latest version from the repository and you can then delete it the correct way using the *Delete* command. The *Delete* action is not enabled when the selection contains *missing* resources.

Copy / Move / Rename resources

Copy resources

You can copy several resources from different locations of the working copy. You select them in the Working Copy View and then you initiate the copy command from the context menu. This is not a simple file system copy but a Subversion command. It will copy the resource and the copy will also have the original resource's history. This is one of Subversion's very important features, as you can keep track of where the copied resources originated.

Please note that you can only copy resources that are under version control and are committed to the repository or unversioned resources. You cannot copy resources that are added but not yet committed.

Figure 3.15. Copy files dialog

In the *Copy File(s)* dialog you can navigate through the working copy directories in order to choose a target directory. If you try to copy a single resource you are also able to change that resource's name in the corresponding text field.

If an entire directory is copied the *Override and update* action will be enabled only for it and not for its descendants. In the Synchronize view and the *Commit dialog* will appear only the directory in question without its children.

Move resources

As in the case of the copy command you can perform the operation on several resources at once. Just select the resources in the Working Copy View and choose the *Move* command from the context menu. The move command actually behaves as if a copy followed by a delete command were issued. You will find the moved resources at the desired destination and also at their original location but marked as deleted.

Rename a resource

The rename action can be found in the context menu in the Working Copy View. This action can only be performed on a single resource. The rename command acts as a move command with the destination being the same as the original location of the resource. A copy of the original resource will be made with the new name and the original will be marked as deleted.

Note

Because the rename and move commands act as a copy followed by delete, when you want to commit a renamed or moved resource you must also commit the deleted original. It is also recommended that you commit the renamed or moved resources before changing their contents in order to avoid difficulties in resolving conflicts.

Lock / Unlock resources

The idea of version control is based on the copy-modify-merge model of file sharing. This model states that each user contacts the repository and creates a local working copy(check out). Users can then work independently and make modifications to their working copies as they please. When their goal has been accomplished it is time for the users to share their work with the others, to send them to the repository(commit). When a user has modified a file that has been also modified on the repository the two files will have to be merged. The version control system assists the user with the merging as much as it can, but in the end the user is the one that must make sure it is done correctly.

The copy-modify-merge model only works when files are contextually mergeable: this is usually the case of line-based text files (such as source code). However this is not always possible with binary formats, such as images or sounds. In these situations, the users must each have exclusive access to the file, ending up with a lock-modify-unlock model. Without this, one or more users could end up wasting time on changes that cannot be merged.

A Subversion lock is a piece of metadata which grants exclusive access to a user. This user is called the lock owner. A lock is uniquely identified by a lock token (a string of characters). If someone else attempts to commit the file (or delete a parent of the file), the repository will demand two pieces of information:

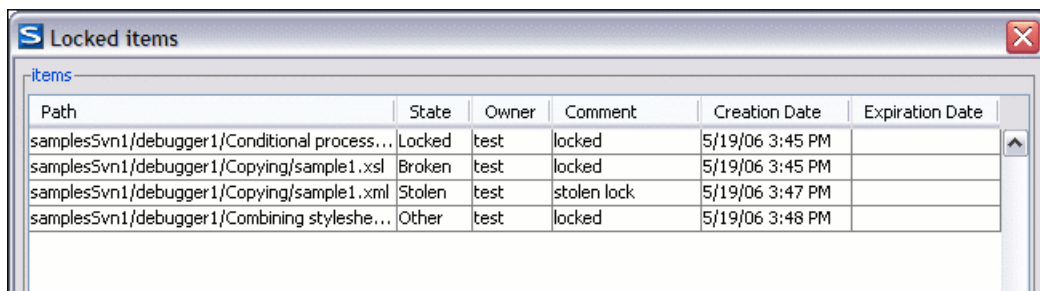
- User authentication. The user performing the commit must be the lock owner.
- Software authorization. The user's working copy must have the same lock token as the one from the repository, proving that it is the same working copy where the lock originated from.

Scanning for locks

When starting to work on a file that is not contextually mergeable (usually a binary file), it is better to verify if someone else isn't already working on that file. You can do this in the Working Copy View by selecting one or more resources, then right clicking on them and choosing the *Scan for locks* action from the context menu.

Locked items

Figure 3.16. The locked items dialog



Path	State	Owner	Comment	Creation Date	Expiration Date
samplesSvn1/debugger1/Conditional process...	Locked	test	locked	5/19/06 3:45 PM	
samplesSvn1/debugger1/Copying/sample1.xml	Broken	test	locked	5/19/06 3:45 PM	
samplesSvn1/debugger1/Copying/sample1.xml	Stolen	test	stolen lock	5/19/06 3:47 PM	
samplesSvn1/debugger1/Combining styleshe...	Other	test	locked	5/19/06 3:48 PM	

The *Locked items* dialog contains a table with all the resources that were found locked on the repository. For each resource there are specified: resource path, state of the lock, owner of the lock, lock comment, creation and expiration date for the lock (if any).

The state of the lock can be one of:

- Other - if someone else locked the file.
- Locked - if you locked the file.
- Broken - if you locked the file but it was forcefully unlocked by someone else afterwards.
- Stolen - if you locked the file but it was forcefully locked by someone else afterwards.

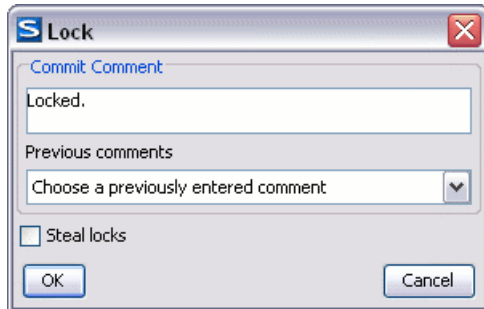
You can unlock a resource by selecting it and pressing the Unlock button.

Locking a file

A locked file allows you exclusive write access to a file from the repository, meaning that you are the only one who can modify and commit the file to the repository.

You can lock a file from the context menu in Working Copy View. Note that you can only lock several files at once but no directories. This is a restriction of Subversion which is used to discourage the use of the lock-modify-unlock model at large scale or when unnecessary.

Figure 3.17. The lock dialog



In the *Lock* dialog you can write a comment for the lock and if necessary steal (force) the lock. Note that you should only steal a lock after you made sure that the previous owner no longer needs it, otherwise you may cause an unsolvable conflict which is exactly why the lock was put there in the first place. The Subversion server can have a policy concerning lock stealing, it may not allow you to steal a lock if a certain condition is not satisfied.

The lock will stay in place until you commit the locked file or until someone unlocks it. There is also the possibility that the lock will expire after a period of time specified in the Subversion server policy.

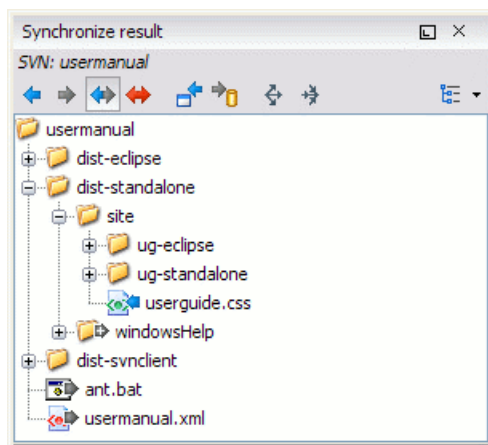
Unlocking a file

A file can be unlocked from the context menu in the Working Copy View. A dialog will prompt you to confirm the unlocking and it will also allow you to break the lock (unlock it by force).

Synchronize with the repository

In the work cycle you will need to incorporate other people's changes(update) and to make your own work available to others(commit). This is what the Synchronize View was designed for, to help you send and receive modifications from the repository.

Figure 3.18. Synchronize View



In the *Synchronize view* you can see the overall status of your working copy resources when compared to the repository resources. The view focuses on incoming and outgoing changes, where incoming changes are the changes that other users have committed since you last updated your working copy. The outgoing changes are the modifications you made to your working copy as a result of editing, removing or adding resources.

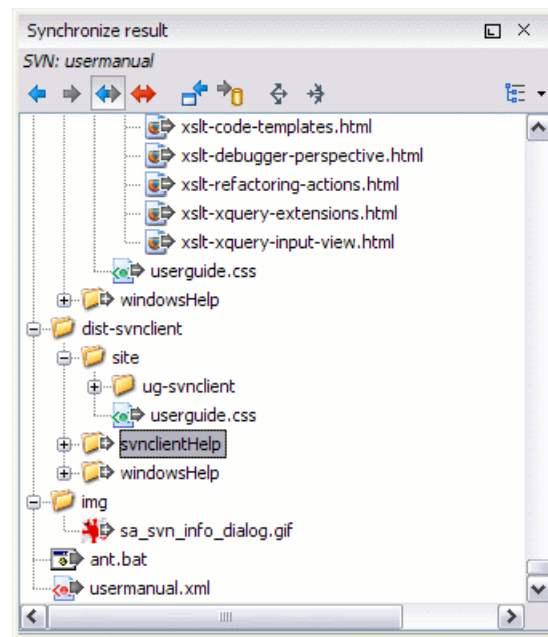
The view presents the status of the working copy resources against the BASE revision after a *Refresh* operation. You can view the state of the resources versus a repository HEAD revision by using the *Synchronize* actions from the Working Copy view.

Presentation modes

The *Synchronize view* has three presentation modes:

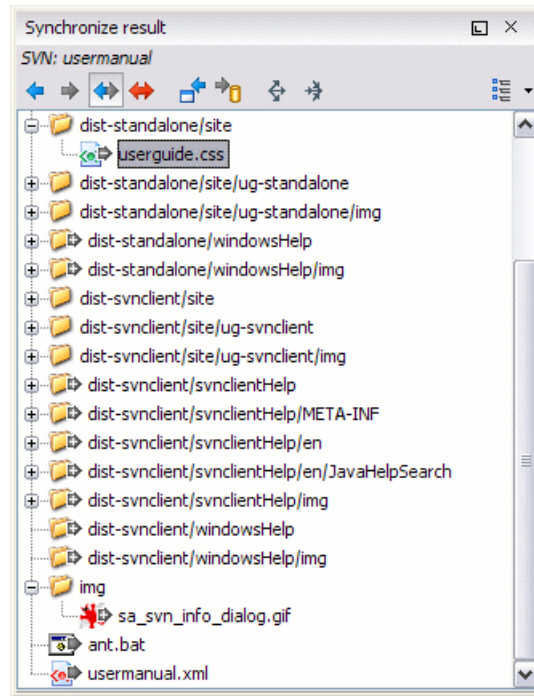
Tree mode The resources are presented in a tree layout as in the above image which mirrors the tree structure of the SVN repository and of the Working Copy view. This mode is more appropriate when you want a quick overview of the locations which need synchronization with the SVN repository or when you want to apply a synchronization operation (Commit, Update, Revert, Add) recursively on a folder.

Figure 3.19. Synchronize View - Tree mode



Compressed mode The resources are presented in a layout with two levels, that is a compressed path for each folder in the list as in the following image. This mode is useful when you need the full list of resources which need synchronization without having to expand a tree to get to the unsynchronized resources of that folder. Also it is useful when you do not want to apply a synchronization operation recursively, that is the operation applied to a folder resource must not have any effect on other unsynchronized resources located in the folder but displayed in other list entries in the view.

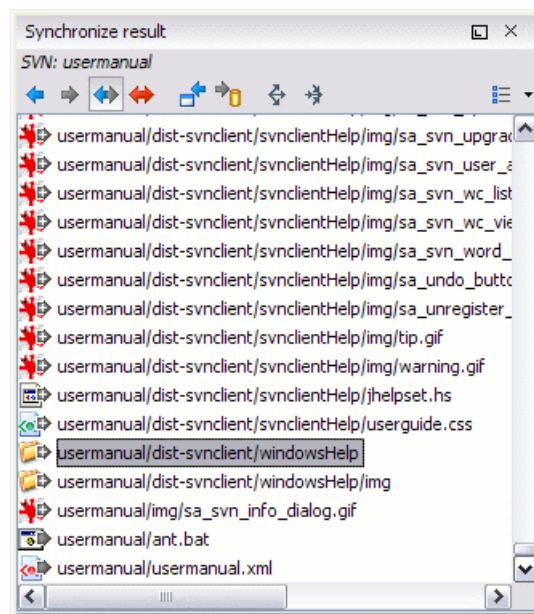
Figure 3.20. Synchronize View - Compressed mode



Flat mode

The full list of the resources that must be synchronized with the repository are presented in a flat list. As in the Compressed mode it is useful when you do not want to apply a synchronization operation recursively on a folder.

Figure 3.21. Synchronize View - Flat mode



Switching between the three presentation modes is done with the switch button on the right side of the toolbar of the *Synchronize view*.

View differences

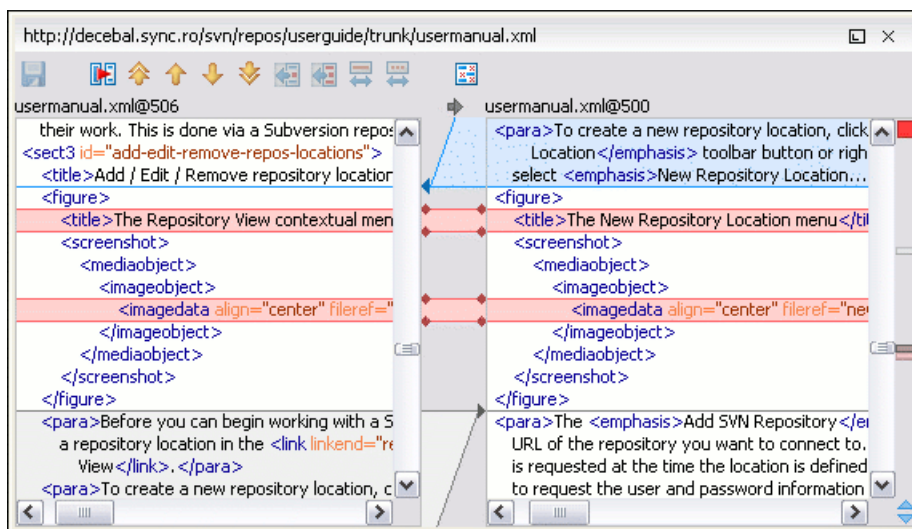
One of the most common requirements in project development is to see what changes have been made to the files from your Working Copy or to the files from the repository. You can examine these changes after a synchronize operation with the repository, by using the *Open in compare editor* action from the contextual menu.

The text files are compared using a built-in Compare View which uses a line differencing algorithm or a specified external diff application if such an application is set in the SVN preferences. When a file with outgoing status is involved, the compare is performed between the file from the working copy and the BASE revision of the file. When a file with incoming or conflict status is involved, the differences are computed using a three-way algorithm which means that the local file and the repository file are each compared with the BASE revision of the file. The results are displayed in the same view. The differences obtained from the local file comparison are considered outgoing changes and the ones obtained from the repository file comparison are considered incoming changes. If any of the incoming changes overlap outgoing changes then they are in conflict.

A special case of difference is a *diff pseudo-conflict*. This is the case when the left and the right sections are identical but the BASE revision does not contain the changes in that section. By default this type of changes are ignored. If you want to change this you can go to SVN Preferences and change the corresponding option.

The right editor of the internal compare view presents either the BASE revision or a revision from the repository of the file so its content cannot be modified. By default when opening a synchronized file in the Compare View, a compare is automatically performed. After modifying and saving the content of the local file presented in the left editor, another compare is performed. You will also see the new refreshed status in the Working copy view.

Figure 3.22. Compare View



There are three types of differences:

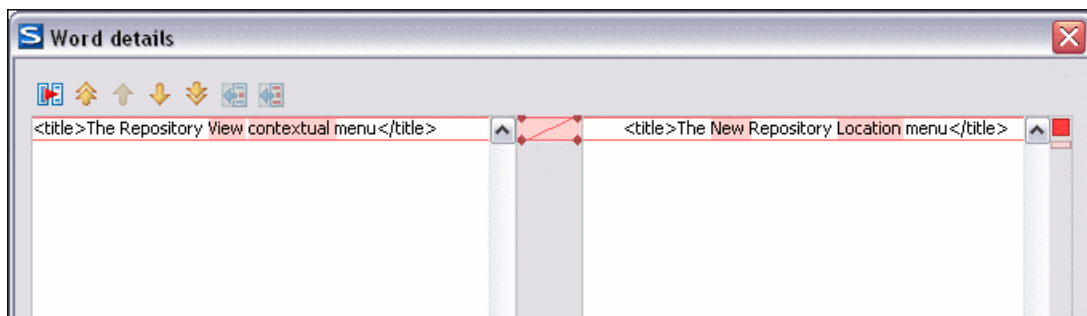
- incoming changes - changes committed by other users and not present yet in your working copy file. They are marked with a blue highlight and on the middle divider the arrows point from right to left.
- outgoing changes - changes you have done in the content of the working copy file. They are marked with a gray highlight and the arrows on the divider are pointing from left to right.

- conflicting changes - this is the case when the same section of text which you already modified in the local file has been modified and committed by some other person. They are marked with a red highlight and red diamonds on the divider.

There are numerous actions and options available in the Compare View toolbar or in the Compare menu from the main menu. You can decide that some changes need adjusting or that new ones must be made. After you perform the adjustments, you may want to perform a new compare between the files. For this case there is an action called *Perform files differencing*. After each files differencing operation the first found change will be selected. You can navigate from one change to another by using the actions *Go to first / Go to previous / Go to next / Go to last modification*. If you decide that some incoming change needs to be present in your working file you can use the action *Copy change from right to left*. This is useful also when you want to override the outgoing modifications contained in a conflicting section. The *Copy all non-conflicting changes from right to left* copies all incoming changes which are not contained inside a conflicting section in your local file.

Let us assume that only a few words or letters are changed, considering that the differences are performed taking into account whole lines of text, the change will contain all the lines involved. For finding exactly what words or letters have changed there are available two dialogs which present a more detailed compare result: *Word Details* and *Character Details*.

Figure 3.23. Word details dialog



When you want to examine only the changes in the real text content of the files disregarding the changes in the number of white spaces between words or lines there is available an option which allows you to enable or disable the white space ignoring feature of the compare algorithm.

Resolve conflicts

Once in a while, you will get file conflicts when you update your files from the repository. A file conflict occurs when two or more developers have changed the same few lines of a file or the properties of the same file. As Subversion knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and try to analyse and resolve the conflicting situation.

Real conflicts vs mergeable conflicts

There are two types of conflicts. The real conflict (conflicted state) is obtained when a file in the working copy has incoming and outgoing changes in the same section. When updated the differences cannot be merged automatically so the file is marked as conflicted. A file can be in real conflict state when its content or its properties are in conflict. A folder can be in real conflict only when its properties are in conflict.

A file is in a mergeable conflict state when it contains both incoming and outgoing changes not necessarily in the same sections. A file is in mergeable conflict when its content has both incoming and outgoing changes but the changes can be merged by the update operation. A folder can be in mergeable conflict when it contains files in mergeable conflict and / or real conflict themselves. After an update it is possible that the state of conflict can be resolved automatically

by merging the incoming changes into the working copy resource. A conflicting resource cannot be committed. In the conflict case the resource will be marked with a conflict icon and will appear in all the Synchronization trees.

Content conflicts vs Property conflicts

On the other hand depending on the situation the conflicts are separated in two categories: Content conflicts and Properties conflicts. *Content conflicts* - this type refers to the fact that the conflict appears in the content of a file. A merge occurs for every inbound change to a file which is also modified in the working copy. In some cases, if the local change and the incoming change intersect each other, Subversion cannot merge these changes without intervention. So if the conflict is real when updating the file in question the conflicting area is marked like this:

```
<<<<<< filename
your changes
=====
code merged from repository
>>>>>> revision
```

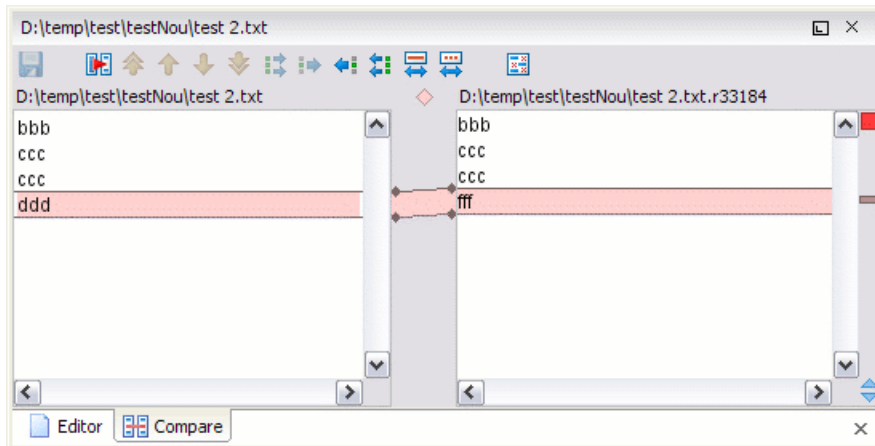
Also, for every conflicted file Subversion places three additional temporary files in your directory:

- `filename.ext.mine` - This is your file as it existed in your working copy before you updated your working copy - that is, without conflict markers. This file has your latest changes in it and nothing else.
- `filename.ext.rOLDREV` - This is the file that was the BASE revision before you updated your working copy. That is, the file revision that you updated before you made your latest edits.
- `filename.ext.rNEWREV` - This is the file that Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD revision of the repository.

OLDREV and NEWREV are revision numbers. If you have conflicts with binary files, Subversion does not attempt to merge the files by itself. The local file remains unchanged (exactly as you last changed it) and you will get `filename.ext.r*` files also. *Properties conflicts* - refer to the conflicts that are obtained when two people modify the same property of the same file or folder. When updating such a resource a file named `filename.ext.prej` is created in your working copy containing the nature of the conflict. Your local file property that is in conflict will not be changed. After resolving the conflict one should use the *Mark resolved* action in order to be able to commit the file. Note that the *Mark resolved* action does not really resolve the conflict. It just removes the conflicted flag of the file and deletes the temporary files.

Edit real content conflicts

The conflicts of a file in the conflicted state (a file with the red double arrow icon) can be edited visually with the *Compare* view (the built-in file diff tool) or with an external diff application to decide for each conflict if the local version of the change will remain or the remote one instead of the special conflict markers inserted in the file by the SVN server.

Figure 3.24. Compare view for editing a conflict

The *Compare* view (or the external diff application set in Preferences) is opened with the action *Edit Conflict* which is available on the context menus of the Synchronize view and the Working Copy view and is enabled only for files in the conflicted state (an update operation was executed but the differences could not be merged without conflicts). The external diff application is called with 3 parameters because it is a 3-way diff operation between the local version of the file from the working copy and the HEAD version from the SVN repository with the BASE version from the working copy as common ancestor.

If the option *Show warning dialog when edit conflicts* is enabled you will be warned at the beginning of the operation that the operation will overwrite the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<<<<, =====, >>>>>>>) with the original local version of the file that preceded the update operation. If you press the OK button the visual conflict editing will proceed and a backup file of the conflict version received from the SVN server is created in the same working copy folder as the file with the edited conflicts. The name of the backup file is obtained by appending the extension *.sync.bak* to the file as stored on the SVN server. If you press the Cancel button the visual editing will be aborted.

The usual operations on the differences between two versions of a file are available on the toolbar of this view:

Save	Save the modifications of the local version of the file displayed in the left side of the view.
Perform Files Differencing	Apply the diff operation on the two versions of the file displayed in the view. It is useful after modifying the local version displayed in the left side of the view.
Go to First Modification	Scroll the view to the topmost difference.
Go to Previous Modification	Scroll the view to the previous difference. The current difference is painted with a darker color than the other ones.
Go to Next Modification	Scroll the view to the next difference. The current difference is painted with a darker color than the other ones.
Go to Last Modification	Scroll the view to the last difference.
Copy All Non Conflicting Changes from Left to Right	Not applicable for editing conflicts so it is disabled.
Copy Change from Left to Right	Not applicable for editing conflicts so it is disabled.

Copy Change from Right to Left	Copy the current difference from the left side to the right side by replacing the highlighted text of the current difference from the left side with the one from the right side.
Copy All Non Conflicting Changes from Right to Left	Apply the previous operation for all the differences.
Show Modification Details at Word Level	Display a more detailed version of the current difference computed at word level.
Show Modification Details at Char Level	Display a more detailed version of the current difference computed at character level.
Ignore Whitespaces	The text nodes are normalized before computing the difference so that if two text nodes differ only in whitespace characters they are reported as equal.

The operation begins by overwriting the conflict version of the file received from the SVN server (the version which contains the conflict markers <<<<<<<, =====, >>>>>>>) with the original local version of the file before running the update action which created the conflict. After that the differences between this original local version and the repository version are displayed in the *Compare* view.

If you want to edit the conflict version of the file directly in a text editor instead of the visual editing offered by the *Compare* view you should work on the local working copy file after the update operation without running the action *Edit Conflict*. If you decide that you want to edit the conflict version directly after running the action *Edit Conflict* you have to work on the .sync.bak file.

If you did not finish editing the conflicts in a file at the first run of the action *Edit Conflict* you can run the action again and you will be prompted to choose between resuming the editing where the previous run left it and starting again from the conflict file received from the SVN server.

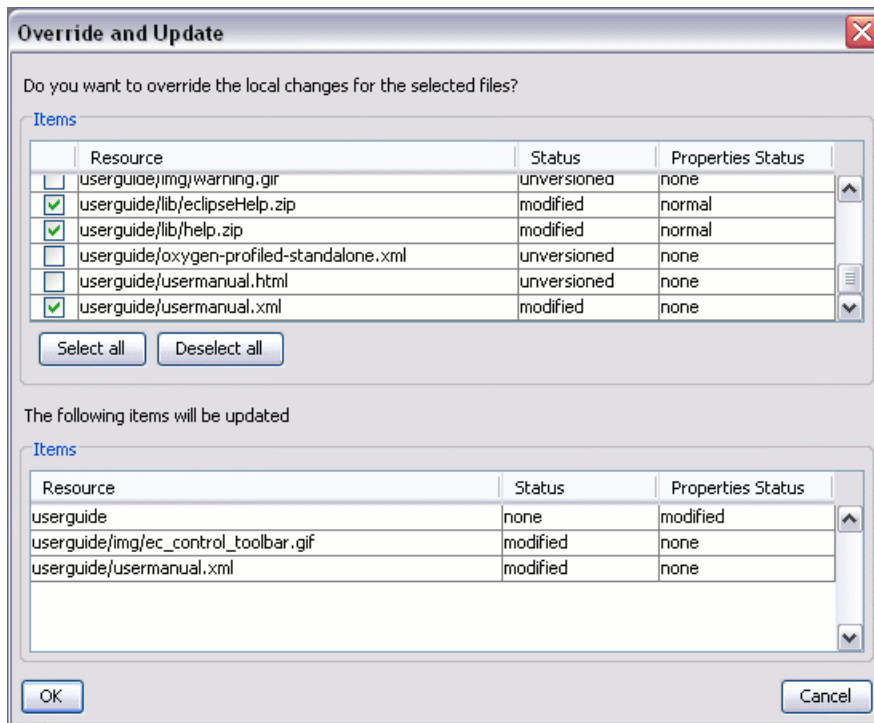
After the conflicts are edited and saved in the local version of the file you usually run the action *Mark Resolved* on the file so that the result of the conflict editing process can be committed to the SVN repository or the action *Revert* so that the repository version overwrites all the local modifications. Both actions remove the backup file and other temporary files created with the conflict version of the local file.

Revert your changes

If you want to undo all changes you made in a file since the last update you need to select the file, right click to pop up the context menu and then select Revert. A dialog will pop up showing you the files that you have changed and can be reverted. Select those you want to revert and click the OK button. Revert will only undo your local changes. It does not undo any changes which have already been committed. If you choose to revert the file to the pristine copy which resides in the administration folders then the eventual conflict is solved by losing your outgoing modifications. If you try to revert a resource not under version control, the resource will be deleted from the file system.

If you want some of your outgoing changes to be overridden you must first open the file in Compare view and choose the sections to be replaced with ones from repository file. This can be achieved either by editing directly the file or by using the action *Copy change from right to left* from Compare view toolbar. After editing the conflicting file you have to use *Mark as merged* before committing it.

If you want to drop all local changes and in the same time bring all incoming changes into your working copy resource you can use the *Override and update* action which discards the changes in the local file and updates it from the repository. A dialog will show you the files that will be affected.

Figure 3.25. Override and update dialog

In the first table in the dialog you will be able to see the resources that will be overridden. You can also select or deselect them as you wish. In the second table you will find the list of resources that will be updated. Only resources that have an incoming status in the *Synchronize view* will be updated.

Merge conflicted resources

Before you can safely commit your changes to the repository you must first resolve all conflicts. In the case of pseudo-conflicts they can be resolved in most cases with an Update operation which will merge the incoming modifications into your working copy resource. In the case of real conflicts, conflicts that persist after an update operation, it is necessary to resolve the conflict using the built-in compare view and editor or, in the case of properties conflict, the Properties view. Before you can commit you must *mark as resolved* the affected files. Both pseudo and real conflicts can be resolved without an update. You can:

- open the file in the compare editor
- analyze the changes
- edit the changes
- decide which incoming changes need to be copied locally
- decide which outgoing changes must be overridden or modified

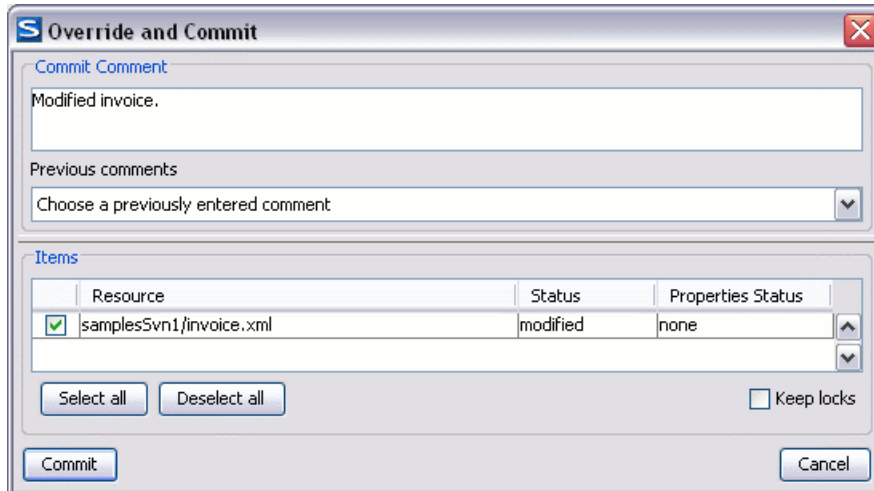
After saving your local file you have to use the *Mark as merged* action from the contextual menu before committing.

Drop incoming modifications

In the situation when your file is in conflict but you decide that your working copy file and its content is the correct one, you can decide to drop some or all of the incoming changes and commit afterwards. The action *Mark as merged*

proves to be useful in this case too. After opening the conflicting files with Compare view, Editor or editing their properties in the *Properties view* and deciding that your file can be committed in the repository replacing the existing one, you should first use *Mark as merged* action. When you want to override completely the remote file with the local file you can use *Override and commit* which drops any remote changes and commits your file.

Figure 3.26. Override and commit dialog

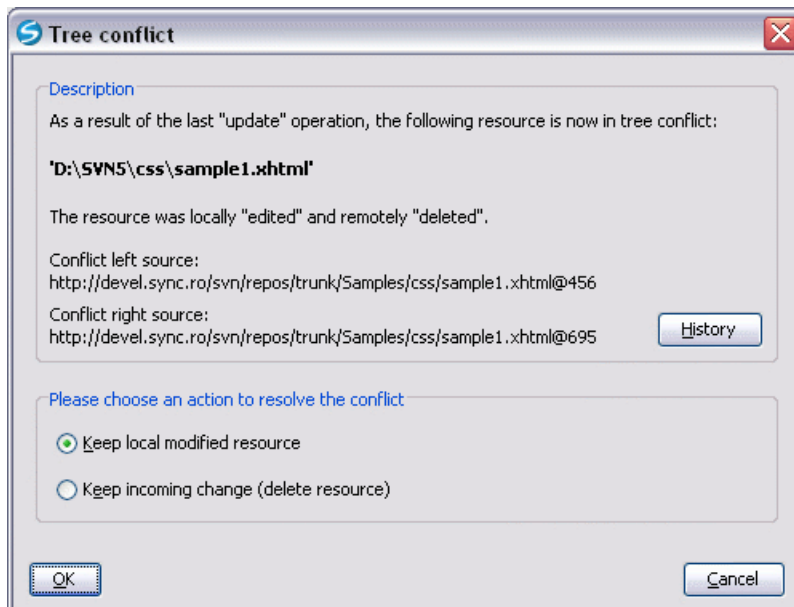


In general it is much safer to analyze all incoming and outgoing changes using Compare view and only after to update and commit.

Tree conflicts

A tree conflict is a conflict at the folder level and occurs when the user runs an update action on a file but the file does not exist in the repository anymore because other user renamed the file, moved the file to other folder or deleted the file from repository. The same conflict situation can occur at folder level in a merge action or switch action. The action ends with an error and the folder containing the file that exists now only in the working copy is marked with a conflict icon ().

Such a conflict can be resolved in one of the following ways which are available when the user double clicks on the conflict in the *Synchronize* view or when he runs the action *Edit conflict*:

Figure 3.27. Resolve a tree conflict

- keep the the local modified file; if there is a renamed version of the file committed by other user that will be added to the working copy too
- delete the local modified file, which means keep the incoming change that comes from the repository.

Update the working copy

While you are working on a project, other members of your team may be committing changes to the project repository. To get these changes, you have to update your working copy. Updating may be done on single files, a set of selected files, or recursively on entire directory hierarchies. The update operation can be performed either from Working Copy view or Synchronize view. The Update action in the Working Copy view is different from the Update action in the Synchronize view. The Update action from the Working Copy view updates the selected resources to the HEAD revision on the repository. The Synchronize view action updates the selected resources to the revision against which the *Synchronize* operation was performed.

There are three different kinds of incoming changes:

- Non-conflicting - A non-conflicting change occurs when a file has been changed remotely but has not been modified locally.
- Conflicting, but auto-mergeable - An auto-mergeable conflicting change occurs when a text file has been changed both remotely and locally (i.e. has non-committed local changes) but the changes are on different lines.
- Conflicting - A conflicting change occurs when one or more of the same lines of a text file have been changed both remotely and locally. Binary files are never auto-mergeable and are conflicting by default.

If the resource contains only incoming changes or the outgoing changes do not intersect with incoming ones then the update will end normally, the Subversion system merging incoming changes into the local file. In the case of conflicting situation the update will have as result a file with conflict status.

The Syncro SVN Client allows you to update your working copy files to a specific revision, not only the most recent one. This can be done by using *Update to revision* action from the History view contextual menu.

If you select multiple files and folders and then you perform an *Update*, all of those files/folders are updated one by one. The Subversion client makes sure that all files/folders belonging to the same repository are updated to the exact same revision, even if between those updates another commit occurred.

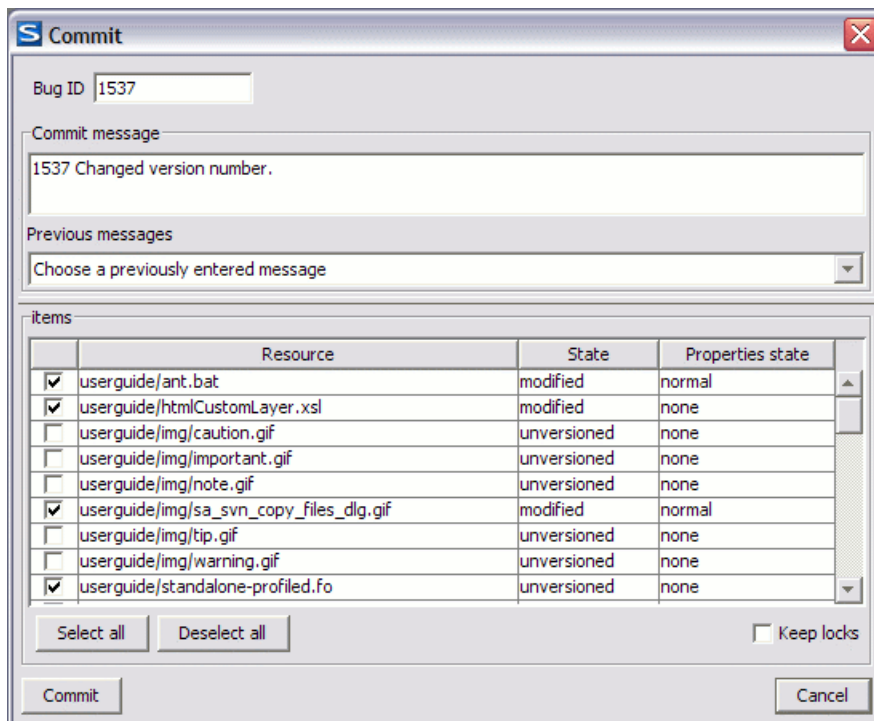
When the update fails with a message saying that there is already a local file with the same name Subversion tried to checkout a newly versioned file, and found that an unversioned file with the same name already existed in your working folder. Subversion will never overwrite an unversioned file unless you specifically do this with *Override and update*. If you get this error message, the solution is simply to rename the local unversioned file. After completing the update, you can check whether the renamed file is still needed.

Send your changes to the repository

Sending the changes you made to your working copy is known as committing the changes. If your working copy is up to date and there are no conflicts, you are ready to commit your changes.

The *Commit* action sends the changes in your local working copy to the repository. After selecting the action from the contextual menu you will see a dialog displaying the resources that can be committed.

Figure 3.28. Commit dialog



Enter a comment to associate with the commit or choose a previously entered comment from the list (the last 10 commit messages will be remembered even after restarting the SVN client application). The dialog will list modified, added, deleted and unversioned resources. All modified, added and deleted resources will be selected by default. If you don't want a changed file to be committed, just uncheck that file. The unversioned items are not selected by default unless you have selected them specifically before issuing the commit command.

To select all resources, click *Select All*. To deselect all resources, click *Deselect All*. Checking the *Keep locks* option will preserve any locks you have on repository resources. Your working copy must be up-to-date with respect to the resources you are committing. This is ensured by using the *Update* action prior to committing, resolving conflicts and

re-testing as needed. If your working copy resources you are trying to commit are *out of date* you will get an appropriate error message.

The table presented in the dialog is sortable. For example if you want to see all the resources that are in the *modified* state click on the *State* column header to sort the table by that column.

The modifications that will be committed for each file can be reviewed in the compare editor window by double clicking on the file in the Commit dialog or by right clicking and selecting the action *Show Modifications*.

If you have modified files which have been included from a different repository using *svn:externals*, those changes cannot be included in the same commit operation.

Integration with Bug Tracking Tools

Users of bug tracking systems can associate the changes they make in the repository resources with a specific ID in their bug tracking system. When the user enters a commit message, the bug ID is added to this message. The format and the location of the ID in the commit message are configured with SVN properties.

To make the integration possible Syncro SVN Client needs some data about the bug tracking tool used in the project. You can configure this using the following SVN properties which must be set on the folder containing resources associated with the bug tracking system. Usually they are set recursively on the root folder of the working copy.

bugtraq:message A string property. If it is set the Commit dialog will display a text field for entering the bug ID. It must contain the string *%BUGID%*, which is replaced with the bug number on commit.

bugtraq:label A string property that sets the label for the text field configured with the *bugtraq:message* property.

bugtraq:url A string property that is the URL pointing to the bug tracking tool. The URL string should contain the substring *"%BUGID%"* which Syncro SVN Client replaces with the issue number. That way the resulting URL will point directly to the correct issue.

bugtraq:warnifnoissue A boolean property with the values "true"/"yes" or "false"/"no". If set to "true", then Syncro SVN Client will warn you if the bug ID text field is left empty. The warning will not block the commit, only give you a chance to enter an issue number.

bugtraq:number A boolean property with the value "true" or "false". If this property is set to "false", then any character can be entered in the bug ID text field. Any other value or if the property is missing then only numbers are allowed as the bug ID.

bugtraq:append A boolean property. If set to "false", then the bug ID is inserted at the beginning of the commit message. If "yes" or not set, then it's appended to the commit message.

bugtraq:logregex This property contains one or two regular expressions, separated by a newline. If only one expression is set, then the bug ID's must be matched in the groups of the regexp string. Example:

```
[Ii]ssue #?(\d+)
```

If two expressions are set, then the first expression is used to find a string which relates to a bug ID but may contain more than just the bug ID (e.g. "Issue #123" or "resolves issue 123"). The second expression is then used to extract the bug ID from the string extracted with the first expression. An example: if you want to catch every pattern "issue #XXX" and "issue #890, #789" inside a log message you could use the following regexp strings:

```
[Ii]ssue #?(\d+)(,? ?#?(\d+))+  
(\d+)
```

The data configured with these SVN properties is stored on the repository when a revision is committed. A bug tracking system or a statistics tools can retrieve from the SVN server the revisions that affected a bug and present the commits related to that bug to the user of the bug tracking system.

If the *bugtraq:url* property was filled in with the URL of the bug tracking system and this URL includes the *%BUGID%* substring as specified above in the description of the *bugtraq:url* property then the History view presents the bug ID as a hyperlink in the commit message. A click on such a hyperlink in the commit message of a revision opens a Web browser at the page corresponding to the bug affected by that commit.

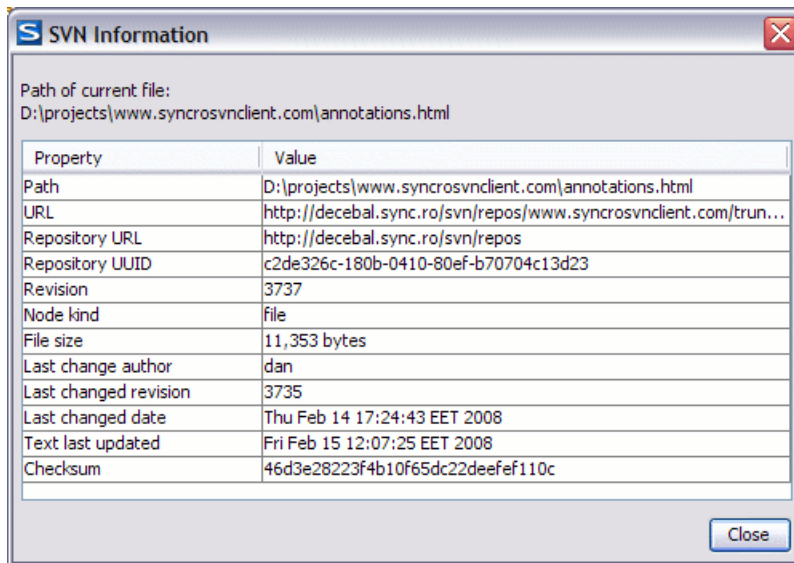
Obtain information for a resource

Request status information for a resource

While you are working you often need to know which files you have changed, added, removed or renamed, or even which files got changed and committed by others. That's where the *Synchronize* action from Working Copy view comes in handy. The *Working Copy View* will show you every file that has changed in any way in your working copy, as well as any unversioned files you may have. If you use Synchronize view then you can also look for changes in the repository. That way you can check before an update if there's a possible conflict.

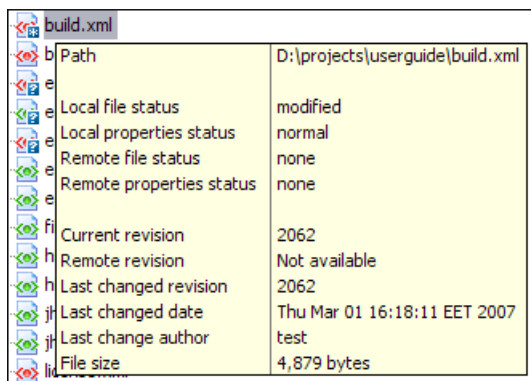
If you want more detailed information about a given resource you can use *Information* action from the *Working copy view* contextual menu or the *Synchronize view* contextual menu. A dialog called *SVN Information* will pop up showing remote and local information regarding the resource, such as:

- local path and repository location
- revision number
- last change author, revision and date
- commit comment
- information about locks
- local file status
- local properties status
- remote file status
- remote properties status
- file size, etc.

Figure 3.29. Information about a SVN resource

The value of a property of the resource displayed in the dialog can be copied by right clicking on the property and selecting the *Copy* action.

A less detailed list of information is also presented when you hover with the mouse pointer over a resource and the tooltip window is displayed.

Figure 3.30. Tooltip for a resource

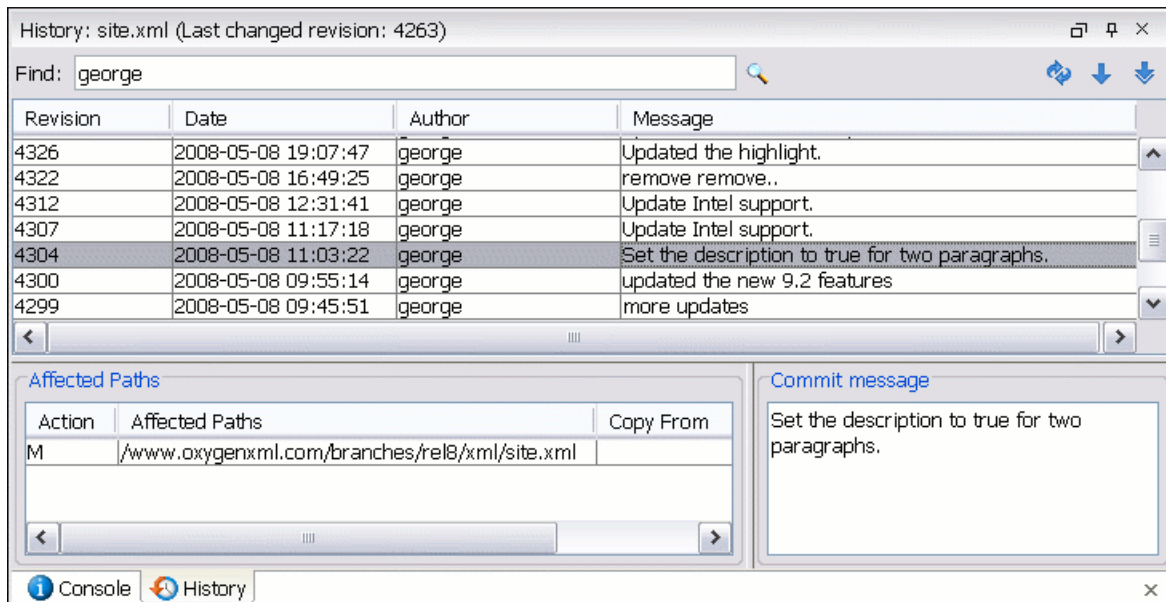
Request history for a resource

In Subversion, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision, what has been changed regarding that resource and who did the changes, drop the modifications made in a certain revision, check out / update the resource to a selected revision, compare two revisions of the same file and other actions, you have to use the *Show history action*. This is available from any of the three views: Repository view menu, Working copy view menu or Synchronize view menu. From the *Repository view* you can display the log history regarding any remote resource residing in repository. From the *Working copy view* you can display the history of local versioned resources. From the *Synchronize view* you can show the history of any incoming or outgoing resources.

The view itself consists of three distinct areas:

- The revision table showing revision numbers, date/time of revision, name of the author, as well as the first line of the commit message. You can click on any revision to show its full details.
- The list of resources affected by this revision (modified, added, deleted or changed properties).
- The commit message for the selected revision.

Figure 3.31. History View



The *Resource history view* does not always show all the changes ever made to a resource because for a large repository there may be thousands of changes and retrieving the entire list can take a long time. Normally you are interested in the more recent ones and that is why the number of revisions is limited by default from the options to 50. This can be changed by accessing the Preferences->SVN page.

Note

When using Subversion servers older than version 1.2, a history request may take a very long time because the server will reply with the entire history even if you limited the number of entries to a smaller number.

Using the resource history view

The *History view* provides a set of actions you can use to get even more information about the project history and make changes to your working copy related with older revisions.

History actions available in the popup menu displayed by a right click in the view when a single resource is selected:

- *Open* - opens the selected revision of the file into the Editor. This is enabled only for files.
- *Save revision as* - saves the selected revision to a file so you have an older version of that file. This option is only available when you access the history of a file, and it saves a version of that one file only.

- *Compare with working copy* - compares the selected revision with your working copy file. It is enabled only when you select a file.
- *Update to revision* - updates your working copy resource to the selected revision. Useful if you want to have your working copy reflect a time in the past. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy will be inconsistent and you will be unable to commit your changes.
- *Check out from revision* - gets the content of the selected revision for the resource into local file system.
- *Revert changes from this revision* - reverts changes which were made in the selected revision. The changes are reverted in your working copy so this operation does not affect the repository file! The action will undo the changes made only in selected revision. It does not replace your working copy file with the entire file at the earlier revision. This is useful for undoing an earlier change when other unrelated changes have been made since the date of the revision. This option is enabled when the resource history was launched for a local working copy resource.
- *Get Contents* - replace the current content of the local version of the selected file with the content of the selected revision.

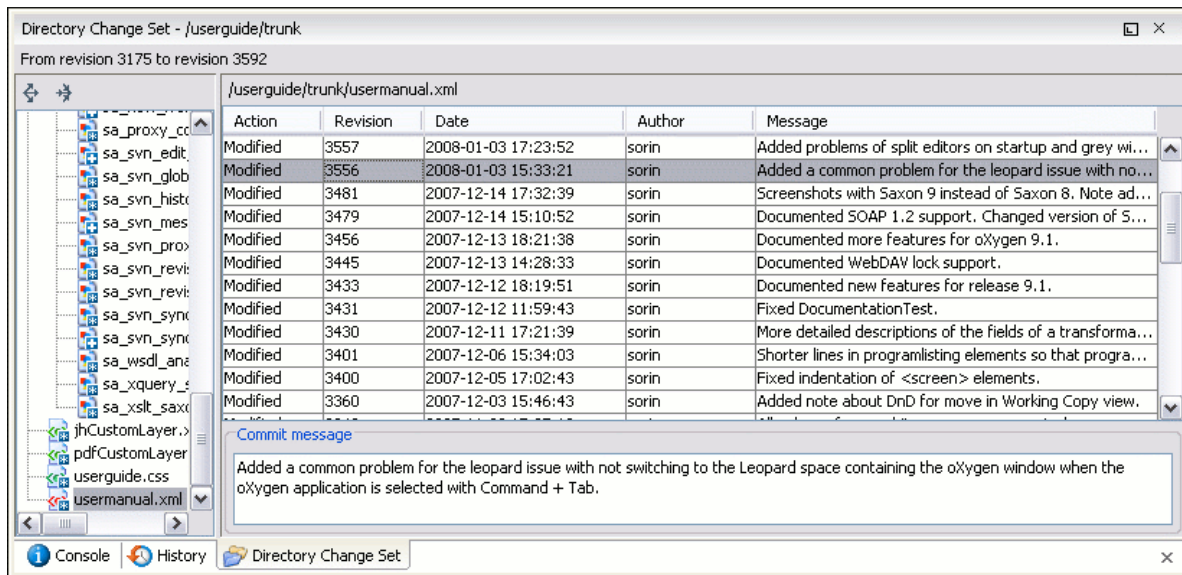
History actions available on the popup menu for double selection:

- *Compare revisions* - When the resource is a file the action compares the two selected revisions using the Compare view. When the resource is a folder the action displays the set of all resources from that folder that were changed between the two revision numbers.
- *Revert changes from these revisions* - Similar to the svn-merge command, it merges two selected revisions into the working copy resource. This action is only enabled when the resource history was requested for a working copy item.

Directory Change Set View

The result of comparing two reference revisions from the history of a folder resource is a set with all the resources changed between the two revision numbers and contained in the folder or in a subfolder of the folder. These resources are presented in a tree format and for each changed resource of the set all the revisions committed between the two reference revision numbers are presented.

Figure 3.32. Directory Change Set View



The set of changed resources displayed in the tree is obtained by running the action *Compare revisions* available on the context menu of the *History* view when two revisions of a folder resource are selected in the *History* view.

The left side panel of the view contains the tree hierarchy with the names of all the changed resources between the two reference revision numbers. The right side panel presents the list with all the revisions of the resource selected in the tree that were committed between the two reference revision numbers. Selecting one revision in the list displays the commit message of that revision in the bottom area of the right side panel.

A double click on a file listed in the left side tree performs a diff operation between the two revisions of the file corresponding to the two reference revisions of the folder for which the change set was computed. A double click on one of the revisions displayed in the right side list of the view performs a diff operation between that revision and the previous one of the same file.

The context menu of the right side list contains the following actions:

- Show Modifications Performs a diff operation between the selected revision in the list and the previous one.
- Open Open the selected revision in the associated editor type.
- Open with Displays a dialog with the available editor types and allow the user to select the editor type for opening the selected revision.
- Save revision to ... Save the selected revision in a file on disk.
- Show Annotation Request the annotations of the file and display them in the *Annotations* view.

Management of SVN properties

In the Properties view you can read and set the Subversion properties of a file or folder. There is a set of predefined properties with special meaning to Subversion. For more information about properties in Subversion see the SVN Subversion specification. Subversion properties are revision dependent. After you change, add or delete a property for a resource, you have to commit your changes to the repository.

Add / Edit / Remove SVN properties

If you want to change the properties of a given resource you need to select that resource from the Working copy view or the Synchronize view and access the *Show properties* action from the contextual menu. The properties view will show the local properties for the resource in the working copy. Once the *Properties View* is visible, it will always present the properties of the currently selected resource.

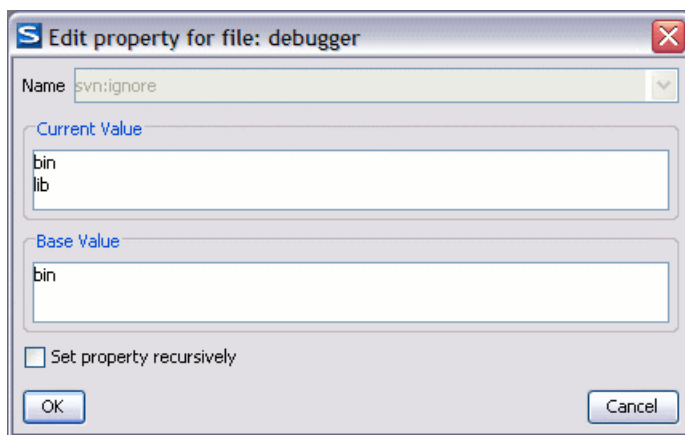
In the Properties view toolbar there are available actions which allow you to add, change and delete the properties.

If you choose the *Add a new property* action, a new dialog will pop-up. The sections in the dialog are:

- Name - it is a combo box which allows you to enter the name of the property. The drop down list of the combo box presents the predefined Subversion properties such as `svn:ignore`, `svn:externals`, `svn:needs-lock`, etc.
- Current value - it is a text area which allows you to enter the value of the new property.

If the selected item is a directory, you can also set the property recursively on its children by checking the *Set property recursively* checkbox.

Figure 3.33. Edit property dialog



If you want to change the value for a previously set property you can use *Edit property* action which will display a dialog where you can set:

- Name - the property name. It cannot be changed; only its value can.
- Current value - presents the current value and allows you to change it.
- Base value - the value of the property, if any, from the resource in the pristine copy. It cannot be modified.

If you want to completely remove a property previously set you can choose *Remove property* action. It will display a confirmation dialog in which you can choose also if the property will be removed recursively.

In the Properties view there is a *Refresh* action which can be used when the properties have been changed from outside the view. This can happen, for example, when the view was already presenting the properties of a resource and they have been changed after an *Update* operation.

Creation and management of Branches/Tags

One of the fundamental features of version control systems is the ability to create a new line of development from the main one. This new line of development will always share a common history with the main line if you look far enough back in time. This line is known as a branch. Branches are mostly used to try out features or fixes. When the feature or fix is finished, the branch can be merged back into the main branch (trunk).

Another feature of version control systems is the ability to take a snapshot of a particular revision, so you can at any time recreate a certain build or environment. This is known as tagging. Tagging is especially useful when making release versions.

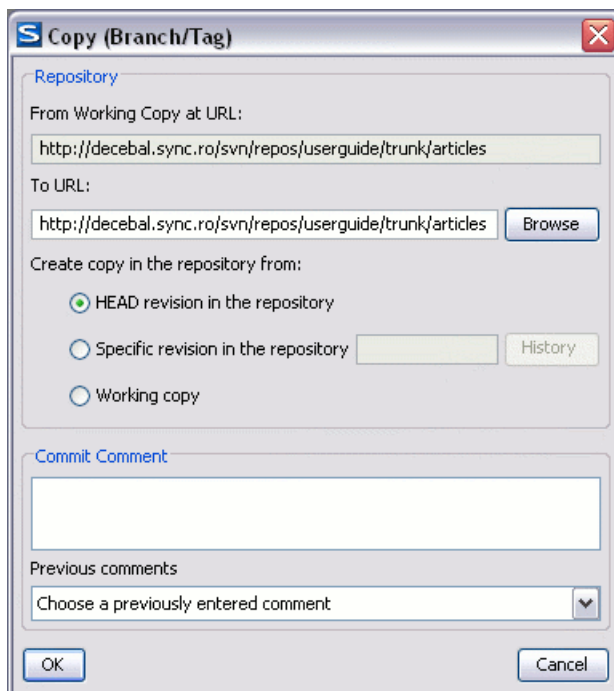
In Subversion there is no difference between a tag and a branch. On the repository both are ordinary directories that are created by copying. The trick is that they are cheap copies instead of physical copies. Cheap copies are similar to hard links in Unix, which means that they merely link to a specific tree and revision without making a physical copy. As a result branches and tags occupy little space on the repository and are created very quickly.

As long as nobody ever commits to the directory in question, it remains a tag. If people start committing to it, it becomes a branch.

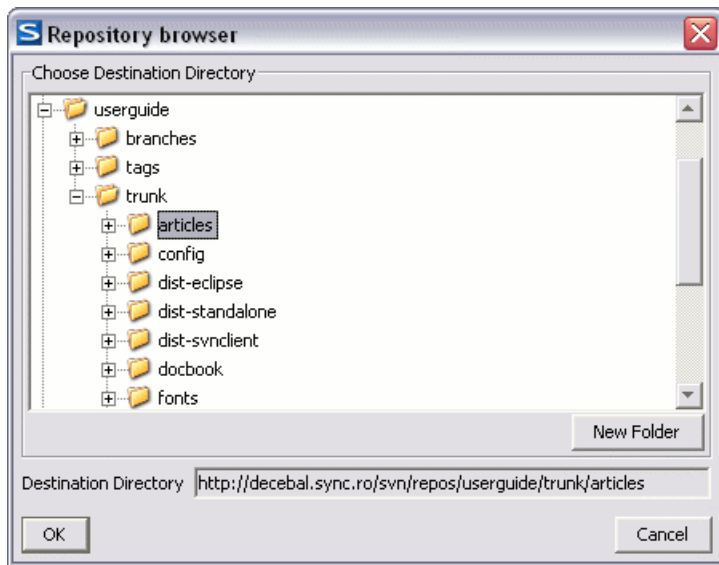
Create a Branch/Tag

In the Working Copy view, select the resource which you want to copy to a branch or tag, then select the command *Branch/Tag...*

Figure 3.34. The Branch/Tag dialog



The default target URL for the new branch/tag will be the repository URL of the selected resource from your working copy. You will need to change that URL to the new path for your branch/tag. To do this, click on the Browse button and choose a repository target directory for your resource.

Figure 3.35. Repository Browser dialog

You can also specify the source of the copy. There are three options:

- HEAD revision in the repository - The new branch/tag will be copied in the repository from the HEAD revision. The branch will be created very quickly as the repository will make a cheap copy.
- Specific revision in the repository - The new branch will be copied in the repository but you can specify exactly the desired revision. This is useful for example if you forgot to make a branch/tag when you released your application. If you click on the History button on the right you can select the revision number from the History dialog. This type of branch will also be created very quickly.
- Working copy - The new branch will be a copy of your local working copy. If you have updated some files to an older revision in your working copy, or if you have made local changes, that is exactly what goes into the copy. This involves transferring some data from your working copy back to the repository, more exactly the locally modified files.

When you are ready to create the new branch/tag, write a commit comment in the corresponding field and press the OK button.

Merging

At some stage during the development you will want to merge the changes made on one branch back into the trunk, or vice versa.

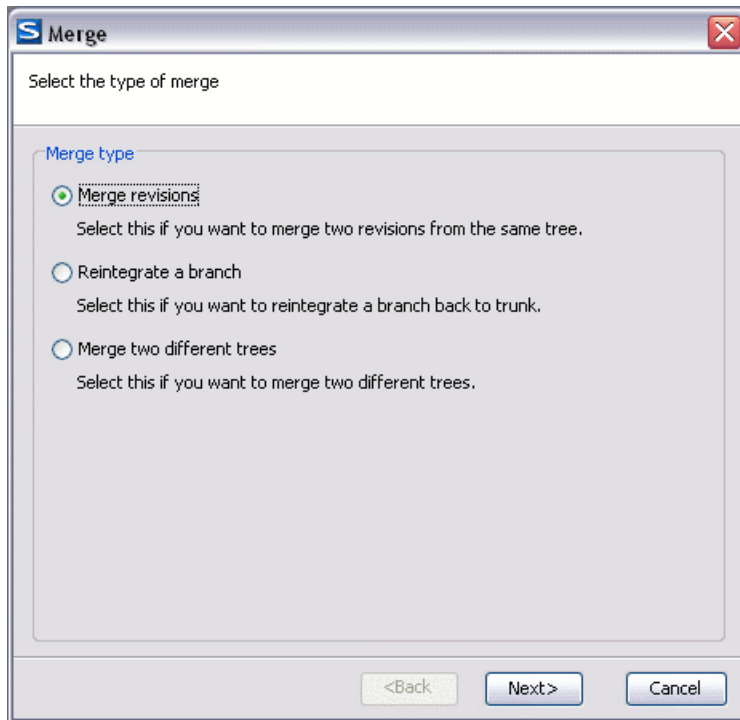
Merge is closely related to Diff. The merge is accomplished by comparing two points (branches or revisions) in the repository and applying the obtained differences to your working copy.

It is a good idea to perform a merge into an unmodified working copy. If you have made changes to your working copy, commit them first. If the merge does not go as you expect, you may want to revert the changes and Revert cannot recover your uncommitted modifications.

The *Merge* command can be found in the *Modify* submenu of the context menu of the Working Copy view. The directory selected when you issued the command will be the result directory of the merge operation.

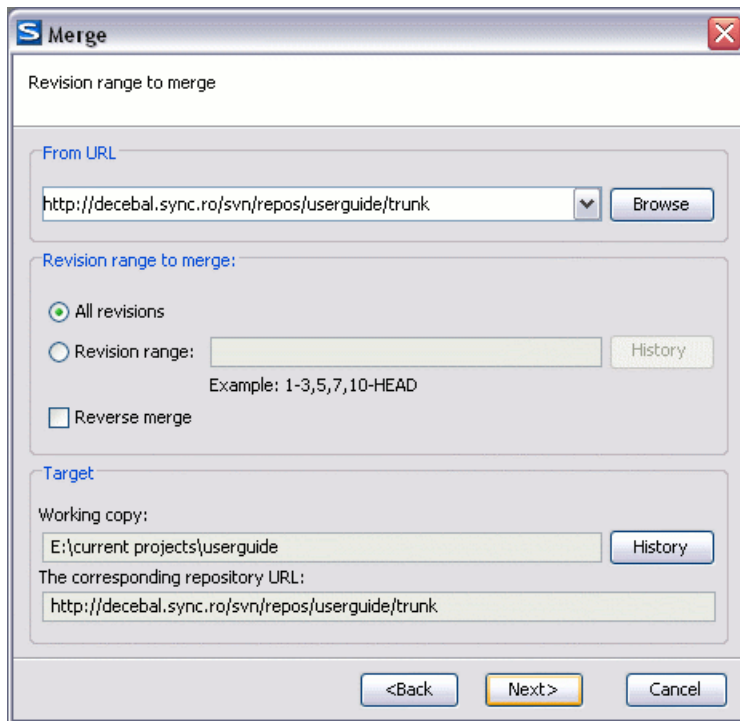
There are three common types of merging which are handled in different ways, as described below. The first page of the merge operation wizard allows you to select the merging method.

Figure 3.36. The Merge type dialog



Merge revisions

This is the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch or trunk. An example of such operation can be the following: Calculate the changes necessary to get (from) revision 17 of branch B1 (to) revision 25 of branch B1, and apply those changes to my working copy, of the trunk or another branch.

Figure 3.37. Merge revisions range dialog

Enter the folder URL of the branch or tag containing the changes you want to port into your working copy in the **From:** field. You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

In the **Revision range to merge** section you can choose to merge all revisions or enter the list of revisions you want to merge in the **Revision range** field. This can be a single revision, a list of comma separated specific revisions, or a range of revisions separated by a dash, or any combination of these.

You can click on **History** button to select in the easiest way the list of revisions to be merged. One or several revisions can be selected then by clicking on **OK** the list of revision numbers to merge will be filled in the **Revision range** text field.

If you want to merge changes back out of your working copy, to revert a change which has already been committed, select the revisions to revert and check the **Reverse merge** box.

If you have already merged some changes from this branch and you remember the last merged revision, you can use **History** for the Working Copy to trace that revision. For example, if you have merged revisions 27 to 33 last time, then the start point for this merge operation should be revision 33.

Subversion has merge tracking features and automatically records the last merged revision so you do not need to remember when you performed the last merge.

Be careful about using the **HEAD** revision. It may not refer to the revision you think it does if someone else made a commit after your last update.

The target panel of the dialog reminds you the location of the target resource from the working copy where the merge result will be saved and its corresponding repository URL.

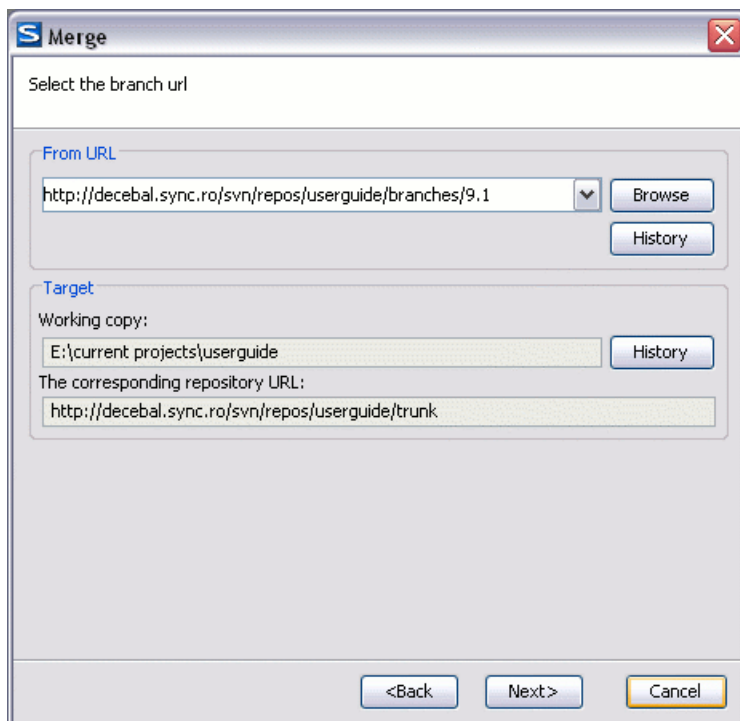
Click **Next** and go to Merge Options.

Reintegrate a branch

This method covers the case when you have made a feature branch. All trunk changes have been ported to the feature branch, and now you want to merge it back into the trunk. Because you have kept the feature branch synchronized with the trunk, the latest versions of branch and trunk will be absolutely identical except for your branch changes. These changes can be reintegrated into the trunk by this method

It uses the merge-tracking features of Subversion to calculate the correct revision ranges to use, and perform additional checks which ensure that the branch has been fully updated with trunk changes. This ensures that you don't accidentally undo work that others have committed to trunk since you last synchronized changes. After the merge, all branch development has been completely merged back into the main development line. The branch is now redundant and can be deleted.

Figure 3.38. Reintegrate a branch dialog



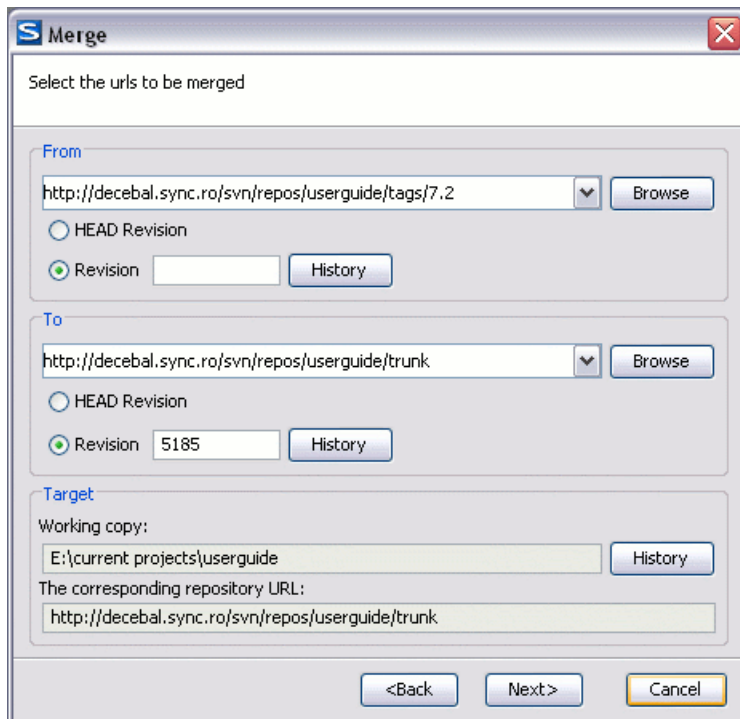
Enter the full folder URL of the branch that you want to merge back. There are some conditions which apply to a reintegrate merge. Firstly, the server must support merge tracking. The working copy must be of depth infinite (no sparse checkouts), and it must not have any local modifications, switched items or items that have been updated to revisions other than HEAD. All changes to trunk made during branch development must have been merged across to the branch (or marked as having been merged). The range of revisions to merge will be calculated automatically.

The target panel of the dialog reminds you the location of the target resource from the working copy where the merge result will be saved and its corresponding repository URL.

Merge two different trees

This is a general case of the reintegrate method. You can consider the following example: calculate the changes necessary to get (from) the HEAD revision of the trunk (to) the HEAD revision of the branch, and apply those changes to my working copy (of the trunk). The result is that trunk will be identical with the branch.

If the server does not support merge-tracking then this is the only way to merge a branch back to trunk.

Figure 3.39. Merge trees dialog

By default the start URL will be the URL of the selected file in the working copy. You can browse the repository and select a start URL and then choose a revision.

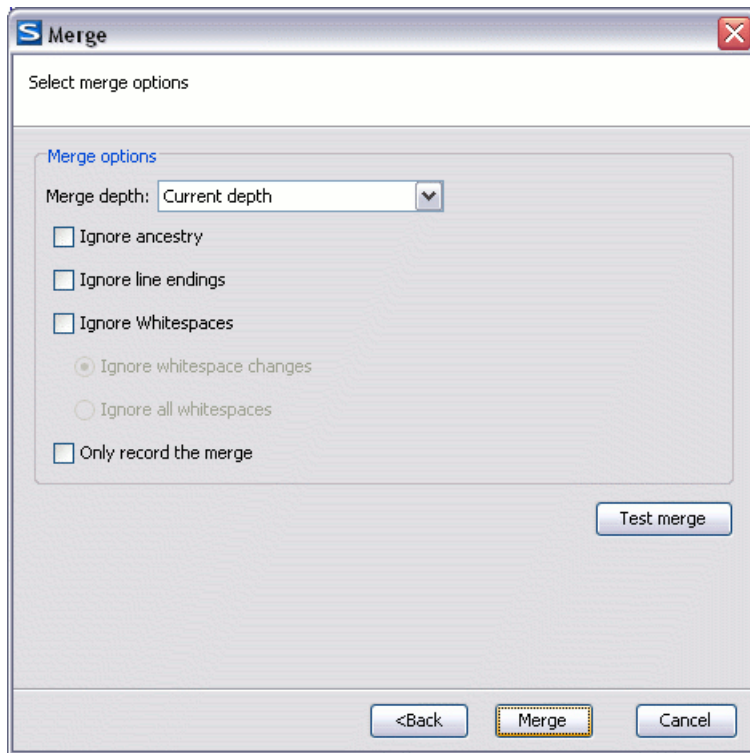
If you are using this method to merge a feature branch back to trunk, you need to start the merge wizard from within a working copy of trunk. In the "From" field enter the full folder URL of the trunk. This may sound wrong, but remember that the trunk is the start point to which you want to add the branch changes. In the "To" field enter the full folder URL of the feature branch.

In both the From Revision field and the To Revision field, enter the last revision number at which the two trees were synchronized. If you are sure no-one else is making commits you can use the HEAD revision in both cases. If there is a chance that someone else may have made a commit since that synchronization, use the specific revision number to avoid losing more recent commits.

The target panel of the dialog reminds you the location of the target resource from the working copy where the merge result will be saved and its corresponding repository URL.

Merge Options

This page lets you specify advanced options, before starting the merge process.

Figure 3.40. Merge options dialog

You can specify how far down into your working copy the merge should go by setting the merge depth. The depth term is described in the Sparse checkouts section. The default depth is Working copy, which uses the existing depth setting.

The **Ignore ancestry** checkbox allows a merge to be applied between a branch and the trunk or between two branches even if they do not share a common ancestry. Normally the branch and the trunk or the two branches that are merged must have a common ancestor revision in the same repository. In case the two merged trees were imported in the repository they are not related in the sense of a common ancestor tree and the merge operation is possible by ignoring the missing common ancestry of the two merged trees.

The **Ignore line endings** and **Ignore whitespaces** checkboxes allow you to specify how the line endings and whitespace changes should be handled. If they are checked the changes due only to the line endings and whitespaces are ignored. The default behavior is to treat all whitespace and line-end differences as real changes to be merged. **Ignore whitespace changes** excludes changes which are caused by a change in the amount or type of whitespace, for example changing the indentation or changing tabs to spaces. Adding whitespace where there was none before, or removing a whitespace completely is still shown as a change. If **Ignore all whitespaces** is checked all whitespace-only changes are excluded.

If you are using merge tracking support and you want to mark a revision as having been merged, without actually doing the merge here, check the **Only record the merge** checkbox. You might want to do this for two possible reasons. You make the changes by hand, then mark the change as merged so that the merge tracking algorithm is aware of it. Or you might want to prevent a particular revision from being merged by marking it as already merged. This will prevent future merging.

By pressing the Test merge button you can choose to do a **Dry run** of the *Merge operation* in order to see what files are affected and how, without modifying the working copy at all. This is very helpful in detecting where conflicts may occur.

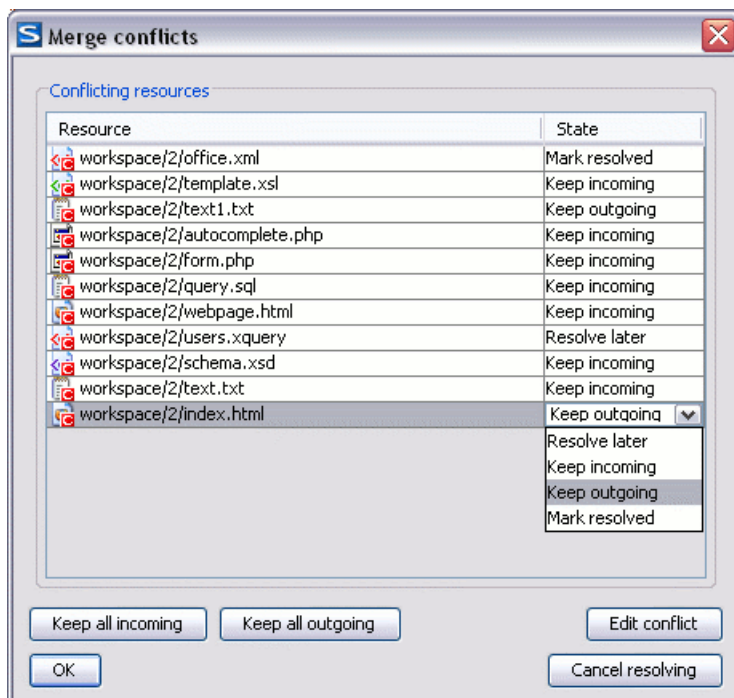
Press the Merge button in order for the operation to take place. You will obtain the result in the selected resource from the working copy.

When the merge is completed it's a good idea to look at the result of the merge and see if it meets your expectations. Because merging is sometimes complicated, when there are major changes, conflicts may appear.

Resolve merge conflicts

After the merge operation is finished it is possible to have some resources in conflict. This means that some incoming modifications for a resource could not be merged with the current modifications from the working copy. If there are such conflicts, a dialog will appear presenting you the resources that are in conflict and from where you can choose a way in which every conflict should be resolved.

Figure 3.41. Merge conflicts dialog



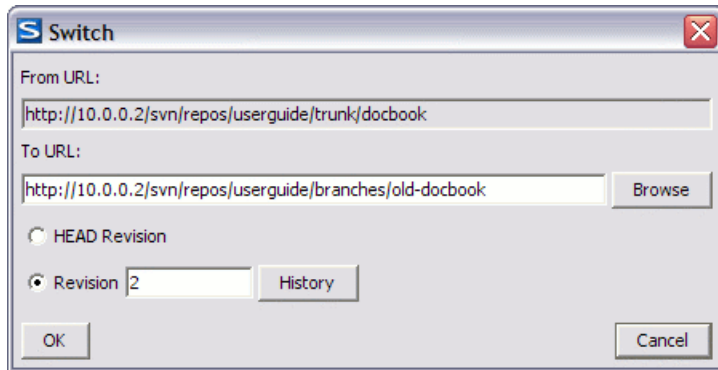
The options to resolve a conflict are:

- Resolve later - used to leave the conflict as it is for manual resolving it later;
- Keep incoming - this option keeps all the incoming modifications, discarding all current ones from your working copy;
- Keep outgoing - this option keeps all current modifications from your working copy, discarding all incoming ones;
- Mark resolved - you should chose this option after you have manually edited the conflict. To do that, use the *Edit conflict* button, which will bring to you a dialog presenting the conflicting resource's content for current working copy version and the one with the incoming modifications. After manually resolving the conflict, the resource will be marked as resolved.

Switch the Repository Location

The Switch action is useful when the repository location of a working copy or only of a versioned item of the working copy must be changed within the same repository. It is available on the contextual menu of the working copy tree when the selected item is a versioned resource except an external folder.

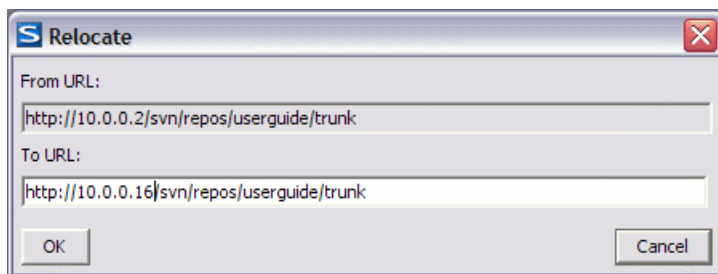
Figure 3.42. The Switch Dialog



Relocate a Working Copy

When the base URL of the repository changed, for example the repository itself was relocated to a different server, you do not have to check out again a working copy from the new repository location. It is easier to change the base URL of the root folder of the working copy to the new URL of the repository. This action is available on the contextual menu of the Working Copy view only if the selected item of the working copy tree is a versioned folder.

Figure 3.43. The Relocate Dialog



If the selected item is not the root folder of the working copy then the effect is the same as for the Switch action applied on the same selected item.

Create Patches

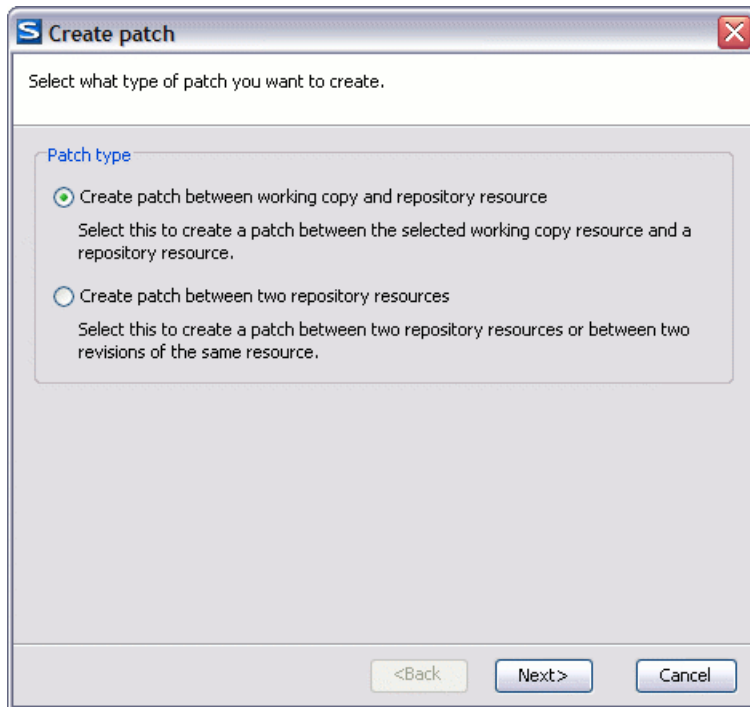
Let's suppose you are working to a set of XML files, that you distribute to other people. From time to time you are tagging the project and distribute the releases. If you continue working for a period correcting problems, you may find yourself in the situations to notify your users that you have corrected a problem. In this case you may prefer to distribute them a patch, a collection of differences that applied over the last distribution would correct the problem. The SVN client creates the patch in the Unified Diff format [http://en.wikipedia.org/wiki/Diff#Unified_format].

Creating patches in Subversion implies the access to two states (revisions) of a project. If you have not committed yet your current working copy and prefer not to do it, it is possible to create a patch between the current working copy and

a revision from the repository. If you want to create a patch between two revisions that are already committed to the repository that is also possible.

In order to create the patch, you will use the action from the submenu *Modify* of the contextual menu of the Working Copy view: Create Patch. This opens the *Create patch* wizard.

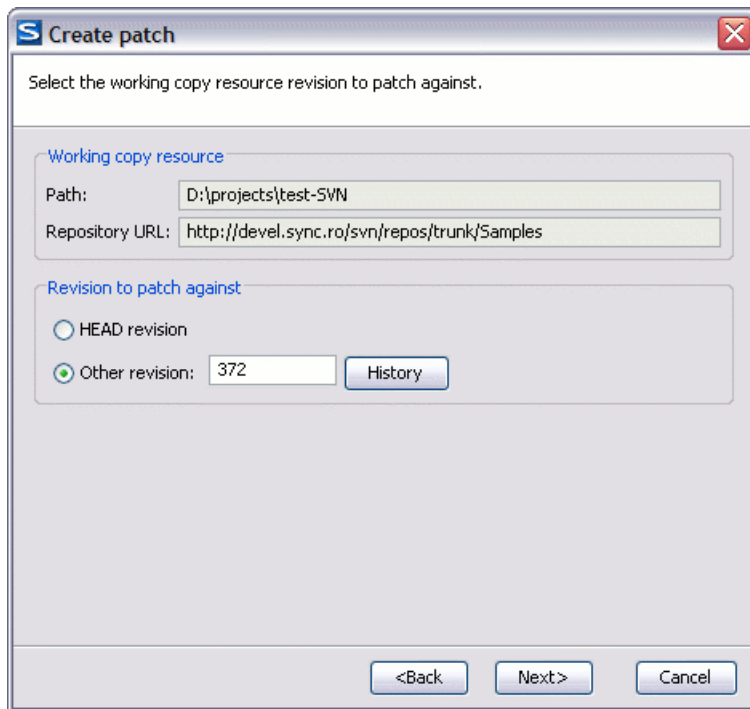
Figure 3.44. The Create patch wizard - step 1



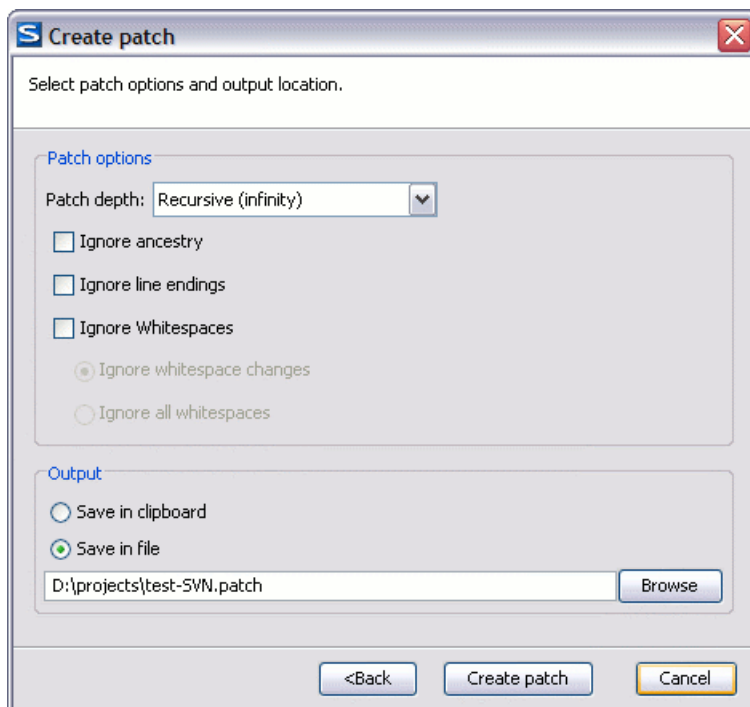
In the first step of the wizard you select the type of the patch: a patch between working copy and repository revision or a patch between two repository revisions. The *Next* button moves the wizard to the second step.

Create a patch from working copy

In case of the first type of patch in this step you specify the revision of the repository for finding the patch between the working copy and the repository. The revision can be HEAD or a revision number selected from the list of all revisions committed to the repository.

Figure 3.45. Patch between working copy and repository - step 2

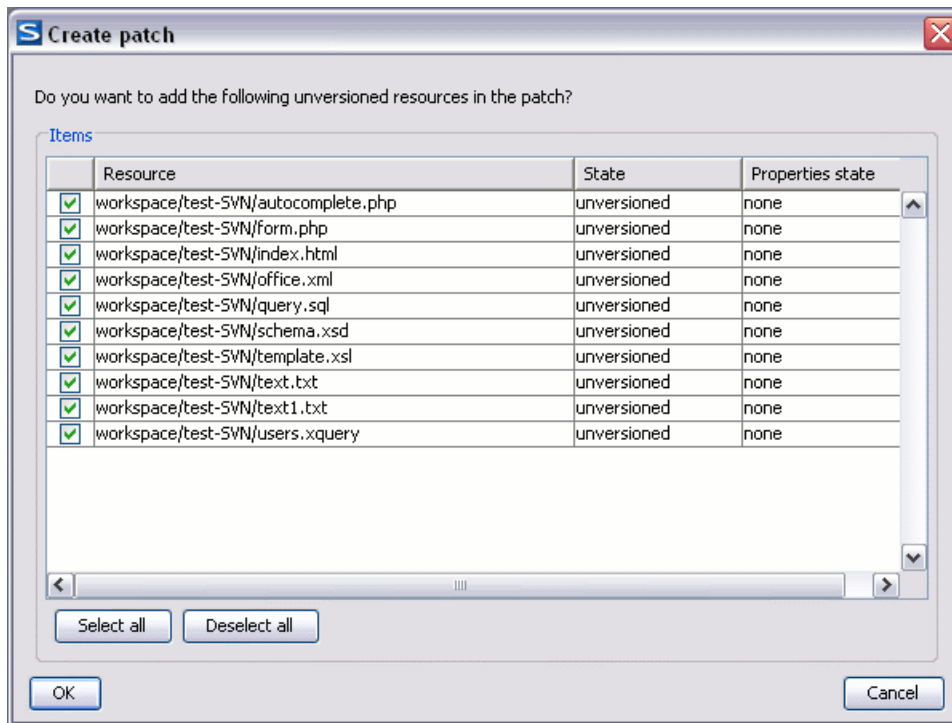
In the next step the following options can be specified:

Figure 3.46. Patch between working copy and repository - step 3

Patch depth	The depth of recursive folders included in the patch. If the patch is created only for a file then the depth is always zero. The depth can have one of the values:
Current depth	The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
Recursive (infinity)	The patch is created on all the files and folders contained in the selected folder.
Immediate children (immediates)	The patch is created only on the child files and folders without going in subfolders.
File children only (files)	The patch is created only on the child files.
This folder only (empty)	The patch is created only on the selected folder (that is no child file or folder is included in the patch).
Ignore ancestry	The SVN ancestry that may exist when the two URLs specified for creating the patch have a common SVN history is ignored when the patch is created.
Ignore line endings	The differences in line endings are ignored when the patch is created.
Ignore whitespaces	The differences in whitespaces are ignored when the patch is created.
Save in clipboard	The patch will be created and saved in clipboard.
Save in file	The patch will be created and saved in the specified file.

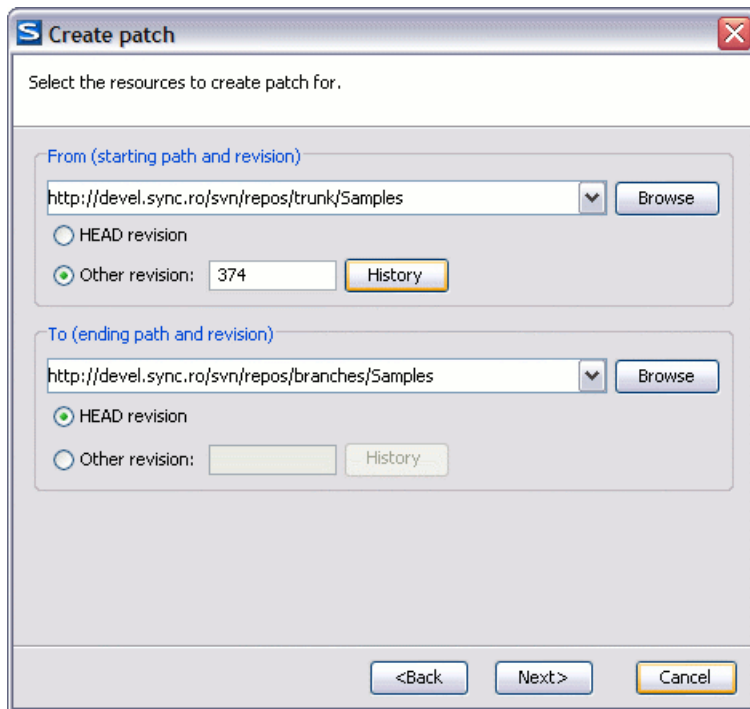
Include unversioned files in the patch

In the next step you can specify the unversioned files that will be included in the generated patch. If the patch is applied on a folder of the working copy and that folder contains unversioned files this step of the wizard offers the option of selecting the ones that will be included in the patch.

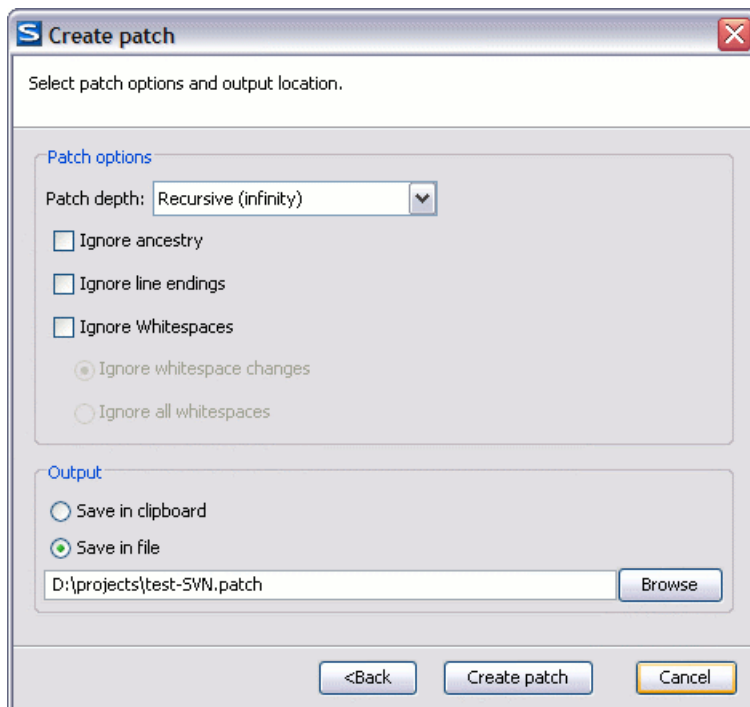
Figure 3.47. Patch between working copy and repository - step 4

Create patch from repository revision

In case of the second type of patch in the second step of the wizard you select the two repository revisions. The two revisions can be on the same repository or on two different repositories. For each revision you can select the HEAD revision or a revision number available on the repository.

Figure 3.48. Patch between two repository revisions - step 2

The next step of the wizard is the same as for the first type of patch. It allows to specify the options patch depth, ignore ancestry, ignore line endings, ignore whitespaces, save in clipboard, save in file. The description of the options is the same as for the first type of patch.

Figure 3.49. Patch between two repository revisions - step 3

Working with repositories

Import / Export resources

Import resources into the repository

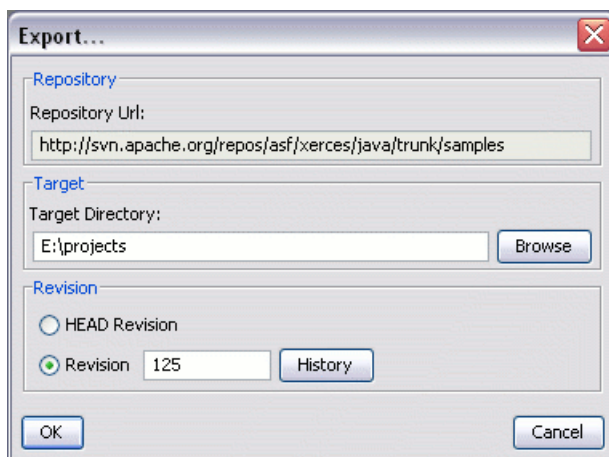
This is the process of taking a project and importing it into a repository so that it can be managed by Subversion. If you have already been using Subversion and you have an existing working copy you want to use, then you will likely want to follow the procedure for Use an existing working copy.

A dialog will ask you to select a directory that will be imported into the selected repository location. The complete directory tree will be imported into the repository including all files. The name of the imported folder will not appear in the repository, but only the contents of the folder will.

Export resources from the repository

This is the process of taking a resource from the repository and saving it locally in a clean form, with no version control information. This is very useful when you need a clean build for an installation kit.

Figure 3.50. The export dialog



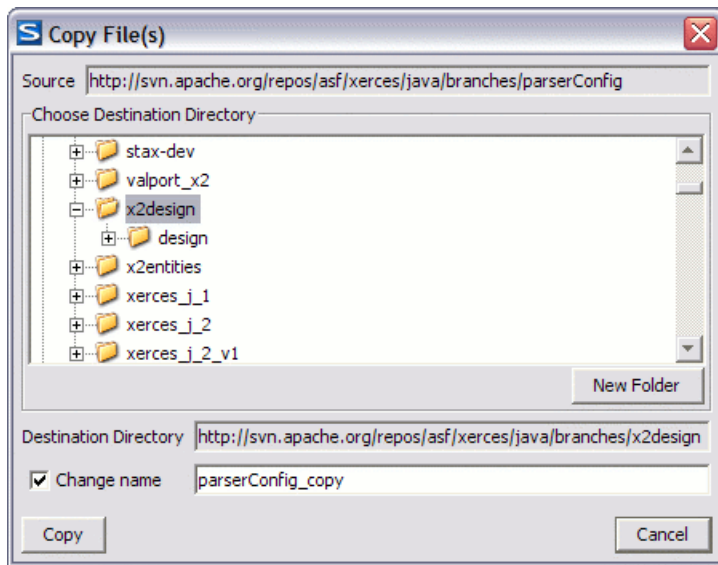
The export dialog is very similar to the check out dialog. You can choose the target directory from the file system by pressing the *Browse* button. If you need to export a specific revision, you can select the *Revision* radio button and then click on the *History* button and choose a revision from the new dialog. Or you could simply type the revision number in the corresponding text field.

Please note that the content of the selected directory from the repository and not the directory itself will be exported to the file system.

Copy / Move / Delete resources from the repository

Once you have a location defined in the Repository view you can execute commands like copy, move and delete directly on the repository. The commands correspond to the following actions in the contextual menu.

The *Copy* action allows you to copy individual or multiple resources. After invoking the action the *Copy file* dialog will pop up.

Figure 3.51. Copy files dialog

The dialog displays the path of the resource that is copied and the tree structure of the repository allowing you to choose the destination directory. The path of this target directory will be presented in the text field *Destination Directory*. If you choose to copy a single resource then an additional checkbox and a text field allow you to choose the new name of the copied resource.

The *Move* action will display a similar dialog allowing you to move the selected resources to a different folder. If you choose to move a single resource you can also change its name. This will allow you to *rename* a resource by moving it into the same parent directory but choosing a different name. The move operation is basically a *copy operation* followed by a *delete operation*. If you select a directory, any other selected descendants will be ignored when you have issued the move command.

Another useful action is *Delete*. This action allows you to delete resources directly from the repository. After choosing the action from the Repository view contextual menu a confirmation dialog will be displayed.

All three actions are commit operations and you will be prompted with the *Commit message* dialog.




Sparse checkouts

Sometimes you need to check out only certain parts of a directory tree. For this you can checkout the top folder and then update recursively only the needed directories. Each directory now understands the notion of depth which has four possible values:

- *Recursive (infinity)* - Updates all descendant folders and files recursively.
- *Immediate children* - Updates the directory including direct child folders and files but does not populate the child folders.
- *File children only (files)* - Updates the directory including only child files without the child folders.
- *This folder only (empty)* - Updates only the selected directory without updating any children.

Current depth - for some operations you can use as depth the current depth of the resource from working copy. This is the depth of directories from the working copy, it can have one of the values defined above. This is the depth value defined in a previous checkout or update operation.

The sparse checked out directories are marked in the Working Copy view with a marker corresponding to the depth value as follows:

- *Recursive (infinity)* - This is the default value and it has no mark.
- *Immediate children (immediates)* - The directory is marked with a purple bubble in the top left corner .
- *File children only (files)* - The directory is marked with a blue bubble in the top left corner .
- *This folder only (empty)* - The directory is marked with a gray bubble in the top left corner .

The depth information is also presented in the Information view and the tool tip displayed when hovering over the directory in the Working copy view.

This feature requires the svn client to be 1.5 or above and will work most efficiently if the server is 1.5 or above. The client will work also with a 1.4 server or lower but will be less efficient.







Repository View

General description

The repository view allows you to define and manage Subversion repository locations. Repository files and folders are presented in a tree view with the repository locations at the first level, where each location represents a connection to a specific Subversion Repository. When hovering with the mouse over a repository resource a tooltip window will display more detailed information regarding: URL, last change revision, last change author, last change date.

Toolbar

The toolbar for the repository view contains the following buttons:

-  New Repository Location - allows you to enter a new repository location by means of the *Add SVN Repository* dialog.
-  Check Out - checks out a working copy from the selected directory in the repository. Displays the new working copy in the *Working Copy view*.
-  Edit Repository Location - context dependent, allows you to edit the selected repository location by means of the *Edit SVN Repository* dialog. It is active only when a repository location root is selected.
-  Remove Repository Location - allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.
-  Move Up - move the selected repository up with one position in the list of repositories in the Repository view.
-  Move Down - move the selected repository down with one position in the list of repositories in the Repository view.

Contextual menu actions

The repository view has one contextual menu for the repository locations (roots) and another contextual menu for the repository resources. Besides the actions described above, the locations context menu from the repository view contains the following actions:



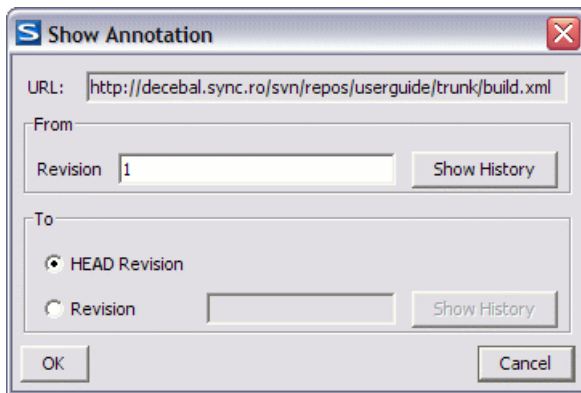



- Import Folder Content ... - imports the content of a specified folder from the file system into the selected folder from the repository.
- Export ... - exports a directory from the repository to the local file system.
-  Show History ... - brings up the History view and displays the log history for the selected resource from the repository.
-  Show Annotation ... - brings up a dialog for selecting the start revision and the end revision of the interval of revisions for which the SVN annotations will be computed and marked in the selected resource in the editor panel.




Figure 3.52. The Show Annotation dialog






After selecting the start revision and the end revision and pressing OK the Annotations view and the History view are displayed and the annotations are marked on the SVN resource in the editor panel.

- New Folder ... - allows you to create a new folder in the selected repository path.
-  Show SVN Properties - brings up the Properties view and displays the SVN properties for the selected repository resource. This view does not allow adding, editing or removing SVN properties of a repository item. These operations are allowed only for working copy resources.
-  File Information ... - provides additional information for the selected repository. For more details please see the section Information view.
-  Refresh - refreshes the currently selected repository.

The repository resources context menu from the view contains the following actions:

-  Check Out ... - checks out a working copy from the selected directory in the repository.
- Import ... - imports a directory from the file system into the selected directory from the repository.
- Export ... - exports a directory from the repository to the local file system.
-  Show History ... - brings up the History view and displays the log history for the selected resource from the repository.
-  Show Annotation ... - brings up the Annotations view .
- New Folder ... - allows you to create a new folder in the selected repository path.

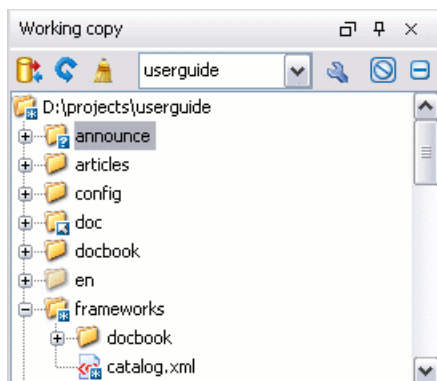
- Open - opens the selected file in the Editor view in read only mode.
- Open with ... - Displays the 'Open with...' dialog for specifying the editor in which the selected file will be opened. In case multiple files are selected only external applications can be used to open the files.
- Copy ... - displays the *Copy Files* dialog which allows you to select the location where the selected resources will be copied.
- Copy URL Location - copies the encoded URL for the selected resource from the repository to the clipboard.
- Rename ... - renames the current folder on the repository.
- Move ... - displays the *Move Files* dialog which allows you to select the location where the selected resources will be moved.
- Delete - deletes the selected resources. It will ask for confirmation.
-  Show SVN Properties - brings up the Properties view and displays the SVN properties for the selected resource from the repository.
-  File Information ... - provides additional information for the selected resource from the repository. For more details please see the section Information view.
-  Refresh - refreshes the currently selected resources from the repository.

Working Copy View

General description

The working copy view allows you to manage with ease the content of the working copy. Resources (files and folders) are presented in a tree view with the root of the tree representing the location of the working copy on the file system. Each resource has an icon representation which describes the type of resource and also depicts the state of that resource with a small overlay icon.

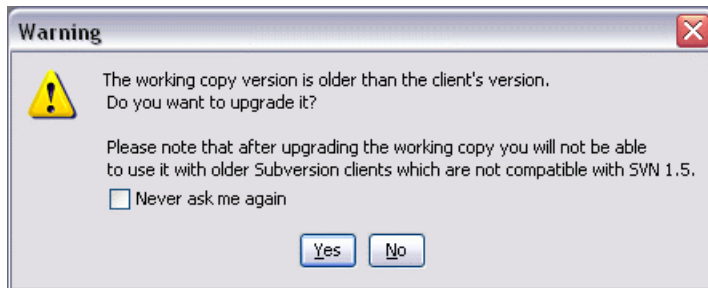
Figure 3.53. The Working Copy View



When a SVN working copy is loaded in the view by selecting it in the combo on the toolbar of the view Syncro SVN Client first checks the format of the working copy. If it is a SVN 1.6 format the admin data of that working copy is loaded and displayed in a tree like form in the view using the icons specific for the status of each resource: normal,

unversioned, modified, etc. If it is the old format, that is the SVN 1.5, SVN 1.4 or SVN 1.3 one, a confirmation dialog is displayed allowing the automatic conversion to the working copy to the new format, that is the SVN 1.6 one.

Figure 3.54. The Working Copy format warning dialog









If you select the *Never ask me again* checkbox and press the *Yes* button then the option *Automatically upgrade working copies to the client's version* is automatically checked.

For each file and folder a tooltip is displayed with details like SVN status, full path, current revision number, last changed date, etc. If the tooltips seem annoying by covering useful information they can be disabled from the option *Show tooltip on Working Copy and Synchronize trees*.

Toolbar

The toolbar from the working copy view contains the following buttons:

-  Synchronize - contacts the repository and determines the changes made by you to the working copy and by others to the repository. The synchronize result will be displayed in the Synchronize view. The action performs a synchronize operation on the root of the working copy.
-  Refresh - refreshes the content of the working copy. The content of the working copy is always scanned(refreshed) when starting the Subversion client or when changing the working copy from the combo box in the toolbar. However, if you make modifications from other applications outside the Subversion client, while the client is started, you will have to manually refresh the working copy. The action performs a refresh operation on the root of the working copy.
-  Cleanup - performs a maintenance cleanup operation on the working copy. Sometimes, when an operation fails, the working copy will enter an inconsistent state in which some resources will remain locked by SVN. Cleanup removes those maintenance locks and allows you to continue your work. When the SVN client determines that an operation failed because the working copy is locked by SVN you will be asked if a *Cleanup* operation should be performed first.
- Combo box - The combo box list contains all the working copies the Subversion client is aware of. When you select another working copy from the combo, the newly selected working copy content will be scanned and displayed in the view.
-  Add/Remove Working Copy - opens the *Working copies list* dialog which displays the working copies the Subversion client is aware of. In this dialog you can add existing or remove no longer needed working copies. If you try to add a directory which is not a valid Subversion working copy, a warning dialog will inform you that the selected directory is not under version control. Please note that removing a working copy from this dialog will NOT remove it from your file system; you will have to do that manually.
-  Show ignored files - shows in the working copy tree the resources that were listed in the `svn:ignore` property of their parents. This option is off by default.

-  Show deleted files - shows in the working copy tree the resources that were marked to be deleted but are not yet committed. This option is off by default.

Contextual menu actions

The contextual menu in the Working Copy view contains the following actions:



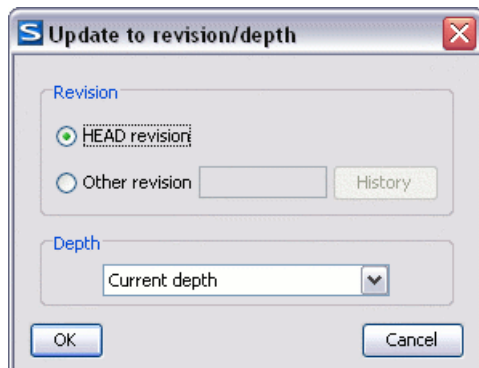
-  Open - This action will open the selected file in an editor where you can make modifications to it. The action is active only when a single item is selected. In case of a file the action opens the file with the internal editor or the external application associated with that file type. In case of a folder the action opens the selected folder with the system application for folders (for example Windows Explorer on Windows, Finder on Mac OS X, etc).
- Open with ... - Displays the 'Open with...' dialog for specifying the editor in which the selected file will be opened. In case multiple files are selected only external applications can be used to open the files
-  Synchronize - it contacts the repository and determines the working copy and the repository changes made to the selected resources. It displays the result of the operation in the *Synchronize view*. This is useful when you have a large working copy and you only want to verify the changes to a specific part you are currently working on.
- Update - This command updates your selected resources from the working copy to the HEAD revision from the repository (latest modifications). It the same as the update action from the *Synchronize view* in that also brings you the HEAD revision from the repository. If a directory is involved it will be updated depending on its depth. The action is active only on resources that are under version control.
- Update to revision/depth - This action allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the depth term in the sparse checkouts section. The action is active only on resources under version control.

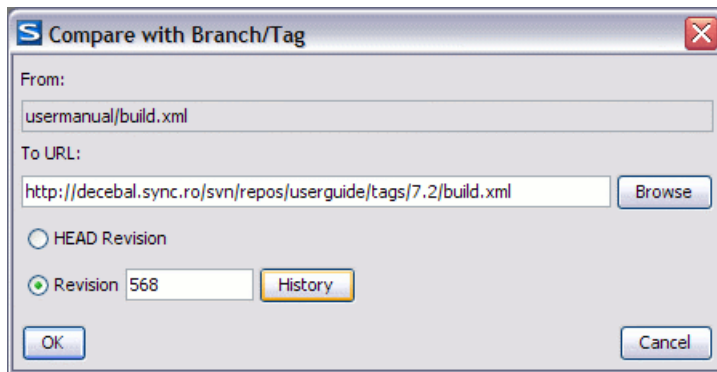
Figure 3.55. Update to revision/depth dialog





- Commit - This action collects the outgoing changes from the selected resources in the working copy and presents them in a dialog. Then you can choose exactly what to commit by selecting or unselecting resources accordingly. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you will also have to enter a commit comment before sending your changes to the repository.
- Compare with:

- Latest from Head - This action will perform a 3-way diff operation between the selected file and the HEAD revision from the repository and will display the result in the Compare view. The common ancestor of the 3-way diff operation is the BASE version of the file from the local working copy.
- Base revision - This will compare the working copy file with the file from the pristine copy (BASE revision).
- Revision - This command will bring to front the History view with the log history for that resource.
- Branch/Tag - This will compare the working copy file with a revision of the file from a branch or tag. The revision is specified by URL (selected with a repository browser dialog) and revision number (selected with a revision browser dialog).

Figure 3.56. Compare with Branch/Tag dialog

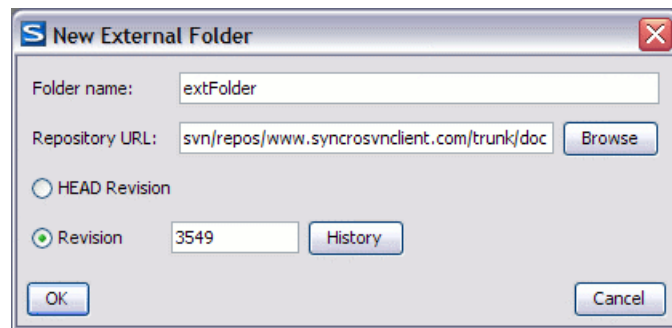


- Each other - This action only works when two files are selected. It will compare the two selected files with each other.
These actions are enabled only if the selected resource is a file.
-  Show History ... - It will display the *History view* where the log history for the selected resource will be presented. For more details about resource history see the sections Using the resource history view and Request history for a resource.
-  Show Annotation - It will display the *Annotations view* where all the users that modified the selected resource will be presented together with the specific lines and revision numbers modified by each user. For more details about resource annotations see the section Annotations View.
- New:
 - New Folder ... - This operation creates a new folder and adds it to version control. If the selected path is not under version control, the newly created directory will not be added to version control.
 - New File ... - This operation creates a new file and adds it to version control. If the selected path is not under version control, the newly created file will not be added to version control.
 - New External Folder ... - This operation sets a folder name in the property *svn:externals* of the selected folder. The repository URL to the folder to which the new external folder will point and the revision number of that repository URL can be selected easily with the *Browse* and *History* buttons of the dialog.






Subversion clients 1.5 and higher support relative external URLs. You can specify the repository URLs to which the external folders point using the following relative formats:

- ../ - Relative to the URL of the directory on which the svn:externals property is set.
- ^/ - Relative to the root of the repository in which the svn:externals property is versioned.
- // - Relative to the scheme of the URL of the directory on which the svn:externals property is set.
- / - Relative to the root URL of the server on which the svn:externals property is versioned.

Figure 3.57. New External Folder dialog



- Edit:
 - Delete - This action allows you to delete resources from the Subversion working copy. If unversioned, added or modified resources will be encountered, a dialog will prompt you to confirm their deletion. This is because their content cannot be recovered. The action is not enabled when the selection contains *missing* resources.
 - Copy ... - This action copies resources from the working copy. Each copy will also have the original resource's history. For more details please read the section Copy / Move / Rename resources.
 - Move ... - This command actually performs as if a copy and then a delete command were issued. You will find the moved resources at the desired destination and also at their original location but marked as deleted.
 - Rename ... - You can only rename a resource at a time. As for the move command, a copy of the original resource will be made with the new name and the original will be marked as deleted.
- Revert - Undoes all local changes for the selected resources. It does not contact the repository, the files will be obtained from Subversion's pristine copy. It is enabled only for modified resources. Read the Revert your changes section for more information.
- Mark resolved - This action is only enabled on *conflicted* resources and its function is to tell the Subversion system that you resolved the conflict. See the section Merge conflicts.
- 🧹 Cleanup - performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.
- Add - This operation adds the selected resources to version control. A directory will be added recursively to version control. It is not mandatory to explicitly add resources to version control but it is recommended. At commit time unversioned resources will have to be manually selected in the commit dialog. It is only active on unversioned resources.













- Add to svn:ignore ... - This action can only be performed on resources not under version control. It allows you to keep inside your working copy, files that should not participate to the version control operations. The action actually modifies the value of the *svn:ignore* property of the resource's parent directory.
-  Show SVN Properties - brings up the Properties view and displays the SVN properties for the selected resource.
- Lock:
 - Scan for locks ... - This action contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see the section Scanning for locks.
 -  Lock ... - It allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force(steal) the lock. The action is active only on files under version control. For more details on the use of this action please read the section Locking a file.
 -  Unlock ... - This action releases(unlocks) the exclusive access to a file from the repository. You can also choose to unlock it by force(break the lock).
- Modify:
 - Branch/Tag ... - This action displays the Branch/Tag dialog where you can select the revision of the resource and the destination URL in the repository. For more details about creating branches and tags see the section Create a branch/tag.
 - Merge ... - The selected resource will be considered the destination for the merge operation. From the displayed Merge dialog you will be able to specify the merge operation you want to perform. See section Merging section for more details.
 - Switch ... - The repository location of the working copy or only of a resource of the working copy is changed within the same repository. It is available when the selected item is a versioned resource except an external folder.
 - Relocate ... - Change the repository path of the selected item. This action is available on the contextual menu only if the selected item is a versioned folder. It is useful when the base URL of the repository changed, for example the repository itself was relocated to a different server. If the selected item is not the root folder of the working copy then the effect is the same as for the Switch action applied on the same selected item.
 - Create patch ... - The selected resource will be used to create a patch. From the displayed Create patch dialog you will be able to specify the two sources of the patch, the first one being either the working copy files or a revision/tag and the second a repository URL or a revision/tag. See section Create patches for more details.
-  File Information ... - provides additional information for the selected resource from the working copy. For more details please see the section Obtain information for a resource.
-  Refresh - This action will rescan the selected resources recursively and refresh their status in the working copy view.

Drag and drop operations

New files and folders can be added to the file tree of the Working Copy view as unversioned resources by drag and drop operations from other applications, for example Windows Explorer on Windows or Finder on Mac OS X. Also the structure of the files tree can be changed with drag and drop operations inside the view.

Icons

The icons in the working copy view have a small overlaid icon which describes the current state of the resource in the working copy. These state icons are:

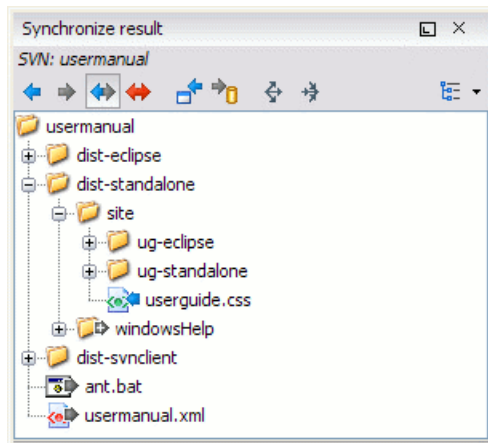
-  Unversioned - The resource marked with this symbol is not under version control. This is how new files are represented when they are created or copied from the file system. Unversioned resources can be filtered from the Working Copy view by setting ignore filters in the Preferences.
-  Added - This resource has been added to version control but has not been committed. This state is obtained after issuing an *Add* command on an unversioned resource.
-  Added with history - This resource has been copied with history. This state is obtained by copying, moving or renaming a resource from the working copy.
-  Modified - The resource has been locally modified since the last update. This is obtained after editing a file and making changes.
-  Deleted - This resource has been deleted from the working copy. This state appears after deleting, moving or renaming files with Subversion.
-  Missing/Incomplete - This resource is in an inconsistent state. If it's *missing*, it means it has been deleted from the file system without Subversion's knowledge. If it's *incomplete*, a check out or update action has probably failed or has been interrupted before finishing. A directory in such a state must be restored with an update action before any other action can be performed.
-  Conflict - This resource has conflicting changes. A resource can be in this state after an update, if it was modified both locally and on the repository and the modifications were overlapping.
-  Tree Conflict - This resource has a tree conflict. A resource can be in this state after an update or merge: the local file is modified but the remote file was removed so the local modifications cannot be committed (the file does not exist on the repository any longer) and a remote version cannot override the local modifications.
-  External - This indicates a mapping of a local directory to the URL of a versioned resource. It is declared with a `svn:externals` property in the parent folder.
-  Normal - A resource with no overlaid icon is an unmodified resource under version control.
-  Grayed - A resource with a grayed icon but no overlaid icon is an ignored resource. It is obtained with the action *Add to svn:ignore*.
-  Switched - This indicates a resource that has been switched from the initial repository location to a new location within the same repository. The resource goes to this state as a result of the Switch action executed from the contextual menu of the Working Copy view.

Synchronize View

General description

The synchronize view is visible in the default layout configuration. It displays the result of a *Refresh* or *Synchronize* operation in a hierarchical form. The nodes represent synchronized or refreshed resources and their status.

Figure 3.58. Synchronize View



Synchronize trees

The results are presented using four tree structures:



- Incoming changes tree - presents items which contain incoming changes. This includes resources modified and committed by others or resources newly added or newly deleted from the repository.
- Outgoing changes tree - presents resources with outgoing changes meaning that they have been modified locally or have been added or deleted from your working copy.
- Incoming - Outgoing tree - includes all the resources with incoming and outgoing changes
- Conflict tree - includes resources with conflicting state meaning they contain both incoming and outgoing changes (pseudo-conflicting state) or they are in a state of real conflict.







A resource which is in a real conflict state will not appear in the *Incoming tree*.

For each file and folder a tooltip is displayed with details like SVN status, full path, current revision number, last changed date, etc. If the tooltips seem annoying by covering useful information they can be disabled from the option *Show tooltip on Working Copy and Synchronize trees*.

Toolbar



The *Synchronize view toolbar* consists of the following buttons:






-  Incoming Mode - filters synchronized resources displaying only the ones with incoming changes.
-  Outgoing Mode - filters synchronized resources displaying the ones with outgoing changes.

-  In-Out Mode - displays resources with incoming or outgoing changes, basically all resources with any type of change.
-  Conflicts Mode - filters synchronized resources displaying the ones in pseudo or real conflict state.
-  Update All - updates all resources with incoming changes. It is disabled when *Outgoing* mode is selected or the synchronization result does not contain resources with incoming changes. It will perform a recursive update on the synchronized resources.
-  Commit All - commits all resources with outgoing changes. It is disabled when *Incoming* mode is selected or the synchronization result does not contain resources with outgoing changes. It will perform a recursive commit on the synchronized resources.
-  Expand All - expands all the descendant nodes of the node currently selected in the view.
-  Collapse All - collapses all the descendant nodes of the node currently selected in the view.

Contextual menu actions








The contextual menu contains the following actions:

- Open in compare editor - if the selected file has a text content type it will be opened in the Compare view and a file differencing will be performed. If the file has a binary content type then the position of the first different byte will be displayed. It is disabled for directories. See also View differences section.
-  Open - it is enabled existing local files and folders. In case of a file the action opens the selected file into the Editor. See also Edit files section. In case of a folder the action opens the selected folder with the system application for folders (for example Windows Explorer on Windows, Finder on Mac OS X, etc).
- Open with ... - Displays the 'Open with...' dialog for specifying the editor in which the selected file will be opened. In case multiple files are selected only external applications can be used to open the files
- Update - it is enabled for resources with incoming changes. Updates all selected resources to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.
- Commit ... - it is enabled for resources with outgoing changes. Commits all selected resources, recursively in the case of directories, to the repository. This action collects the outgoing changes from the selected resources and presents them in a dialog. You can choose exactly what items will be committed by checking or unchecking them in the table. Also you will have to enter a commit comment. See also Sending your changes to the repository section.
-  Show History ... - It will display the *History view* where the log history for the selected resource will be presented. For more details about resource history see the sections Using the resource history view and Request history for a resource.
- Revert ... - it is useful when you want to undo all local changes made to a resource. It is enabled on resources which contain outgoing changes. Read the Revert your changes section for more information.
- Mark Resolved - it is enabled on real conflicting resources. Its function is to tell the Subversion system that you resolved the conflict and the resource can be committed. See also Merge conflicts part.
- Mark as Merged - the action is enabled on pseudo-conflicting resources. It is used after you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the Merge conflicts section for more information on methods to solve the pseudo-conflicts.

- **Override and Update ...** - it is enabled on resources with outgoing changes including the conflicting ones. It is used for dropping any outgoing change and replacing the local resource with the HEAD revision. See [Revert your changes](#) section.
- **Override and Commit ...** - it is enabled on conflicting resources. The action will drop any incoming changes and will send your local version of the resource to the repository. See also [Drop incoming modifications](#).
- **Add** - it is enabled for unversioned resources. It performs a `svn add` command by adding the resources to version control. You can directly commit unversioned items because in this case the *Commit* action will perform first a `svn add` command.
- **Add to svn:ignore ...** - it is also enabled for unversioned resources. It is useful when you decide that some resources, for example some compiler generated Java classes, should be ignored by the Subversion system. The action modifies the value of the `svn:ignore` property of the parent directory. For more information read the section [Ignore resources not under version control](#).
-  **Show SVN Properties** - brings up the Properties view and displays the SVN properties for the selected resource.
-  **File Information ...** - provides additional information for the selected resource from the working copy. For more details please see the section [Obtain information for a resource](#).
-  **Expand All** - expands the selected directories to leaf level.
-  **Collapse All** - collapses all child nodes of the selected tree node.
-  **Refresh** - This action will rescan the selected resources recursively and refresh their status in the working copy view.

Icons

The icons for the items displayed into the synchronized trees are the same icons used in the Syncro SVN Client decorated with overlay images. The overlay icons correspond to the status of the resource as follows:

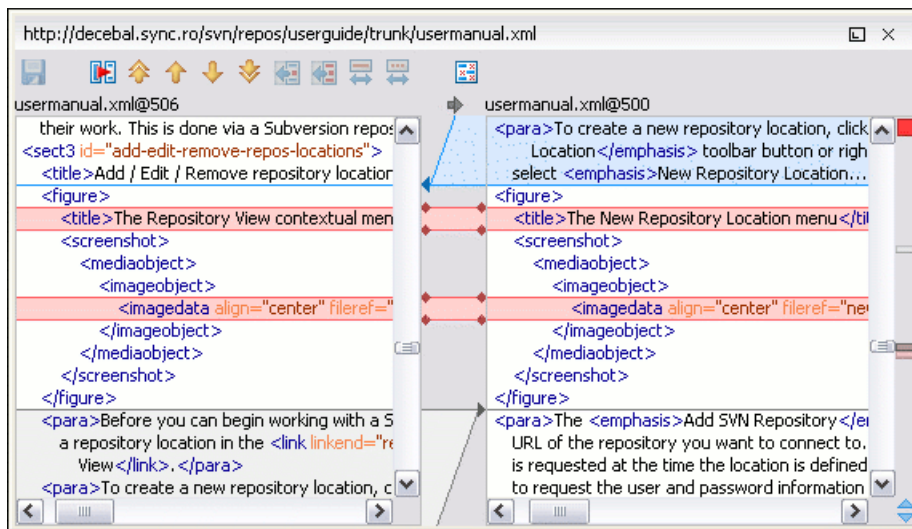
-  **Incoming resource** - resource with incoming changes.
-  **Remote added resource** - resource that was added on the repository and is not present in your working copy.
-  **Remote deleted resource** - resource that no longer exists in the repository.
-  **Outgoing resource** - resource with outgoing changes.
-  **Locally added or unversioned resource** - resource added locally to version control or a resource not yet under version control.
-  **Locally deleted or missing resource** - a resource that you deleted with *Delete* action or that was deleted from the file system in some other way.
-  **Conflicting resource** - resource in a pseudo or real conflicting state.

Compare View

Description

In the Syncro SVN Client there are three types of files that can be checked for differences: text files, image files, and binary files. For the text files and image files you can use the built-in *Compare view*.

Figure 3.59. Compare View






When comparing text, the differences are computed using a line differencing algorithm. The view can be used to show the differences between two files in the following cases:









- After obtaining the outgoing status of a file with a Refresh operation, the view can be used to show the differences between your working file and the pristine copy. In this way you can find out what changes you will be committing.
- After obtaining the incoming and outgoing status of the file with the Synchronize operation, you can examine the exact differences between your local file and the HEAD revision file.
- You can use the Compare view from the History view to compare the local file and a selected revision or compare two revisions of the same file.

If in any of the cases one of the involved files cannot be loaded then you will be prompted with a dialog informing you about the file that cannot be opened. The Compare view contains two editors. Edits are allowed only in the left editor and only when it contains the working copy file. To learn more about how the view can be used in the day by day work you can read the section View differences.

Toolbar

The list of actions available in the toolbar consists of:

-  Save action - it allows you to save the content of the left editor when it can be edited.
-  Perform files differencing - used to perform files differencing on request.
-  Go to first modification - used to navigate to the first difference.

-  Go to previous modification - used to navigate to the previous difference.
-  Go to next modification - used to navigate to the next difference.
-  Go to last modification - used to navigate to the last difference.
-  Copy change from right to left - this action copies the selected change from the right editor to the left editor.
-  Copy all non-conflicting changes from right to left - this action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.
-  Show modification details at word level - because the differences are computed using a line differencing algorithm sometimes is useful to see exactly what words are different in a changed section.
-  Show modification details at character level - useful when you want to find out exactly what characters are different between the two analyzed sections.
-  Ignore whitespaces - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before the strings are compared they are first normalized and then the whitespace at the beginning and the end of the strings is trimmed.

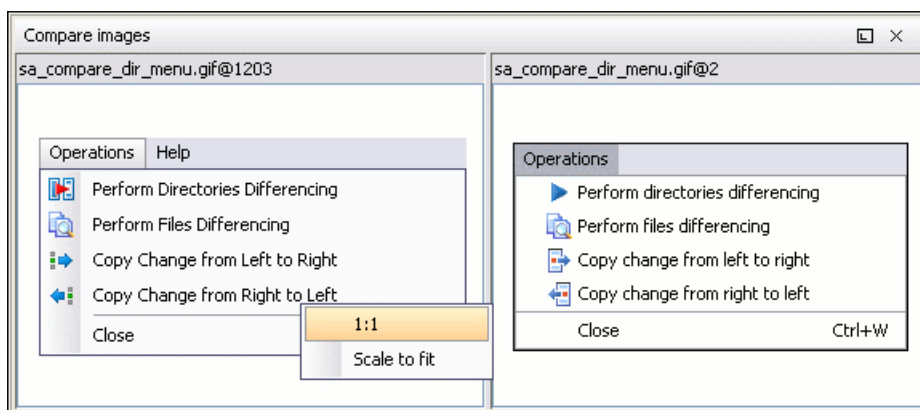
These actions are available also from the Compare menu.

Compare images view

The images are compared using the Compare images view. The images are presented in the left and right part scaled to fit the view's available area. You can use the contextual menu actions to scale the images at its original size or scale it down to fit in the view's available area.

The supported image types are: GIF, JPG / JPEG, PNG, BMP.

Figure 3.60. Compare images view



Editor

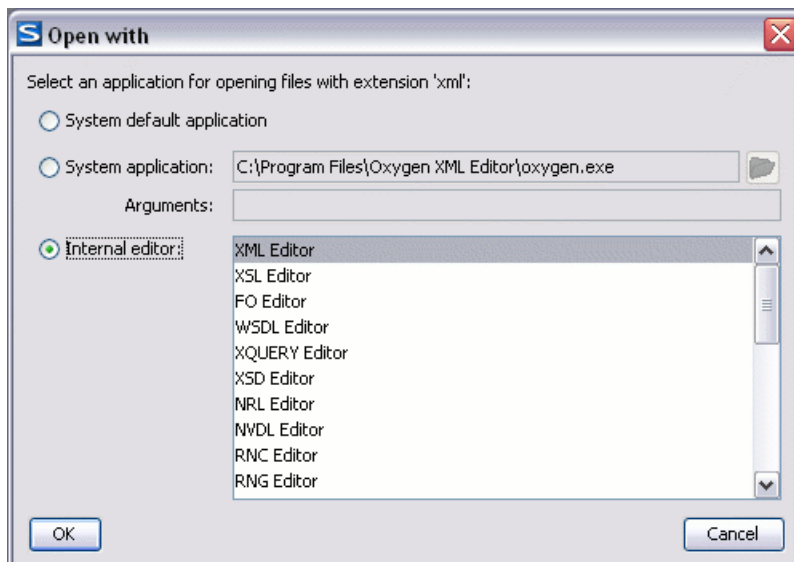
Description

You can open a file for editing in:

- an internal built-in editor
- the default external application associated in the operating system with the type of the file
- an external application installed in the operating system and specified by the path to its executable launcher

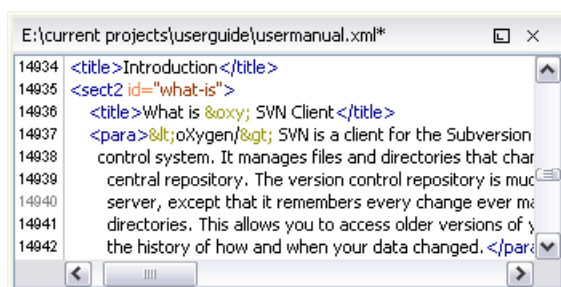
There are default associations between frequently used file types and the internal editors but new associations can be added and the default associations can be modified with the action *Open with* available in the Repository view, the Working Copy view and the History view which opens the following dialog:

Figure 3.61. The editor association dialog



The internal editor can be accessed either from the Working copy view or from the Synchronize view. The editor can also be used from the History view to view a selected revision of a file. In this case there are no edits allowed.

Figure 3.62. Editor View



Only one file can be edited in an internal editor at a time. If you try to open another file it will be opened in the same editor window. The editor has syntax highlighting for known file types. This means that a different color will be used for each type of recognized token in the file. If the content type of the file is unknown you will be prompted to choose the proper way the file should be opened.

After editing the content of the file in an internal editor you can save it to disk by using the *Save* action from the File menu or the Ctrl + S key shortcut. After saving your file you can see the file changed status into Working copy view and Synchronize view.

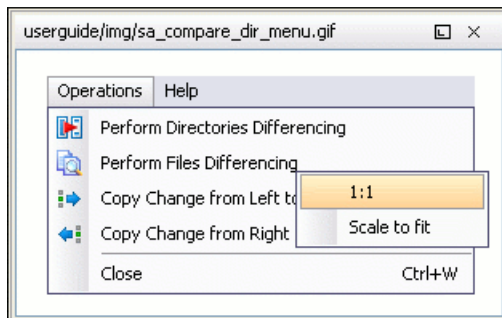
If the internal editor associated with a file type is not the XML Editor then the encoding set in the preference *Encoding for non XML files* is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the encoding of the file.

Image preview

Description

You can view your local files by using the built-in *Image preview component*. The view can be accessed either from the Working copy view, Synchronize view or from the Repository view. It can also be used from the History view to view a selected revision of a image file.

Figure 3.63. Image preview



Only one image file can be opened at a time. If an image file is opened in the *Image preview* and you try to open another one it will be opened in the same window. Supported image types are GIF, JPEG/JPG, PNG, BMP. Once the image is displayed in the *Image preview* panel using the actions from the contextual menu one can scale the image at its original size (1:1 action) or scale it down to fit in the view's available area (Scale to fit action).

History View

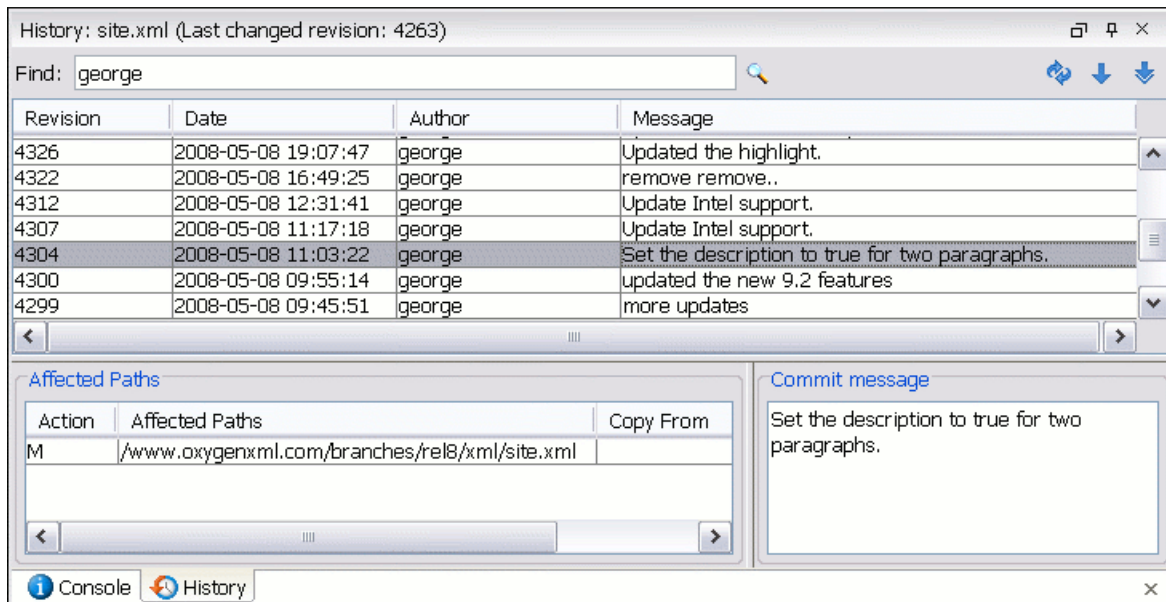
Description

In Subversion, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the *History view* that can be accessed from any of the three views: Repository view menu, Working copy view menu or Synchronize view menu. From the *Repository view* you can display the log history regarding any repository resource. From the *Working copy view* you can display the history of local versioned resources. From the *Synchronize view* you can show the history of any incoming or outgoing resources.

The view consists of three distinct areas:

- The revision table showing revision numbers, date/time of revision, the name of the author, as well as the first line of the commit message.
- The list of resources affected by this revision (modified, added, deleted or changed properties).
- The commit message for the selected revision.

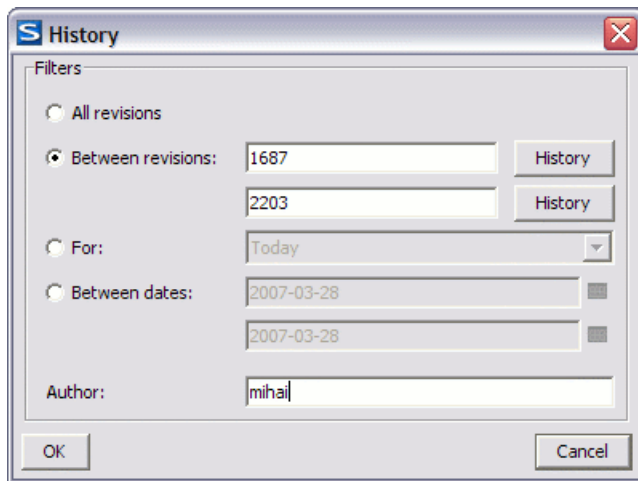
Figure 3.64. History View



History Filters

The History filter dialog

The *History view* does not always show all the changes ever made to a resource because there may be thousands of changes and retrieving the entire list can take a long time. Normally you are interested in the more recent ones. That is why you can specify the criteria for the revisions displayed in the History view by selecting one of several options presented in the *History* dialog which is displayed when you invoke the *Show History* action.

Figure 3.65. History filters dialog

The options for the set of revisions presented in the History view are:


- all the revisions of the selected resource
- only the revisions between a start revision number and an end revision number
- only the revisions added in a period of time like today, last week, last month, etc.
- only the revisions between a start calendar date and an end calendar date
- only the revisions committed by a specified SVN user

The toolbar of the History view has two buttons for extending the set of revisions presented in the view: *Get next 50* and *Get all*.

Note

When using Subversion servers older than version 1.2, a history request may take a very long time because the server will reply with the entire history even if you limited the number of entries to a smaller number.

The History filter field

When only the history entries which contain a specified substring need to be displayed in the History view the filter field displayed at the top of this view is the perfect fit. Just enter the search string in the field next to the label *Find*. Only the items with an author name, commit message, revision number or date which match the search string is kept in the History view. The filter action is executed and the content of the table is updated when the button  Search is pressed.

Features

Single selection actions:

- Open - opens the selected revision of the file into the Editor. This is enabled only for files.
- Open with ... - Displays the 'Open with...' dialog for specifying the editor in which the selected file will be opened.

- Save revision to - saves the selected revision to a file so you have an older version of that file. This option is only available when you access the history of a file, and it saves a version of that one file only.
- Compare with working copy - compares the selected revision with your working copy file. It is enabled only when you select a file.
- Update to revision - updates your working copy resource to the selected revision.
- Check out from revision - gets the content of the selected revision for the resource into local file system.
- Revert changes from this revision - reverts changes made in the selected revision.
- Get Contents - replaces the current version from the working copy with the contents of the selected revision from the history of the file. The BASE version of the file is not changed in the working copy so that after this action the file will appear as modified in a synchronization operation, that is newer than the BASE version, even if the contents is from an older version from history.
- Show Annotation - computes the latest revision number and author name that modified each line of the file up to the selected revision, that is no modification later than the selected revision is taken into account.
- Change Author - changes the name of the SVN user that committed the selected revision.
- Change Message - changes the commit message of the selected revision.

Double selection actions:

- Compare revisions - When the resource is a file the action compares the two selected revisions using the Compare view. When the resource is a folder the action displays the set of all resources from that folder that were changed between the two revision numbers.
- Revert changes from these revisions - Similar to the `svn-merge` command, it merges two selected revisions into the working copy resource. This action is only enabled when the resource history was requested for a working copy item.
- Change Message - changes the commit message of the revisions that are selected.

For more information about the *History view* and its features please read the sections Request history for a resource and Using the resource history view

Annotations View

Description

Sometimes you need to know not only what lines have changed, but also who changed specific lines in a file. This view displays the author and the revision that changed every line in a file. Just click on a line in the editor panel where the file is opened to see the revision that edited that line last time highlighted in the History view and to see all the lines changed by that revision highlighted in the editor panel. Also the entries of the Annotations view corresponding to that revision are highlighted. So the Annotations view, the History view and the editor panel are synchronized. Clicking on a line in one of them highlights the corresponding lines in the other two.

Figure 3.66. The Annotations View

The screenshot displays the Syncro SVN Client interface with the following components:

- Main Editor:** Shows XML code for `htmlCustomLayer.xml`. Annotations are visible as colored highlights on the code, corresponding to different SVN revisions.
- Annotations View (Right Pane):** Lists annotations for various revisions. A detailed view for revision 1886 is shown below the list:

Author	sorin
Revision	294
Date	2006-03-24 09:21:33
Lines	193
- History View (Bottom Pane):** Shows a table of revisions for `htmlCustomLayer.xml`. The last changed revision is 1986.

Revision	Date	Author	Message
1986	2007-02-06 12:31:03	sorin	Corrected Docbook image paths in HTML ...
1886	2007-01-23 17:33:36	sorin	Fix name and target of link to home page in the ...
1851	2007-01-22 15:44:16	sorin	Removed 'oxygen' from SVN Client manual.
1380	2006-11-16 17:10:36	sorin	Make title of HTML pages of website help a link t...
579	2006-05-26 15:17:17	test	Removed include of jhCustomLayer.
448	2006-05-15 09:47:33	test	Documented more new features for release 7.2.
294	2006-03-24 09:21:33	sorin	Removed dependency.
292	2006-03-22 16:53:21	sorin	Refactored cust.layers.
- Affected Paths:** Shows a table with columns for Action, Affected Paths, and Copy From. The action is 'M' for the path `/userguide/trunk/htmlCustomLayer.xml`.
- Commit message:** A text area containing the message: "Fix name and target of link to home page in the header of all pages of SVN Client".

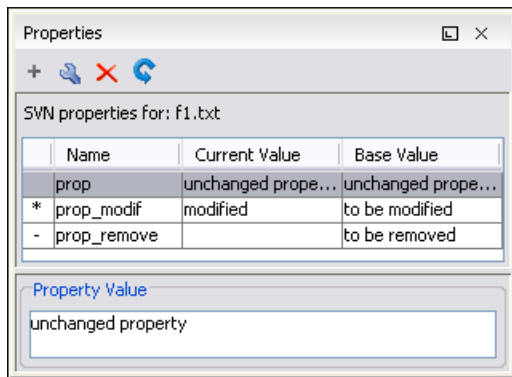
The annotations of a file are computed with the action *Show Annotation* available on the right click menu of the History view and the Repository view.

If the file has a very long history the computation of the annotation data can take long. If you want only the annotations of a range of revisions you can specify the start revision and the end revision of the range in a dialog similar with the History filter dialog that will be displayed in the History view. The action is called *Show Annotation* and is available on the right click menu of the Working Copy view.

Properties View

Description

The properties view presents the Subversion properties for the currently selected resource from either the *Working copy view* or the *Synchronize view*.

Figure 3.67. The properties View

Above the table it is specified the currently active resource for which the properties are presented. Here you will also find a warning when an unversioned resource is selected.

The table in which the properties are presented has four columns:

- State - can be one of
 - (empty) - normal unmodified property, same current and base values.
 - *(asterisk) - modified property, current and base values are different.
 - +(plus) - new property.
 - -(minus) - removed property.
- Name - the property name.
- Current value - the current value of the property.
- Base value - the base(original) value of the property.

The *svn:externals* property

The *svn:externals* property can be set on a folder or a file. In the first case it stores the URL of a folder from other repository.

In the second case it stores the URL of a file from other repository. The external file will be added into the working copy as a versioned item. There are a few differences between directory and file externals:

- The path to the file external must be in a working copy that is already checked out. While directory externals can place the external directory at any depth and it will create any intermediate directories, file externals must be placed into a working copy that is already checked out.
- The file external's URL must be in the same repository as the URL that the file external will be inserted into; inter-repository file externals are not supported.
- While commits do not descend into a directory external, a commit in a directory containing a file external will commit any modifications to the file external.

The differences between a normal versioned file and a file external:

- File externals cannot be moved or deleted; the `svn:externals` property must be modified instead; however, file externals can be copied.





A file external shows up as a X in the switched status column.

Warning

Incomplete support - In subversion 1.6 it is not possible to remove a file external from your working copy once you have added it, even if you delete the `svn:externals` property altogether. You have to checkout a fresh working copy to remove the file.



Toolbar / Contextual menu

The properties view toolbar and contextual menu contain the following actions:

-  Add a new property - This button invokes the *Add property* dialog in which you can specify the property name and value.
-  Edit property - This button invokes the *Edit property* dialog in which you can change the property value and also see its original(base) value.
-  Remove property - This button will prompt a dialog to confirm the property deletion. You can also specify if you want to remove the property recursively.
-  Refresh - This action will refresh the properties for the current resource.

Console View

Description

The *Console View* shows the communication between your client and the Subversion repository. The output is expressed as subcommands to the Subversion server and simulates the Subversion command line notation. For a detailed description of the Subversion console output read the *SVN User Manual*. In the right toolbar there are available a *Clear*  action which clears the content of the view and a *Lock/Unlock* scroll  action which disables the automatic console scrolling.

The maximum number of lines displayed in the console (the length of the buffer) can be modified from Preferences. By default this is set to 100.

Help View

Description

The *Help view* is a dynamic help window. It changes its content displaying the help section referring to the currently selected view. As you change the focused view you will be able to read a short description of it and its functionality.

The Revision Graph of a SVN Resource

The history of a SVN resource can be watched on a graphical representation of all the revisions of that resource together with the tags in which the resource was included. The graphical representation is identical to a tree structure and is easy to view.


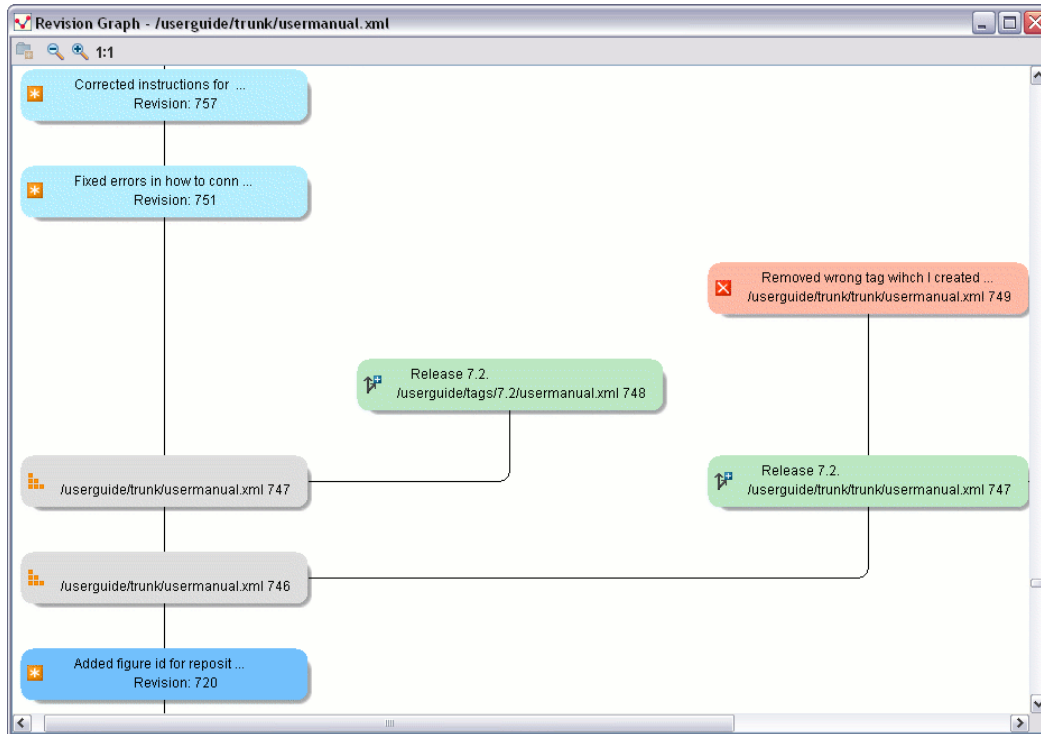






The graphical representation of a resource history is invoked with the action  Revision graph available on the right click menu of a SVN resource in the *Working Copy* view and the *Repository* view.

Figure 3.68. The Revision Graph of a File Resource



In every node of the revision graph an icon and the background color represent the type of operation that created the revision represented in that node. Also the commit message associated with that revision, the repository path and the revision number are contained in the node. The tooltip displayed when the mouse pointer hovers over a node specifies the URL of the resource, the SVN user who created the revision of that node, the revision number, the date of creation, the commit message, the modification type and the affected paths.

The types of nodes used in the graph are:

added resource	the icon for a new resource added to the repository () and green background
copied resource	the icon for a resource copied to other location, for example when a SVN tag is created () and green background
modified resource	the icon for a modified resource () and blue background
deleted resource	the icon for a resource deleted from the repository () and red background
replaced resource	the icon for a resource removed and replaced with another one on the repository () and orange background
indirect resource	the icon for a revision from where the resource was copied or an indirectly modified resource, that is a directory in which a resource was modified () and grey background; the <i>Modification type</i> field of the tooltip specifies how that revision was obtained in the history of the resource

A directory resource is represented with two types of graphs:

- simplified graph lists only the changes applied directly to the directory
- complete graph lists also the indirect changes of the directory resource, that is the changes applied to the resources contained in the directory

Figure 3.69. The Revision Graph of a Directory (Direct Changes)

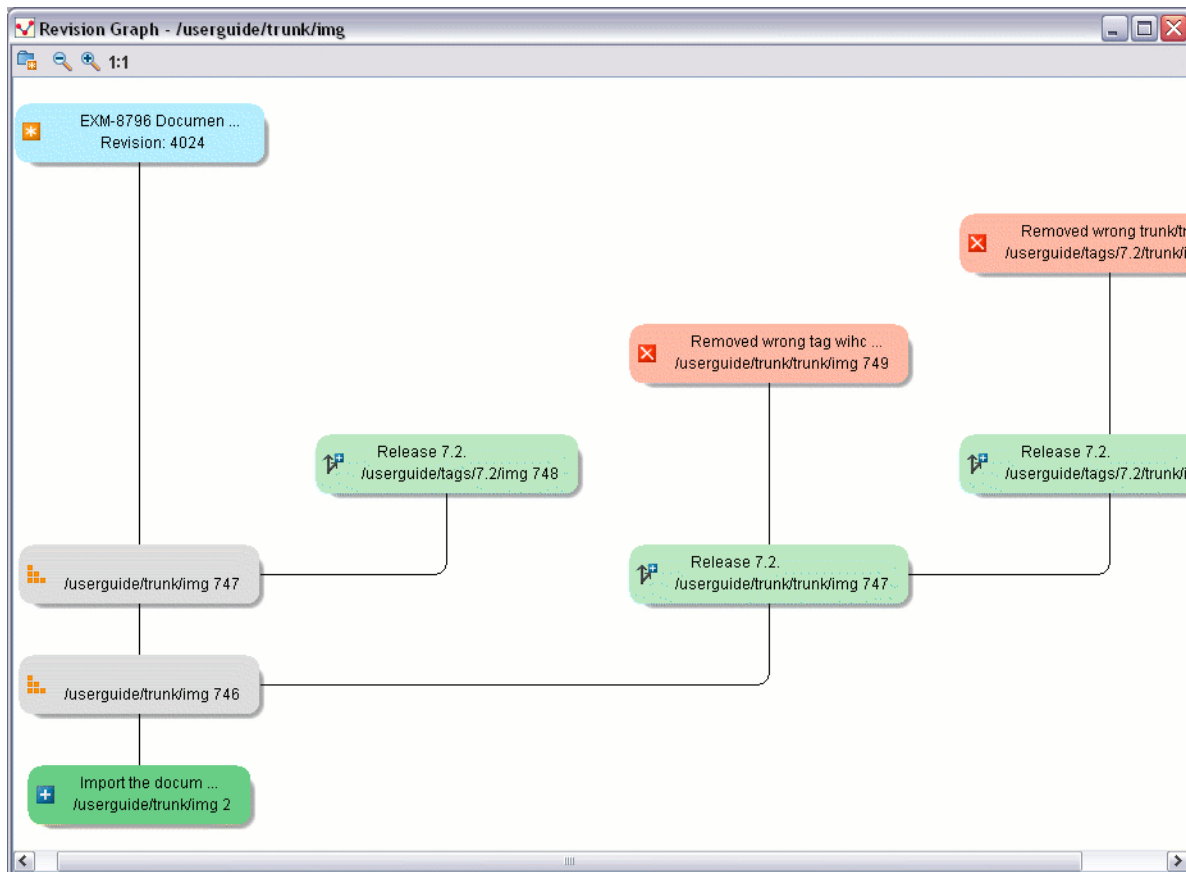
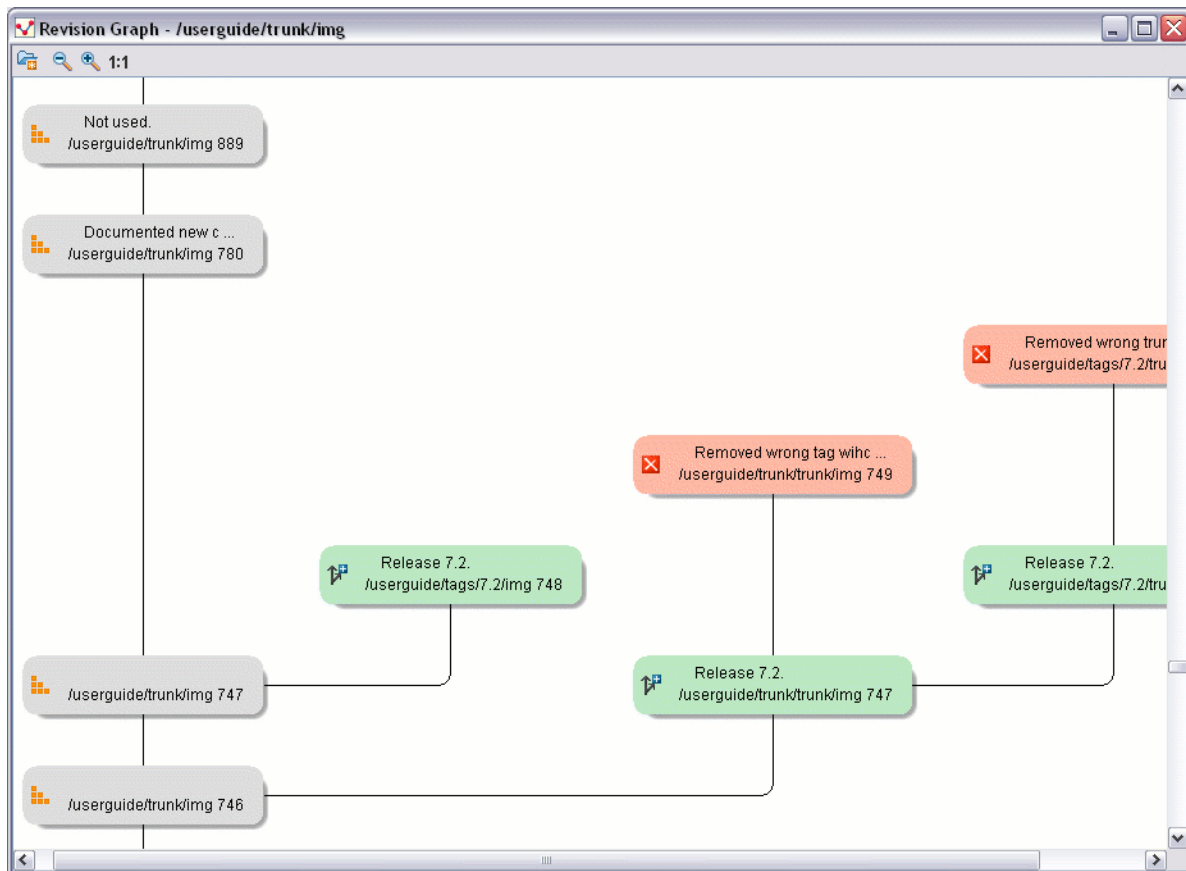





Figure 3.70. The Revision Graph of a Directory (Also Indirect Changes)

You can switch between the two graphs with the  Show/Hide indirect modifications button of the toolbar of the *Revision graph* dialog.

The toolbar also contains buttons for zooming in and zooming out the graph and also for resetting the scale of the graphical representation. When the font has the minimum size (zoom out to last level) the text content is not displayed in nodes anymore, only the icon is displayed and the distances between graph nodes are smaller so that the graph has a more compact representation.

The right click menu of a node of the graph contains the following actions:

-  Open available only for files, opens the selected revision in the editor panel.
-  Show History available for both files and directories, displays the history of the resource in the History view.
- Compare with HEAD available only for files, compares the selected revision with the HEAD revision and displays the result in the diff panel.
-  Check Out available only for directories, checks out the selected revision of the directory.

When two nodes are selected in the revision graph of a file the right click menu of this selection contains only one item: *Compare* for comparing the two revisions corresponding to the selected nodes. If the resource for which the revision graph was built is a folder then the right click menu displayed for a two nodes selection also contains the item *Compare* but it computes the differences between the two selected revisions as a set of directory changes. The result

is displayed in the Directory Change Set view in the same way as for the compare action invoked from the History view on two revisions of a folder.

Warning

Generating the revision graph of a resource with many revisions may be a slow operation. You should enable caching for revision graph actions so that future actions on the same repository will not request the same data again from the SVN server which will finish faster.

Command line interface cross reference

This section specifies the equivalent Subversion commands for each action in the Syncro SVN Client action.

Actions commands reference

Checkout

svn checkout [--revision rev] URL PATH *--revision rev* specifies the desired revision(if necessary).
URL is the repository URL you want to check out from.
PATH is the location on the file system.

Update

svn update [--revision rev] PATH *--revision rev* specifies the desired revision(if necessary).
PATH is the location on the file system of the resource to update.

There are two behaviours for the update action in Syncro SVN client. If invoked from the Synchronize View, it updates the resources to the HEAD revision. If invoked from the Working Copy view it always updates to the HEAD revision.

Commit

svn commit -m "log message" [--no-unlock] PATH... *-m "log message"* specifies the commit comment.
--no unlock specifies that the resource should keep locks after commit if this is the case.
PATH is the location on the file system of the resource to commit. Can be more than one.

Diff

svn diff --revision rev1:rev2 PATH *--revision rev1:rev2* specifies the desired revisions to be compared.
PATH is the location on the file system of the resource to be compared.

If you use the *Compare with latest from HEAD* from the Working copy view you will be comparing the local file with the HEAD revision file. If you use *Compare with BASE revision* the local file will be compared with the pristine copy. From the Synchronize view you can compare the working copy file with the HEAD revision file. You can choose to compare the local file with an older revision or two revisions of the same file from the History view.

Show History

svn log [--revision rev1:rev2] [--limit N] --verbose PATH *--revision rev1:rev2* - specifies the range of revisions for which to obtain the log.
--limit N - limits the number of log messages to N.
--verbose - gives detailed information about the operation.

Syncro SVN Client uses by default the *--limit* option in order to obtain only 50 log messages.

Refresh

svn status --verbose PATH *--verbose* - specifies that the status of all files should be reported.
PATH - The location on the file system to get status for.

Synchronize

svn status --show-updates PATH *--show-updates* - get the resource status by contacting the repository.
PATH - The location on the file system to get status for.

Import

svn import -m "log message" PATH URL *-m "log message"* - specifies the commit log message
PATH - the local path to the resource on the file system.
URL - the URL on the repository where the resource will be imported.

Export

svn export [--revision rev] URL PATH *--revision rev* specifies the desired revision(if necessary).
URL is the repository URL you want to export from.
PATH is the location on the file system where to export.

Information

svn info [--revision HEAD] PATH / URL *--revision HEAD* - specifies that the information will be for the HEAD revision of the resource.
PATH - the local file system path to the resource.
URL - the repository URL for the resource.

This command can obtain information for a resource from a working copy or from a Subversion repository.

Add

svn add PATH... *PATH*- the local file system path for the unversioned resources to be added to version control. More than one can be specified.

Add to svn:ignore

**svn propset svn:ignore PATH
PARENTPATH**

svn:ignore - the predefined property name for ignoring resources.

PATH - the relative path from the working copy root for the resource to be ignored.

PARENTPATH - the path to the parent of the resource to be ignored.

Delete

svn delete --recursive PATH | URL

--recursive - specifies that the operation should be performed recursively.

PATH- the local file system path for the resource to delete.

URL- the repository URL for the resource to delete.

This command can delete resources from a working copy or from a Subversion repository.

Copy

**svn copy (SRCPATH DSTPATH)
| (SRCURL DSTURL)**

SRCPATH - the working copy path of the resource to be copied.

DSTPATH - the working copy path to be copied to.

SRCURL - the repository path of the resource to be copied.

DSTURL - the repository path to be copied to.

Move / Rename

**svn move (SRCPATH DSTPATH)
| (SRCURL DSTURL)**

SRCPATH - the working copy path of the resource to be moved.

DSTPATH - the working copy path to be moved to.

SRCURL - the repository path of the resource to be moved.

DSTURL - the repository path to be moved to.

Mark resolved

svn resolved --recursive PATH

--recursive - specifies that the operation should be performed recursively.

PATH - the path to the resource in the local working copy.

Revert

svn revert [--recursive] PATH

--recursive - specifies that the operation should be performed recursively.

PATH - the local working copy path to revert.

Cleanup

svn cleanup PATH

PATH - the working copy path to cleanup.

Show / Refresh Properties

svn proplist *PATH* & **svn propget** *PROPNAME PATH*
PATH - the local path for the resource
PROPNAME - the property name.

First you can discover the property names with **svn proplist**, then you can obtain their values with **svn propget**.

Branch / Tag

svn copy -m "log message" URL1 URL2 or *svn copy -m "log message" URL1@rev1 URL2* or *svn copy -m "log message" PATH URL*
-m "log message" - the commit comment
URL1 - the source repository URL.
rev1 - the revision of the source.
URL2 - the destination repository URL.
PATH - the source working copy path.
URL - the destination repository URL.

Merge

Merge - *svn merge [--dry-run] rev1:rev2 URL PATH* or *svn merge [--dry-run] URL1@rev1 URL2@rev2 PATH*
--dry-run - specifies that the operation will be simulated without making any modifications.
URL - the repository URL for the resource to merge.
URL1 - the repository URL for the start branch to merge.
rev1 - the start revision for the resource to merge.
URL2 - the repository URL for the end branch to merge.
rev2 - the end revision for the resource to merge.
PATH - the destination path in the working copy for the result of the merge

Scan for locks

svn status --show-updates --verbose PATH
--show-updates - get the resource status by contacting the repository.
--verbose - specifies that the status of all files should be reported.
PATH - The location on the file system to get status for.

The command will obtain the repository status for all the resources in the path.

Lock


svn lock [--force] [-m "log message"] PATH
--force - forces(steals) the lock
-m "log message" - the lock comment.
PATH - the path to the file from the working copy..

URL - The SVN URL corresponding to the resource.

Chapter 4. Text editor specific actions

Syncro SVN Client provides user actions common in any text editor:

Undoing and redoing user actions

- Edit → Undo (**Ctrl+Z**) or the toolbar button  Undo to reverse a maximum of 100 editing actions to return to the preceding state. Complex operations like "Replace All", "Indent selection", etc are treated as a single undo event.
- Edit → Redo (**Ctrl+Y for Windows, Ctrl+Shift+Z for Mac OSX and Linux**) to recreate a maximum of 100 editing actions that were undone by the Undo function.

Copying and pasting text

- Edit → Cut (**Ctrl+X**) to remove the current selected node from the document and places it in the clipboard.
- Edit → Copy (**Ctrl+C**) to place a copy of the current selection in the clipboard as RTF. All text attributes such as color, font or syntax highlight are preserved when pasting into another application.
- Edit → Paste (**Ctrl+V**) to place the current clipboard content into the document at the cursor position.
- Edit → Select All (**Ctrl+A**) selects the entire body of the current document, including whitespace preceding the first and following the last character.

Finding and replacing text in the current file


The Find/Replace dialog

The Find/Replace dialog opened with the menu entry Edit → Find/Replace... (**Ctrl+F**) enables you to define "search for" or "search for and replace" operations on the current document. The find works on multiple lines, which means a find match can cover characters on more than one line. To insert a new line in the find or replace text area press **CTRL + Enter** instead of **Enter**. The replace operation can bind Perl 5 regular expression group variables (\$1, \$2, etc.) from the find match. For example to replace the tag with attributes called *tag-name* with the tag *tag-name1* use as text to find `<tag-name(\s+)(.*)>` and as replace text `<tag-name1$1$2>`.

- Find occurrences of a word or string of characters including white spaces represented on a line or on multiple lines and highlight the position in the editor.
- Replace occurrences of target defined in the Text to find area with a word or string of characters, including white spaces, that can be on a line or on multiple lines, defined in the Replace with area.
- Replace all occurrences of a word or string of characters including white spaces that can be on a line or on multiple lines.

Figure 4.1. Find/Replace Dialog

Complete the dialog as follows:

Text to find	<p>The target character string to search for. The string can be on a line or on multiple lines. Special characters like newline and tab can be inserted using the contextual menu.</p> <p>You can search for Unicode characters specified in the <code>\uNNNN</code> format. Also, hexadecimal notation (<code>\xNNNN</code>) and octal notation (<code>\0NNNN</code>) can be used. Note that in this case you have to check the <i>Regular expression</i> checkbox. For example to search a space character you can use <code>\u0020</code> code.</p>
Replace with	<p>The character string with which to replace the target. The string for replace can be on a line or on multiple lines. Special characters like newline and tab can be inserted using the contextual menu.</p> <p>Unicode characters can also be used in the <i>Replace with</i> area.</p>
The Find and Replace history buttons	<p>The last find and replace operations history is available using the  History buttons from the top of the find or replace text area.</p> <p>The character string with which to replace the target. It may contain regexp group markers if the search expression is a regular expression and the regular expression checkbox is checked.</p>
Direction	Specify if the search direction is from current position to end of file (forward direction) or to start of file (backward direction).
Scope	Specify if the search is executed on all file or only on the lines that were selected when the dialog was invoked. If the selection was on a single line the search is executed on all the file.

Find	Execute a find operation for the next occurrence of the target and stop.
Replace	Execute a replace operation for the target followed by a find operation for the next occurrence.
Replace all	Execute a replace operation in the entire scope of the document.
Replace to end	Execute a replace operation starting from current target until the end of the document, in the direction specified by the current selection of the Direction switch (forward or backward).
Case sensitive	When checked, operations are case sensitive.
Whole words only	When checked only whole occurrences of a word will be included in the operation.
Search also in tags	When checked, operation will include content of the start and end tags of the XML elements.
Incremental	When checked, search operation is started for every letter typed in or deleted. The first match that obeys the checked conditions will be highlighted.
Regular expression	When checked allows using any regular expression in PERL syntax.
Wrap around	Continues the find from the start (end) of the document after reaching the end (start) if the search is in forward (backward) direction.

Keyboard shortcuts for finding the next and previous match

Navigation from a find match to the next one or the previous one is very easy with two keyboard shortcuts: F3 and Shift F3. They are useful to quickly repeat the last find action performed with the Find/Replace dialog, taking into account the same find options set there through check boxes.

Find → Find next (**F3**) performs another search in forward direction using the last search configuration.

Find → Find previous (**Shift+F3**) performs another search in backward direction using the last search configuration.

VI editor actions

The Text mode editor implements many actions known from the VI text editor:

Ctrl+Delete (Meta+Delete on Mac)	Delete next word
Ctrl+Backspace (Meta+Backspace on Mac)	Delete previous word
Ctrl+W (Meta+W on Mac)	Cut previous word
Ctrl+K (Meta+K on Mac)	Cut to end of line

Dragging and dropping the selected text

To move a whole region of text to other location in the same edited document just select the text, drag the selection by holding down the left mouse button and drop it to the target location.

Exiting the application

File → Exit (**Ctrl+Q**) : Terminates Syncro SVN Client . Session information such as the current Project, open Documents and Option settings is made persistent. When Syncro SVN Client is re-opened, the persistence information returns to the last saved state.

Chapter 5. Configuring the application

Importing/Exporting Global Options

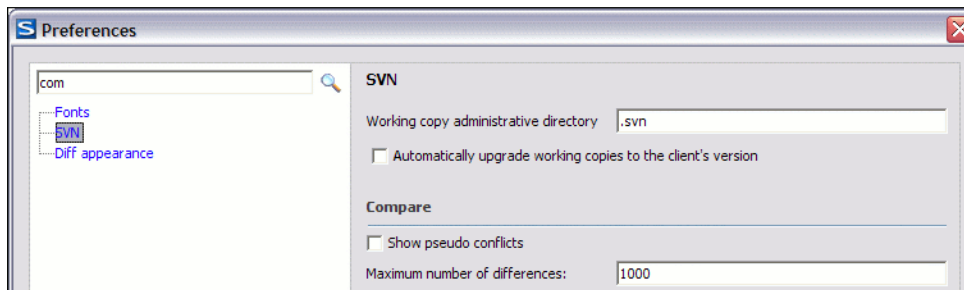
In the Options menu you can find the Import/Export preferences operations which allow you to move your global preferences in XML format from one computer to another.

Preferences

Once the application is installed you can use the Preferences dialog accessed from Options → Preferences to customize the application for your requirements and network environment.

There is a search field available in the dialog for selecting only the preferences panels containing required words in the panel title or in the text of a label or a button contained in the panel. If you want to go to first match press Enter, Up Arrow or Down Arrow.

Figure 5.1. The Search field from the Preferences dialog

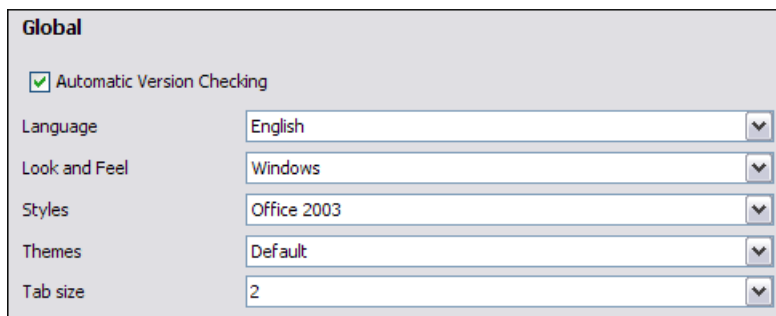


You can always revert modifications to their default values by using the Restore Defaults button, available in each preference page.

Global

The Global preferences panel is opened from menu Options → Preferences+Global

Figure 5.2. The Global preferences panel



Automatic Version Checking

When enabled, checks the availability of new Syncro SVN Client versions at <http://www.syncrosvnClient.com/> [<http://www.syncrosvnClient.com/>].

Language

The application supports a number of languages for localization of the GUI. Select Options → Preferences → Global+Language drop-list to display the language choices.

 **Note**

After restarting the application, if some GUI labels are not rendered correctly you will need to install the corresponding language pack from your OS installation kit.

Look and Feel

Use this option to change graphic style (look and feel) of the GUI.

Styles

On Windows there are available the following styles:

- Office 2003
- Vsnet
- Eclipse
- Xerto
- Default

 **Note**

After changing the style one has to restart the application in order for the modification to take effect.

On Linux there are available the following styles:

- Eclipse
- Default
- GTK+ (not recommended due to stability and paint issues)

 **Note**

After changing the style one has to restart the application in order for the modification to take effect.

On Mac OS X this option is not available.

Themes

On Windows this option is enabled only if the Office 2003 or Default styles. In these cases, the following themes are available:

- Normal Color
- Home Stead
- Metallic
- Default

- Gray

On Linux this option is not available.

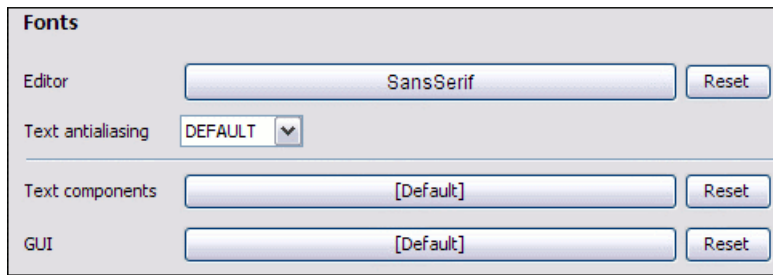
On Mac OS X this option is not available.

Tab size	The number of spaces used to render one tab character in the compare panel and in the editor panel. Use this option to configure the indentation of text in these panels which display the text content of a file.
Show hidden files and directories	Show system hidden files and folders in the file and directory browsers. This setting is not available on Mac OS X.

Fonts

The Fonts preferences panel is opened from menu Options → Preferences+Fonts

Figure 5.3. The Fonts preferences panel

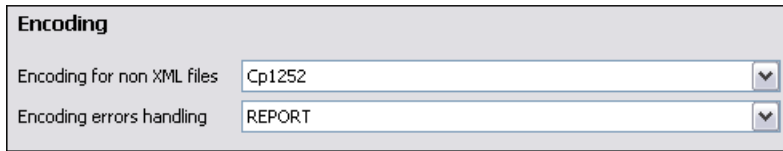


Editor	Use this option to select the font family and size used to display text in the editor.
Text antialiasing	Enable text antialiasing at the specified level. On JVM versions prior to 1.6 this combo box contains only the values Default, On and Off. Default means that Syncro SVN Client will not set anything special for text antialiasing but the JVM will use the setting of the operating system if it is available. The On option sets the text antialiasing to pixel level and the Off option disables it. Starting with version 1.6 the combo contains also values specific for sub pixel antialiasing, like GASP, LCD_HRGB, LCD_VRGB which sets the respective antialiasing mode for the text displayed in the Syncro SVN Client editors and views.
Text components	Use this option to select the font family and size used to display text in text components. After changing the font one has to restart the application.
GUI	Use this option to select the font family and size used to display GUI labels. After changing the font one has to restart the application.

Encoding

The Encoding preferences panel is opened from menu Options → Preferences+Encoding

Figure 5.4. The Encoding preferences panel

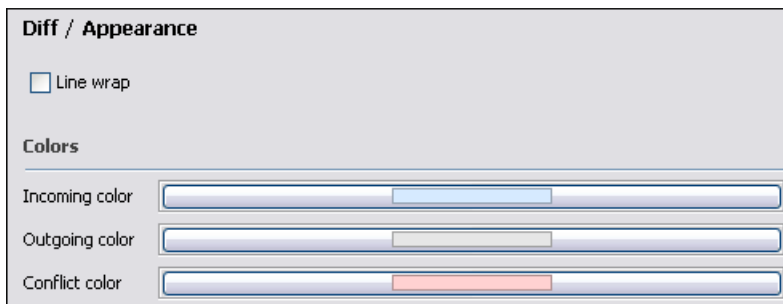


- | | |
|----------------------------|--|
| Encoding for non XML files | This option defines the default encoding to be used when opening non XML documents. This is necessary because non XML files have a large variety of formats and there is no standard mechanism for declaring the encoding that should be used for opening and saving the file. In case of XML files the encoding is usually declared at the beginning of the file in a special declaration or it assumes the default value UTF-8. |
| Encoding errors handling | <p>This option defines how to handle characters that cannot be represented in the specified encoding of the document when the document is opened. The available options are:</p> <ul style="list-style-type: none"> • REPORT - Show an error dialog with the character that cannot be represented in the specified encoding and allow the user to decide how to continue (ignore that character, replace it with a standard replacement character). This is the default option. • IGNORE - The character is ignored and it will not be included in the document displayed in the editor panel. • REPLACE - Replace the character with a standard replacement character. For example if the encoding is UTF-8 the replacement character has the Unicode code FFFD, and if the encoding is ASCII the character code is 63. |

Diff Appearance

The Diff Appearance preferences panel is opened from menu Options → Preferences+Diff+Appearance

Figure 5.5. The Diff appearance preferences panel



- | | |
|----------------|---|
| Line wrap | If checked the lines presented in the two diff panels are wrapped at the right margin of each panel so that no horizontal scrollbar is necessary. |
| Incoming color | The color used for incoming changes on the vertical bar that shows the differences between the files compared. |

Outgoing color The color used for outgoing changes on the vertical bar that shows the differences between the files compared.

Conflict color The color used for conflicts on the vertical bar that shows the differences between the files compared.

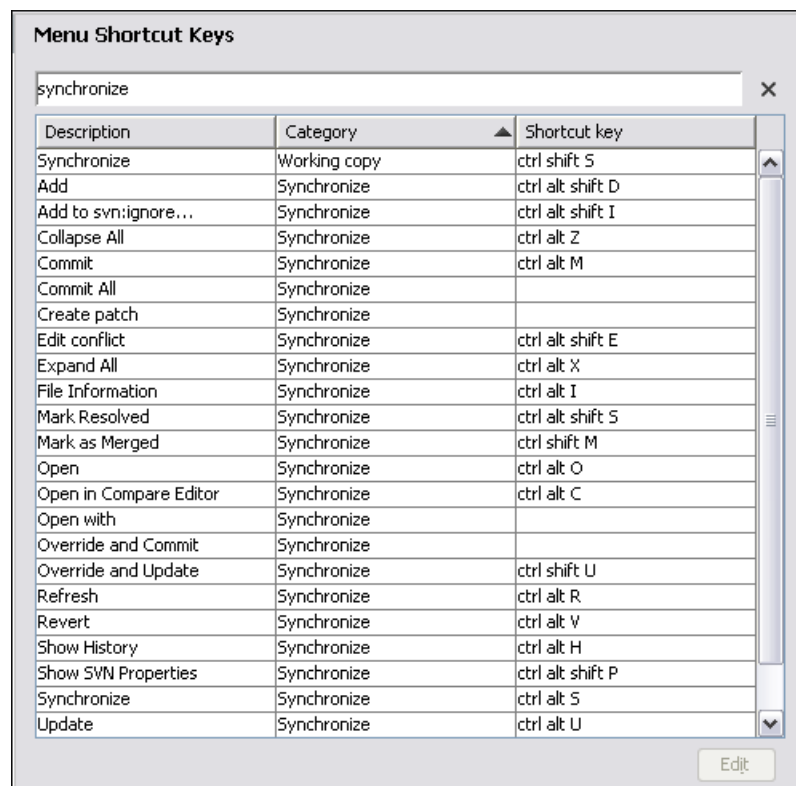
Menu Shortcut Keys

The user can configure in one place the keyboard shortcuts available for menu items available in Syncro SVN Client . The current shortcuts assigned to menu items are displayed in the following table.

The user can search an operation using the filter field by the operation's description, category or shortcut key.

The Menu Shortcut Keys preferences panel is opened from menu Options → Preferences+Menu Shortcut Keys

Figure 5.6. The Menu Shortcut Keys preferences panel



Description A short description of the menu item operation.

Category The shortcuts are classified in categories for easier management. For example the "Cut" operation for the source view is distinguished from the tree view one by assigning it to a separate category.

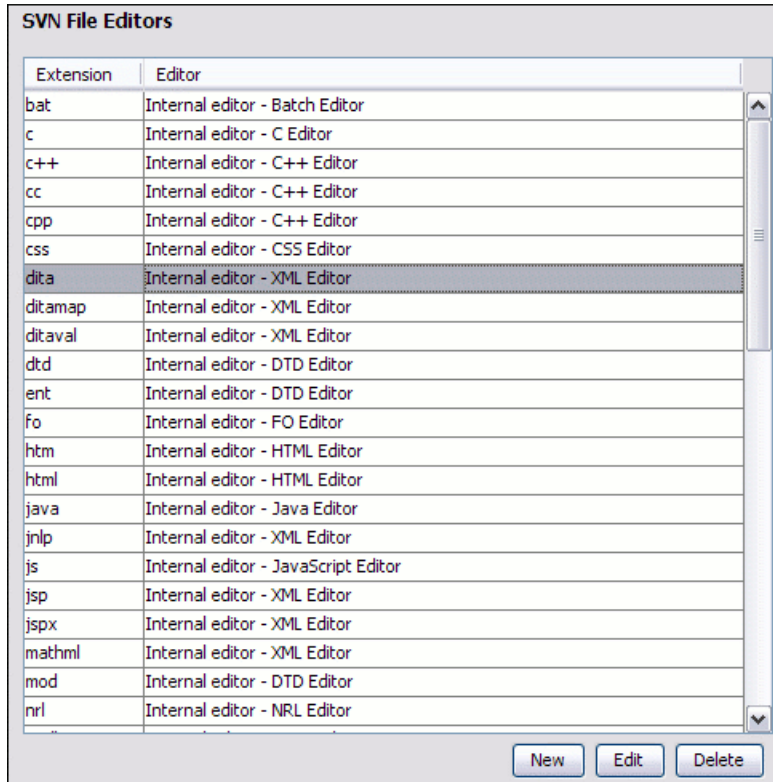
Shortcut key The keyboard shortcut that launches the operation. Double-clicking on a table row or pressing the "Edit" button allows the user to register a new shortcut for the operation displayed on that row.

SVN File Editors

Each type of file is associated with an editor which opens files of that type for editing. The editor can be the built-in one specially provided for the file type (for example the internal XML editor, the internal XSLT editor, the internal XSL-FO editor, etc) or an external application installed on the computer, either the default system application associated with that file type in the operating system or other particular application specified by the path to its executable file. The list of all the associations file type - editor is displayed in the panel *SVN File Editors*.

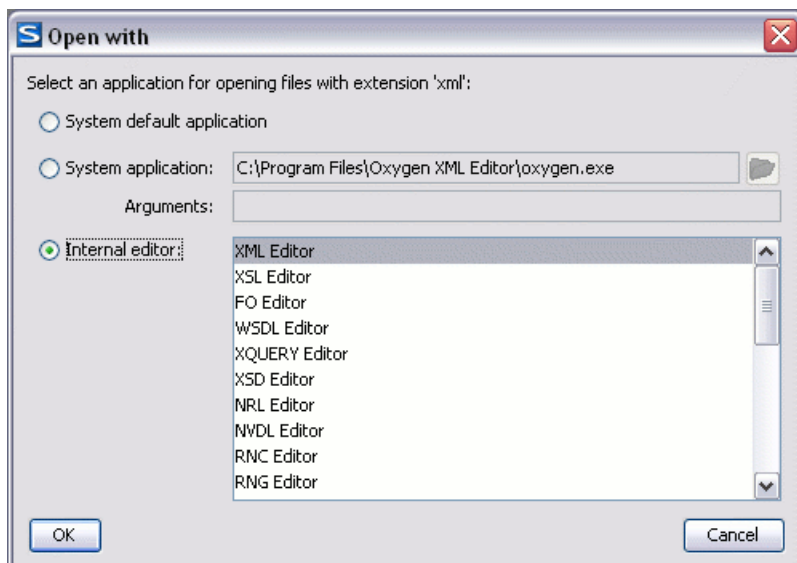
The SVN File Editors preferences panel is opened from menu Options → Preferences+SVN File Editors

Figure 5.7. The SVN File Editors preferences panel



The *Edit* button under the table or a double click on a table row opens a dialog for specifying the editor associated with the file type.

The same dialog is displayed and after an Open with action on a selected file from Syncro SVN Client.

Figure 5.8. The Open With dialog for file type - editor associations

In this dialog are offered three options for opening a file:

- System default application - this option allows you to open the selected file using the application that is associated with that file extension by default in your operating system;
- System application - opens the selected file using an external application that you have to specify by the path of its executable file. Also, you can specify some arguments for that application, if they are needed. This option also works for directories, if you wish to choose a file browser other than the system default.
- Internal editor - this options allows you to select an editor type from the built in editors that Syncro SVN Client comes with. By default, this option is disabled when selecting directories.

Note

For opening a directory, you can choose one of the first two options. The System default application will be used to open that directory in the system built-in file browser (i.e. Windows Explorer for Windows, Finder for Max OS X etc.), and the second one for opening that directory using a file browser other than the system default.

If a file type is associated with an internal editor other than the XML Editor then the encoding set in the preference *Encoding for non XML files* is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning of the XML file in a special declaration or it assumes the default value UTF-8 but in case of non XML files there is no standard mechanism for declaring the encoding of the file.

HTTP / Proxy Configuration

Some networks use Proxy servers to provide Internet Services to LAN Clients. Clients behind the Proxy may therefore, only connect to the Internet via the Proxy Service. The Proxy Configuration dialog enables this configuration. If you are not sure whether your computer is required to use a Proxy server to connect to the Internet or the values required by the Proxy Configuration dialog, please consult your Network Administrator.

Open the HTTP / Proxy Configuration panel by selecting Options → Preferences+HTTP / Proxy Configuration.

Figure 5.9. The HTTP / Proxy Configuration preferences panel

Complete the dialog as follows:

Direct connection When checked the HTTP and HTTPS connections go directly to the target host without going through a proxy server.

Use system settings When checked the HTTP and HTTPS connections go through the proxy server set in the operating system. For example on Windows the proxy settings are the ones available in Internet Explorer.

Warning

The system settings for the proxy cannot be read correctly from the operating system on some Linux systems. The system settings option should work properly on Gnome based Linux systems but it does not work yet on KDE based ones as the Java virtual machine does not offer the necessary support yet [http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6385839].

Manual proxy configuration When checked the HTTP and HTTPS connections go through the proxy server specified in the fields *Address* and *Port* of the section *Web Proxy (HTTP / HTTPS)*. Also this section specifies the hosts to which the connections must not go through a proxy server.

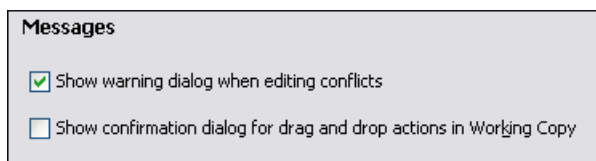
Web Proxy authentication (HTTP / HTTPS)	In this section one must set the user and password necessary for authentication with the proxy server. The user and password set here will be used both in case of manual proxy configuration and in case of system settings selected above.
SOCKS Proxy	In this section one must set host and port of a SOCKS proxy through which all the connections must pass. If the <i>Address</i> field is empty the connections will use no SOCKS proxy.
SSL authentication with client certificate	If checked and the SVN server accessed by the https protocol requires a digital certificate then the user is asked to specify the file containing a certificate in the PKCS format for accessing that server.

The proxy settings are first looked up in the options. If there were no previous options set then the settings are loaded from the "servers" file located in the "%HOME%\Application Data\Subversion\" folder on Windows and "%HOME%\subversion\" folder on Linux and Mac OS X.

Messages

The Messages preferences panel is opened from menu Options → Preferences+Messages

Figure 5.10. The Messages preferences panel

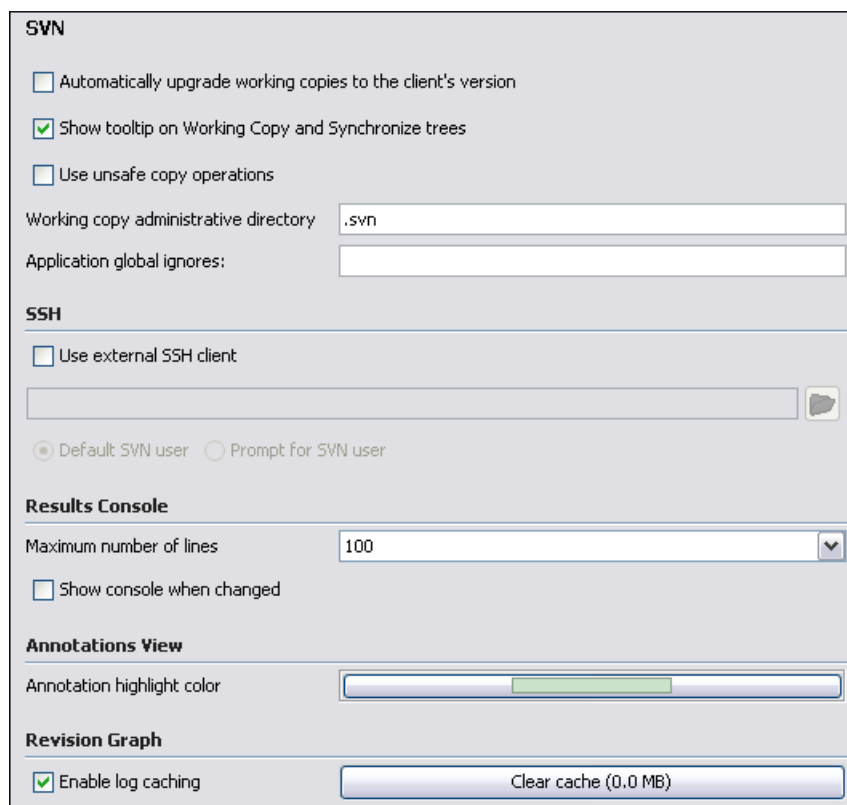


This page allows disabling warning messages which may appear in the application.

Show warning dialog when editing conflicts	If checked, when the <i>Edit Conflicts</i> action is executed in the SVN Client, a dialog will be shown that warns you that the action will overwrite the conflicted version of the file created by an update operation. The conflicted file will be overwritten with the version of the same file which existed in the working copy before the update operation and then proceeds with the visual editing of the conflict file. If the button Cancel is pressed in this warning dialog the <i>Edit Conflicts</i> action is aborted.
Show confirmation dialog for drag and drop actions in Working Copy	Set this option to avoid doing a drag and drop when you just want to select multiple files in the Working Copy view and run the same action on all selected files, for example a Copy or a Commit.

SVN

The SVN preferences panel is opened from menu Options → Preferences+SVN and it is the place where the user preferences for the embedded SVN client tool are configured. More preferences that configure how the embedded SVN client tool works can be set in the global files called 'config' and 'servers', that is the files with parameters that act as defaults applied to all the SVN client tools that are used by the same user on his login account on the computer. These files can be opened for editing with the two edit actions available in the SVN client tool on the *Global Runtime Configuration* submenu of the *Options* menu.

Figure 5.11. The SVN preferences panel

- Automatically upgrade working copies to the client's version - if this option is checked and a working copy stored locally on disk using an old SVN format (for example the SVN 1.3 one) is loaded in the Working Copy view then the working copy is automatically converted by Syncro SVN Client to the most recent SVN format. If the option is not checked a confirmation dialog will be displayed when such a working copy is loaded in the Working Copy view.
- Enable symbolic link support (*available only on Mac OS X and Linux*) - Subversion has the ability to put a symbolic link under version control, via the usual SVN add command. The Subversion repository has no internal concept of a symbolic link, it stores a "versioned symbolic link" as an ordinary file with a 'svn:special' property attached. The SVN client (on Unix) sees the property and translates the file into a symbolic link in the working copy.

Note

Win32 has no symbolic links, so a Win32 client won't do any such translation: the object appears as a normal file.

If the symbolic link support is disabled then the versioned symbolic links, on Linux and OS X, are supported in the same way as on Windows - i.e. a text file instead of symbolic link is created.

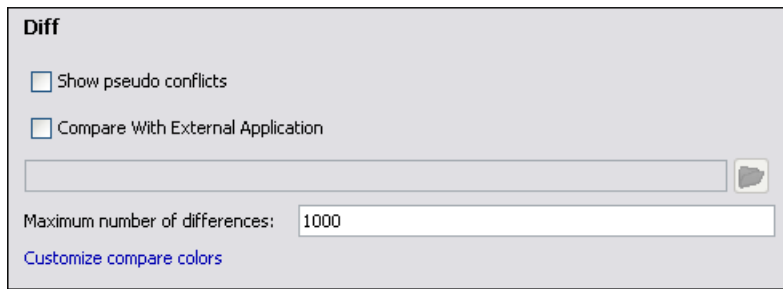
Important

We recommend to disable symbolic links support if you do not have versioned symbolic links in your repository, because the SVN operations will work faster. However, you should not disable this option when you do have versioned symbolic links in repository. In that case a workaround would be to refer to working copy by its "real" path, not path that includes a symbolic link.

- Show tooltip on Working Copy and Synchronize trees - For each file and folder a tooltip is displayed with details like SVN status, full path, current revision number, last changed date, etc. If the tooltips seem annoying by covering useful information they can be disabled with this option.
- Use unsafe copy operations - Sometimes when the working copy is accessed through Samba and SVN client cannot make a safe copy of the committed file due to a delay in getting write permission the result is that the committed file will be saved with zero length (the content is removed) and an error will be reported. In this case this option should be selected so that SVN client does not try to make the safe copy.
- Working copy administrative directory - allows you to customize the directory name where the svn entries are kept for each directory in the working copy.
- Application global ignores - allows setting file patterns that may include the wildcard * and ? for unversioned files and folders that must be ignored when displaying the working copy resources in the Working Copy view.
- SSH - here you can specify the command line for an external SSH client which will be used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments(if any). Depending on the SSH client used and your SSH server configuration you may need to specify in the command line the username and/or privatekey/passphrase. Here you can also choose if the default SVN user will be used(the same as the SSH client user) or you should be prompted for a user whenever SVN authentication is required.
- Compare - allows you to specify if you want to see *pseudo-conflicts* in the Compare view. You can also change the maximum number of differences allowed in the view and to specify an external application to be launched for compare operations in the following cases: when two history revisions are compared, when the working copy file is compared with a history revision, when a conflict is edited. The parameters `${firstFile}` and `${secondFile}` specify the positions of the two compared files in the command line for the external diff application. There is also available a link to quickly customize the used compare colors.
- Results Console - here you can specify the maximum number of lines displayed in the *Console View* and if the Console view should come to the foreground when there is some output that is displayed in this view.
- Annotations View - here you can set the color used for highlighting in the editor panel all the changes contributed to a resource by the revision selected in the Annotations view.
- Revision Graph - here you can enable caching for the action of computing a revision graph. When a new revision graph is requested one of the caches from the previous actions may be used which will avoid running the whole query again on the SVN server which will finish the action much faster.

SVN Diff

The SVN Diff preferences panel is opened from menu Options → Preferences+Diff and it allows you set the compare options for SVN.

Figure 5.12. The SVN Diff preferences panel

- Show pseudo conflicts - it allows you to specify if you want to see *pseudo-conflicts* in the Compare view. A pseudo conflict occurs when two developers make the same change, for example when both add or remove the same line of code.
- Compare With External Application - you can specify an external application to be launched for compare operations in the following cases: when two history revisions are compared, when the working copy file is compared with a history revision or when a conflict is edited. The parameters `${firstFile}` and `${secondFile}` specify the positions of the two compared files in the command line for the external diff application.
- Maximum number of differences - you can change the maximum number of differences allowed in the view.

Reset Global Options

To reset all custom user settings of the application that are stored in a local file (not in the project), to the installation defaults go to: Options → Reset Global Options

Chapter 6. Common problems

- 6.1. I cannot connect to a SVN repository from the Repository Browser view of SVN client. How can I find more data about the error? 111
- 6.1. I cannot connect to a SVN repository from the Repository Browser view of SVN client. How can I find more data about the error?

First check that you entered the correct URL of the repository in the Repository Browser view. Also check that a SVN server is running on the server machine specified in the repository URL and is accepting connections from SVN clients. You can check that the SVN server accepts connections with the command line SVN client from CollabNet.

If you try to access the repository with a svn+ssh URL also check that a SSH server is running on port 22 on the server machine specified in the URL.

If the above conditions are checked and you cannot connect to the SVN repository please generate a logging file on your computer and send the logging file to <support@syncrosvnclient.com>. For generating a logging file you need to create a text file called log4j.properties in the install directory with the following content:

```
log4j.rootCategory= debug, R2

log4j.appender.R2=org.apache.log4j.RollingFileAppender
log4j.appender.R2.File=logging.log
log4j.appender.R2.MaxFileSize=12000KB
log4j.appender.R2.MaxBackupIndex=20
log4j.appender.R2.layout=org.apache.log4j.PatternLayout
log4j.appender.R2.layout.ConversionPattern=%r %p [ %t ] %c - %m%n
```

Restart the application, reproduce the error, close the application and send the file logging.log generated in the install directory to <support@syncrosvnclient.com>.

Index

C

- Common problems, 111
- Configure the application, 99
 - Diff Appearance, 102
 - encoding, 101
 - fonts, 101
 - global, 99
 - import/export global options, 99
 - menu shortcut keys, 103
 - messages, 107
 - reset global options, 110
 - SVN, 107
 - SVN Diff, 109
 - SVN file editors, 104

E

- edit
 - copy/paste, 95
 - find/replace, 95
 - find/replace (keyboard shortcuts), 97

F

- find/replace, 95
 - keyboard shortcuts, 97

I

- Installation
 - All Platforms version, 4
 - Linux, 4
 - Mac OS X, 3
 - multiple instances (Unix/Linux server), 5
 - requirements, 2
 - unattended (Windows and Linux only), 5
 - Windows, 3

L

- License
 - register a license key, 7
 - registration code, 8
 - unregister license key, 8

P

- Performance problems
 - large documents, 10
 - problems on Linux/Solaris, 11

S

- Startup parameter, 6
- SVN Branches / Tags, 48
 - create a Branch / Tag, 48
 - create patches, 56
 - from repository revision, 60
 - from working copy, 57
 - merging, 49
 - merge options, 53
 - merge revisions, 50
 - merge two different trees, 52
 - reintegrate a branch, 52
 - resolve merge conflicts, 55
 - relocate a working copy, 56
 - switch the repository location, 56
- SVN Client, 12
 - Annotations view, 82
 - command line interface, 89
 - add, 90
 - add / edit property, 93
 - add to svn:ignore, 91
 - branch / tag, 92
 - checkout, 89
 - cleanup, 91
 - commit, 89
 - copy, 91
 - delete, 91
 - diff, 89
 - export, 90
 - import, 90
 - information, 90
 - lock, 92
 - mark as merged, 93
 - mark resolved, 91
 - merge, 92
 - move / rename, 91
 - override and commit, 93
 - override and update, 93
 - refresh, 90
 - remove property, 93
 - revert, 91
 - revert changes from these revisions, 93
 - revert changes from this revision, 93
 - scan for locks, 92
 - show / refresh properties, 92
 - show history, 90
 - synchronize, 90
 - update, 89
 - Compare view
 - Compare images view, 77
 - description, 76
 - Toolbar, 76
 - Console view, 85

- define a repository location, 18
 - Add/Edit/Remove repository locations, 18
 - authentication, 19
 - define a working copy, 21
 - check out, 21
 - use an existing working copy, 24
 - editor, 78
 - Help view, 85
 - History view
 - description, 79
 - history actions, 81
 - history filter dialog, 80
 - history filter field, 81
 - image preview, 79
 - main window, 13
 - main menu, 14
 - starting Syncro SVN Client, 13
 - views, 14
 - obtain information for a resource
 - request history, 43
 - request status information, 42
 - Properties view
 - description, 83
 - toolbar and contextual menu, 85
 - Repository view
 - contextual menu actions, 64
 - general description, 64
 - Toolbar, 64
 - Resource History view, 44
 - Directory Change Set view, 45
 - history actions available on double selection, 45
 - history actions available on single selection, 44
 - Revision Graph, 85
 - sparse checkouts, 63
 - SVN Branches / Tags, 48
 - SVN properties, 46
 - Add / Edit / Remove, 47
 - SVN working copy resources, 24
 - Synchronize view
 - contextual menu actions, 74
 - general description, 73
 - icons, 75
 - synchronize trees, 73
 - Toolbar, 73
 - Synchronize with the SVN repository, 29
 - Working Copy view
 - contextual menu actions, 68
 - general description, 66
 - icons, 72
 - Toolbar, 67
 - working with repositories
 - Copy / Move / Delete resources, 62
 - import / export resources, 62
 - SVN working copy resources
 - add resources to version control, 25
 - Copy / Move / Rename resources, 26
 - delete resources, 26
 - edit files, 24
 - ignore resources, 25
 - lock / unlock resources, 27
 - locked items, 28
 - locking a file, 28
 - scanning for locks, 28
 - unlocking a file, 29
 - Synchronize with the SVN repository, 29
 - commit changes, 40
 - integration with Bug Tracking tools, 41
 - presentation modes, 30
 - resolve conflicts, 33
 - content conflicts vs property conflicts, 34
 - drop incoming modifications, 37
 - edit real content conflicts, 34
 - merge conflicted resources, 37
 - real conflicts vs mergeable conflicts, 33
 - revert changes, 36
 - update the working copy, 39
 - view differences, 32
- ## T
- Text editor specific actions, 95
 - drag and drop, 98
 - exit the application, 98
 - undo and redo, 95
 - VI editor actions, 97
- ## U
- Uninstalling the application, 10
 - Upgrade, 9
 - check for new version, 9