Syncro SVN Client 9.1

Notice

Copyright

Syncro SVN Client User Manual

Syncro Soft SRL.

Copyright © 2002-2013 Syncro Soft SRL. All Rights Reserved.

Contact Syncro Soft SRL. Syncro Soft SRL provides a telephone number and e-mail address for you to report problems or to ask questions about your product, see the *Oxygen XML Web site*.

Contents

Chapter

1

Introduction

Welcome to the User Manual of Syncro SVN Client 9.1!

Syncro SVN Client is a cross-platform application designed 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.

This user guide is focused mainly at describing features, functionality and application interface to help you get started in no time. It is assumed that you are familiar with the basic concepts of a version control system.

Chapter

2

Installation

Topics:

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 Syncro SVN Client.

If you need help at any point during these procedures, please send us an email at support@syncrosvnclient.com

Installation Requirements

This section presents the details about the platform and environment requirements necessary to install and run Syncro SVN Client.

Platform Requirements

The run-time requirements of Syncro SVN Client are the following:

- CPU (processor): minimum - Intel Pentium III™/AMD Athlon™ class processor, 500 *Mhz*; recommended - Dual Core class processor.
- Computer memory: minimum - 512 MB of RAM (1 GB on Windows Vista and Windows 7); recommended - 512 MB of RAM (2 GB on Windows Vista and Windows 7).
- Hard disk space: minimum - 100 MB free disk space; recommended - 200 MB free disk space.

Operating System

The operating system requirements of Syncro SVN Client are the following:

Windows	Windows XP, Windows Vista, Windows 7, Windows 2003, Windows Server 2008, Windows Server 2012.
Mac OS	Mac OS X version 10.5 64-bit or later.
Unix/Linux	Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.6.0 or later from Oracle (formerly from Sun).

Environment Requirements

This section specifies the Java platform requirements and other tools that may be needed to install Syncro SVN Client.

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 at *the Download page* for every archive. You should check the MD5 sum of the downloaded archive with an MD5 checking tool available on your platform.

Java Virtual Machine Prerequisites

Prior to installation, ensure that your operating system environment complies with the following:

- Syncro SVN Client supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.6.0) from Oracle, formerly Sun Microsystems (available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*) and from Apple Computer. The Java Virtual Machine from Apple is pre-installed on Mac OS X computers. For Mac OS X, Java Virtual Machine updates are available at the Apple website. 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*.

- The PATH environment variable is set to the most current Java Virtual Machine installation.

- References to older Java Virtual Machine installations are removed from the PATH.

Installation Instructions

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

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.

> **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 before 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.

Windows Installation

Windows installation procedure.

The following steps describe the installation procedure of Syncro SVN Client, valid for Windows:

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 `com.syncrosvnclient` subfolder of the folder that is the value of the `APPDATA` Windows variable for the user that starts the application.

   > **Note:**
   >
   > To specify a different Java Virtual Machine for Syncro SVN Client to use, set the home folder of the desired Java Virtual Machine in one of the `JAVA_HOME` or `JDK_HOME` Windows variables. In case `JAVA_HOME` and `JDK_HOME` are not set, the Syncro SVN Client launcher looks for a JVM installed in a standard location on your computer and uses it to start the application. If you installed the kit that includes a Java Virtual Machine, rename or remove the `jre` subdirectory of the installation directory for the `JAVA_HOME` or `JDK_HOME` variables to take effect.

3. Copy to clipboard the license key you have received by email and paste it in the Syncro SVN Client license dialog.

   > **Note:**
   >
   > If you have difficulties installing the product, use the (syncroSVNClient.zip) archive distribution. Just unzip it in a directory where you have write permissions and use the product launchers.

Mac OS X Installation

The following steps describe the installation procedure of Syncro SVN Client, valid for Mac OS X:

1. Download the Mac OS X Installation package (`syncroSVNClient.zip`).

   The compressed file should be recognized and expanded by the web browser (Safari). If it is not automatically expanded, you can expand it manually by double clicking it.

2. In **Finder**, move the expanded folder to your `Applications` folder.
3. To run Syncro SVN Client, double click `SyncroSVNClient.app`

> **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 1.6.0 or newer. To change the version of the Java Virtual Machine that runs the Syncro SVN Client, go to Applications -> Utilities -> Java -> Java Preferences and drag the new JVM version up in the list of preferred versions.

4. Copy to clipboard the license key you have received by email and paste it in the Syncro SVN Client license registration dialog.

> **Note:**
>
> To avoid an unpredictable behavior of the cursor, use a fixed-size font, for example Monaco. You can find more details about this behaviour *here*.

Linux Installation

Linux installation procedure.

The following steps describe the installation procedure of Syncro SVN Client, valid for Linux:

1. Download the `syncroSVNClient-32bit.sh` installation kit for 32 bit Linux machines or the `syncroSVNClient-64bit.sh` installation kit for 64 bit Linux machines and run it.

2. Follow the instructions presented in the installation program.

> **Note:**
>
> To specify a different Java Virtual Machine for Syncro SVN Client to use, set the home folder of the desired Java Virtual Machine in the `JAVA_HOME` or `JDK_HOME` environment variables. If `JAVA_HOME` and `JDK_HOME` are not set, the Syncro SVN Client launcher looks for a Java Virtual Machine installed in a standard location on your computer and uses it to run the application.

All Platforms Installation

The following steps describe the installation procedure of Syncro SVN Client, valid for all supported platforms:

1. Create a folder named `syncroSVNClient` on your local disk.

2. In the `syncroSVNClient` folder, create a child folder named after the version number of Syncro SVN Client. The directory structure is /../syncroSVNClient/9.1/ .

3. Download the All Platforms Installation package (`syncroSVNClient.tar.gz`) to this folder.

4. Extract the archive to the same folder.

5. From a command line, run the following scripts: `syncroSVNClient.bat` on Windows, `syncroSVNClientMac.sh` on Mac OS X, and `syncroSVNClient.sh` on Unix / Linux.

> **Note:**
>
> To change the version of the Java Virtual Machine that runs Syncro SVN Client, go to the Java command at the end of the script file and specify the full path to the Java executable of the desired JVM version, for example:
>
> ```
> "C:\Program Files\Java\jre1.6.0\bin\java" -Xmx256m
> -Dsun.java2d.noddraw=true ...
> ```
>
> on Windows, or

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java
 "-Xdock:name=SyncroSVNClient" ...
```

on Mac OS X.

> ⚠ **Important:**
>
> A Java virtual machine version 1.6.0 or later has to be installed.

6. Copy to clipboard the license key you have received by email and paste it in the application license dialog.

Unix / Linux Server Configuration

Unix / Linux Server configuration procedure.

To install Syncro SVN Client on a Unix / Linux server, follow these steps:

1. Install Syncro SVN Client on the server and make 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. If you need to run multiple instances of the Syncro SVN Client, make sure you add the *-Dcom.oxygenxml.MultipleInstances=true* parameter in the startup script.

3. Make sure you allocate sufficient memory to Syncro SVN Client by setting an appropriate value for the `-Xmx` parameter in the `.sh` startup script.

4. Make sure the X server processes located on the workstations allow connections from the server host. For this, use the **xhost** command.

5. Telnet (or ssh) on the server host.

6. Start an xterm process, with the **display** parameter set on the current workstation. For example: `xterm -display workstationip:0.0`.

7. Start Syncro SVN Client by typing `syncroSVNClient.sh`.

Unattended Installation

Unattended installation is possible only on Windows and Linux by running the installer executable from the command line and passing the -q parameter. The installer executable is `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 Syncro SVN Client is installed in the same folder. The -overwrite parameter added after the -q parameter forces the overwriting of these files.

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

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

Custom Settings in Unattended Installation

An unattended installation applies the default settings of the installer.

If you want to install Syncro SVN Client on a large number of computers but you need to change the default values of some settings (like the installation directory, whether a desktop icon or a quick launch shortcut is created, the file associations created in the operating system, the name of the program group on the Start menu, etc.), use a special settings file which specifies the new values for these settings. To generate the settings file, 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 9.1\.install4j on Windows. This is a one time process. After that, to apply these options on all the computers where an unattended installation is performed, specify this file in the command line. For example copy the file in the same location as the installer program and use the following command:

- on Windows:

```
syncroSVNClient.exe -q -varfile response.varfile
```

- on Linux:

```
syncroSVNClient.sh -q -varfile response.varfile
```

Privacy Options

The following parameters control the privacy options:

Table 1: Privacy Options Parameters

Parameter name	Description	Values
autoVersionChecking	Automatic version checking.	true / false. Default setting is true.
reportProblem	Allows you to report a problem encountered while using Syncro SVN Client.	true / false. Default setting is true.
usageDataCollector	When set to true, you can choose whether Syncro SVN Client sends anonymous information about how it is used.	true / false. Default setting is true.

They can be set either in the response.varfile configuration file, or directly in the command line, like in the following example:

- on Windows:

```
syncroSVNClient.exe -q -VautoVersionChecking=false
-VreportProblem=false -VusageDataCollector=false
```

- on Linux:

```
syncroSVNClient.sh -q -VautoVersionChecking=false
-VreportProblem=false -VusageDataCollector=false
```

Making Syncro SVN Client Portable

To create a portable distribution of Syncro SVN Client that you can keep on an USB stick, follow these instruction:

- For Windows:

  - go to *www.oxygenxml.com/InstData/Editor/Windows/VM/oxygen.zip* and save the ZIP archive on your disk;

    📝 **Note:** The ZIP includes Java Virtual Machine.

  - unpack the ZIP and copy the license.txt file to the unpacked directory;
  - use /../syncroSVNClient/9.1/ .exe to start the application;

- For Mac OS X:

  - go to *http://www.oxygenxml.com/download_oxygenxml_editor.html?os=MacOSX* and download the MAC distribution;

    📝 **Note:** The MAC distribution does not include Java Virtual Machine.

  - unpack the ZIP and copy the license.txt file to the unpacked directory;
  - use Oxygen.app to start the application. Java will be installed automatically in case it is not installed already.

- For Linux and Ubuntu:

  - Download and install a Linux kit on one computer and copy the installation directory to the USB/network drive to distribute it the everyone:

    - go to *http://www.oxygenxml.com/download_oxygenxml_editor.html?os=Linux* and download the Linux kit that corresponds to your platform architecture (32 bit/ 64 bit);

      📝 **Note:** The ZIP includes Java Virtual Machine.

    - install it as a normal user (no root or `sudo`) in the user home and copy the `license.txt` file to the installation directory;
    - to start the application, use sh /../syncroSVNClient/9.1/ . Java is not needed on the system.

      📝 **Note:** The `.desktop` files will not work because the installation path is hard coded.

  - Download and unpack the all platforms archive(tar.gz), download Java for Linux and place it in the Oxygen directory:

    - go to *http://www.oxygenxml.com/download_oxygenxml_editor.html?os=All* and download the all platforms kit;

      📝 **Note:** Java Virtual Machine is not included.

    - go to *http://www.oracle.com/technetwork/java/javase/downloads/index.html*, download and unpack/install Linux JRE;
    - place the JRE in the `Oxygen` home directory, in a new directory named `jre`. The java executable should be `Oxygen\jre\bin\java`;
    - copy the `license.txt` file to the `Oxygen` directory;
    - use sh oxygen.sh to start the application.

      📝 **Note:** Java is not needed on the system. You have it bundled it with Syncro SVN Client.

Setting a Java Virtual Machine Parameter in the Launcher Configuration File / Start-up Script

There are two ways you can set new Java Virtual Machine parameters:

- *Setting parameters for the Syncro SVN Client launchers;*
- *Setting parameters in the command line scripts.*

Setting Parameters for the Application Launchers

Increasing the amount of memory that Syncro SVN Client uses on Windows

To increase the memory available to Syncro SVN Client on Windows:

- navigate to the installation directory of Syncro SVN Client;
- locate the -Xmx parameter in the `syncroSVNClient.vmoptions` file;

  📝 **Note:** For 32-bit Windows modify the parameter to -Xmx1024m or larger, but not over -Xmx1200m. Make sure you do not exceed your physical RAM. For 64-bit Windows modify the parameter to a larger value (for example -Xmx2048m). We recommended you to not use more than half of your existing physical RAM.

Restart Syncro SVN Client. Go to **Help** > **About** and verify the amount of memory that is actually available (see the last row in the **About** dialog). In case Syncro SVN Client does not start and you receive and error message saying that it could not start the JVM, decrees the -Xmx parameter and try again.

For Windows Vista/7, copy the syncroSVNClient.vmoptions to your desktop (or to any other folder with write access), modify it there, then copy it back to the Syncro SVN Client installation folder.

> **Note:** The parameters from the .vmoptions file are used when you start Syncro SVN Client with the *oxygen* launcher (or with the desktop shortcut). In case you use the command line script oxygen.bat/oxygen.sh, modify the -Xmx parameter in the script file.

Increasing the amount of memory that Syncro SVN Client uses on Mac OS X

To increase the memory available to Syncro SVN Client on Mac OS X:

* **Ctrl (Meta on Mac OS) + Click** (or right click) the Syncro SVN Client icon in **Finder**;
* from the pop-up menu, select **Show Package Contents**;
* navigate to the Contents directory and open for editing the Info.plist file;

> **Note:** You can open this file either with the **Property List Editor**, or the **TextEdit**.

* Look for the **VMOptions** key and adjust the -Xmx parameter to a larger value (for example -Xmx1500m)

> **Note:** Try not to use more than half of your existing physical RAM if possible.

Setting a system property

To set a system property, you have to provide a parameter of the following form:

```
-Dproperty.name=value
```

Setting Parameters in the Command Line Scripts

If you start Syncro SVN Client with the syncroSVNClient.bat script, you have to add or modify the parameter to the java command at the end of the script.

For example, to set the maximum amount of Java memory to 600 MB on **Windows**, modify the **-Xmx** parameter like this:

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

on **Mac OS X** the java command should look like:

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

and on **Linux** the Java command should look like:

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

Starting Syncro SVN Client

This section specifies the steps for starting Syncro SVN Client on Windows, Mac OS X, and Linux and also for starting it using the all platforms kit.

Starting Syncro SVN Client on Windows

Start the Syncro SVN Client launcher.

Use one of the following two launchers:

- Syncro SVN Client - started from the shortcut created by the installer on the **Start** menu.
- syncroSVNClient.bat - located in the install folder and started from command line.

Starting Syncro SVN Client on Mac OS X

Start the Syncro SVN Client launcher.

Use one of the following two methods:

- The syncroSVNClient shortcut that the installer created on your **Desktop**.
- The sh syncroSVNClientMac.sh command, executed from the command line. This launcher file is located in the installation folder.

⚠ **Notice:** Two or more instances can be started on the same computer with the following command that should be executed for any new instance:

```
open -n SyncroSVNClient.app
```

Starting Syncro SVN Client on Linux

Start the Syncro SVN Client launcher.

Use one of the following two methods:

- The syncroSVNClient shortcut created on **Desktop** by the installer.
- The sh syncroSVNClient.sh command executed from the command line. This launcher file is located in the installation folder.

Starting Syncro SVN Client with the All Platforms Kit

Start the Syncro SVN Client launcher.

Use the following command:

- syncroSVNClient.bat on Windows;
- sh syncroSVNClient.sh on Linux;
- sh syncroSVNClientMac.sh on Mac OS X.

Obtaining and Registering a License Key

Syncro SVN Client is not free software. To enable and use Syncro SVN Client, you need a license.

For demonstration and evaluation purposes, a time limited license is available upon request at the *Syncro SVN Client website*. This license is supplied at no cost for a period of 30 days from the date of issue. During this period, the software is fully functional, enabling you to test all its aspects. Thereafter, the software is disabled. To further use it, you must purchase a permanent license. For special circumstances, if a trial period greater than 30 days is required, please contact support@syncrosvnClient.com .

For definitions and legal details of the license types, consult the End User License Agreement received with the license key. It is also available at *http://www.syncrosvnClient.com/eula.html*.

There are several ways to register a license, depending on its type and use case:

• using the **Register** dialog of Syncro SVN Client (the most common case) to register a named-user based or concurrent license;
• storing the license key into a file (either text or xml) that is copied to the installation directory of Syncro SVN Client.

At start-up, Syncro SVN Client looks for a valid license key in the following locations, in this order:

• *in the xml file registration*;
• *in the text file registration*;
• in its internal settings files created after you used the **Register** dialog to validate a *named-user based*.

Named User License Registration

1. Save a backup copy of the message containing the new license key.
2. Start Syncro SVN Client.
3. Copy to the clipboard the license text as explained in the message.
4. If this is a new install of Syncro SVN Client, the registration dialog is displayed automatically at start-up. In the case you already used Syncro SVN Client and obtained a new license, use the menu option Help / Register
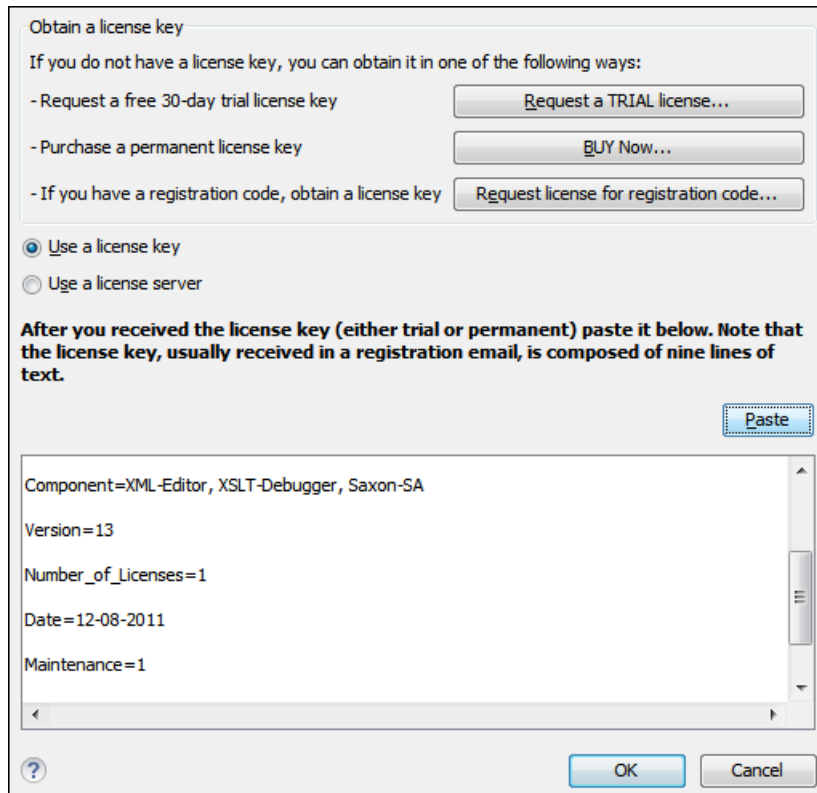
Figure 1: Registration Dialog

5. Paste the license text in the registration dialog.
6. Press the OK button.

Named User License Registration with Text File

1. Save the license key in a file named `licensekey.txt`.
2. Copy `licensekey.txt.` in the installation folder of Syncro SVN Client or in the `lib` subfolder of the installation folder.
3. Start the Syncro SVN Client.

Named User License Registration with XML File

This procedure is designed to help system administrators register the application for multiple users, without the hassle of configuring the application licensing for each of them.

1. Register Syncro SVN Client for the currently logged user, using the *Named User License Registration* procedure.
2. Copy the `license.xml` file from the *preferences directory* of Syncro SVN Client to its installation directory (or its `lib` sub-directory).
3. For each installation of Syncro SVN Client (either on the same computer or on different computers) repeat step 2.

License Registration with an Activation Code

If you have only an activation 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 may need to unregister your license key. For example, to transfer your license key to other computer before other user starts using your current computer.

1. Go to menu **Register**
   This displays the license registration dialog.

2. Make sure the text area for the license key is empty.

3. Make sure the option **Use a license server** is not selected.

4. Press the **OK** button of the dialog.
   This displays a confirmation dialog.

5. Select between falling back to the license key entered previously (for the case of releasing a floating license and reverting to Named User license) and removing your license key from your user account of the computer using the **Reset** button.

   The **Reset** button erases all the licensing information. To complete the reset operation, close and restart Syncro SVN Client.

Upgrading Syncro SVN Client

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.

Any personal configuration settings and customizations are preserved by installing an upgrade or a patch.

Upgrading the Standalone Application

Upgrading to a new version might require a new license key. To check if your license key is compatible with the new version, select **Help** > **Check for New Version**. Note that the application needs an Internet connection to check the license compatibility.If there is a previous version of Syncro SVN Client already installed on your computer, it can coexist with the new one, which means you don't have to uninstall it. The user preferences are stored in a different directory. They will not be removed and will be imported automatically in the new version at the first application launch.

1. Download the install kit of the new version.

2. Create a new folder under `/../syncrosvnClient` e. g. /../syncrosvnClient/9.1

3. Extract the content of the install kit into the new folder.

4. If you have defined Syncro SVN Client in the system PATH, modify it to point to the new installation folder.

5. Start Syncro SVN Client
   This will test that the application can start and that your license is recognized by the upgrade installation.

Checking for New Versions

Syncro SVN Client offers *the option of checking for new versions* at the *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.

To check for new versions manually, at any time, go to **Help** > **Check for New Versions**.

You are informed automatically that a new version is available as follows:

- in case the version you are using is the same with the latest released version, you are notified when new builds are available at *www.syncrosvnclient.com*;
- in case the version you are using is the penultimate version (for example you are using version 7.1 and version 7.2 is available at *www.syncrosvnclient.com*), you are informed that a new version was released;
- in case at *www.syncrosvnclient.com* two versions are available, for example version 7.2 and version 8, and you are using version 7.2, you are informed that version 7.2 was released. Apart from this notification, if you do not have the latest build installed, you are also informed about new 7.2 builds;
- in case at *www.syncrosvnclient.com* both a minor and a major version is available, and you are using a version older than both of them, you are informed about the major version and also about the minor one.

Uninstalling the Syncro SVN Client

This section contains uninstallation procedures.

Uninstalling the Syncro SVN Client Standalone

⚠️ **Caution:** The following procedure will remove Syncro SVN Client from your system. **Please ensure that all valuable data stored in the install folder is saved to another location prior to performing this procedure.**

1. Backup all valuable data from the Syncro SVN Client installation folder.
2. Remove the application.

   - 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 want to remove also the user preferences that were configured in the **Preferences** dialog you must remove the folder `%APPDATA%\com.syncrosvnClient` on Windows (usually %APPDATA% has the value [user-home-dir]\Application Data) / the subfolder `.com.syncrosvnClient` of the user home folder on Linux / the subfolder `Library/Preferences/com.syncrosvnClient` of the user home folder on Mac OS X.

Unattended Uninstall

The unattended uninstall procedure is available only on Windows and Linux.

Run the uninstaller executable from command line with the -q parameter.

The uninstaller executable is called `uninstall.exe` on Windows and `uninstall` on Linux and is located in the application's install folder.

Performance Problems

This section contains the solutions for some common problems that may appear when running Syncro SVN Client.

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 *adding the startup parameter* -Dsun.java2d.pmoffscreen=false.

Chapter

3

Tools

Topics:

- *Syncro SVN Client*
- *Comparing and Merging Documents*

Syncro SVN Client ships with a set of tools oriented to XML related tasks. You gave access to a revision control system, a comparing and merging solution and also to other tools like a large file viewer and a hex viewer.

Syncro SVN Client

Syncro SVN is a client for the Apache Subversion^{TM} 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.

To start Syncro SVN Client, go to **Tools** > **SVN Client**.

Main Window

This section explains the main window of Syncro SVN Client.

Views

The main window consists of the following views:

- *Repositories view* - Allows you to define and manage Apache Subversion^{TM} repository locations.
- *Working Copy view* - Allows you to manage with ease the content of the working copy.
- *History view* - Displays information (author name, revision number, commit message) about the changes made to a resource during a specified period of time.
- *Editor view* - Allows you to edit different types of text files, with full syntax-highlight.
- *Annotations view* - Displays a list with information regarding the structure of a document (author and revision for each line of text).
- *Compare view* - Displays the differences between two revisions of a text file from the working copy.
- *Image Preview* - Allows you to preview standard image files supported by Syncro SVN Client: JPG, GIF and PNG.
- *Compare Images view* - Displays two images side by side.
- *Properties view* - Displays the SVN properties of a resource under version control.
- *Console view* - Displays information about the currently running operation, similar with the output of the Subversion command line client.
- *Help view* - Shows information about 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 menus:

- **File** menu:

  - **New** submenu:

    - **New File** - This operation creates a file in the working copy and adds it to version control. If the selected path is not under version control, the newly created file is added to the repository only by an explicit action. Creating a file in the working copy does not add it automatically to the repository. This action works only for selected paths in the **Working Copy** tree.
    - **New Folder (Ctrl (Meta on Mac OS) + Shift + F)** - This operation creates a new folder as child of the selected folder from the *Repositories view* tree or from the *Working Copy view* tree, depending on which view was focused last when performing this action. For the *Working Copy view*, the folder is added to version control only if the selected path is under version control, otherwise the newly created directory is not added to version control.
    - **New External Folder (Ctrl (Meta on Mac OS) + Shift + W)** - 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 points and the revision number of that repository URL can be selected easily with the **Browse** and **History** buttons of the dialog. This action works only for selected paths in the *Working Copy* tree.

Subversion 1.5 clients 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.

- **Open (Ctrl (Meta on Mac OS) + O)** - This action opens the selected file in an editor where you can modify it. The action is active only when a single item is selected. The action opens a 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). Folder opening is available only for folders selected in the *Working Copy view*. This action works on any file selection from the *Repositories view*, *Working Copy view*, *History view* or *Directory Change Set view*, depending on which view was last focused when invoking it.

- **Open with(Ctrl (Meta on Mac OS) + Shift + O)** - Displays the *Open with* dialog for specifying the editor in which the selected file is opened. In case multiple files are selected only external applications can be used to open the files. This action works on any file selection from *Repositories view*, *Working Copy view*, *History view* or *Directory Change Set view*, depending on which view was last focused when invoking it.

- **Show in Explorer**/**Show in Finder** - Opens the parent directory of the selected working copy file and selects the file.

- **Save (Ctrl (Meta on Mac OS) + S)** - Saves the local file currently opened in the editor or the **Compare** view.

- **Save as** - Saves locally any file selected in the **Repositories**, **History**, and **Directory Change Sets** view.

- **Copy URL Location (Ctrl (Meta on Mac OS) + Alt + U)** - Copies to clipboard the URL location of the resource currently selected in the **Repositories** view.

- **Copy to** - Copies to a specified location the currently selected resource(s) either in **Repositories** or **Working copy** view.

  > **Note:** This action can also be used from **History** and **Directory Change Set** views to recover older versions of a repository item.

- **Move to (Ctrl (Meta on Mac OS) + M)** - Moves to a specified location the currently selected resource(s) either in **Repositories** or **Working copy** view.

- **Rename (F2)** - Renames the resource currently selected either in **Repositories** or **Working copy** view.

- **Delete (Delete)** - Deletes the resource currently selected either in **Repositories** or **Working copy** view.

- **Locking**:

  - **Scan for locks (Ctrl (Meta on Mac OS) + L)** - 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 *Scanning for locks*.

  - **Lock (Ctrl (Meta on Mac OS) + K)** - 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 see *Locking a file*.

  - **Unlock (Ctrl (Meta on Mac OS) + Alt + K)** - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

- 🏷 **Show SVN Properties (Ctrl (Meta on Mac OS) + Shift + P)** - Brings up the *Properties view* and displays the SVN properties for a selected resource from *Repositories view* or *Working Copy view*, depending on which view was last focused when invoking it.
- ℹ️ **File Information (Ctrl (Meta on Mac OS) + I)** - Provides additional information for a selected resource from the *Working Copy view*. For more details please see the section *Obtain information for a resource*.
- **Exit (Ctrl (Meta on Mac OS) + Q)** - Closes the application.

- **Edit** menu:

    - ↩ **Undo (Ctrl (Meta on Mac OS) + Z)** - Undo edit changes in the local file currently opened in the editor or the **Compare** view.
    - ↪ **Redo (Ctrl (Meta on Mac OS) + Y)** - Redo edit changes in the local file currently opened in the editor or the **Compare** view.
    - ✂ **Cut (Ctrl (Meta on Mac OS) + X)** - Cut selection to clipboard from the local file currently opened in the editor view or the **Compare** view.
    - 📋 **Copy (Ctrl (Meta on Mac OS) + C)** - Copy selection to clipboard from the local file currently opened in the editor or the **Compare** view.
    - 📋 **Paste (Ctrl (Meta on Mac OS) + V)** - Paste selection from clipboard in the local file currently opened in editor or the **Compare** view.
    - 🔍 **Find/Replace (Ctrl (Meta on Mac OS) + F)** - Perform find / replace operations in the local file currently opened in the editor or the **Compare** view.
    - 🔍 **Find Next (F3)** - 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 (Shift + F3)** - 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** menu:

    - 🗄 **New Repository Location (Ctrl (Meta on Mac OS) + Alt + N)** - Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.

    ![Add SVN Repository dialog box showing Repository URL field with http://svn.public-repository.com/svn/repos, a checked Validate repository connection checkbox, and OK and Cancel buttons.]
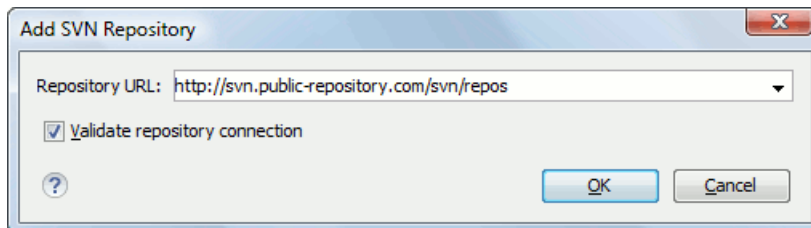
    Figure 2: Add SVN Repository Dialog Box

    If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

    - 🔍 **Edit Repository Location (Ctrl (Meta on Mac OS) + Alt + E)** - Context-dependent action that allows you to edit the selected repository location using the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.
    - **Change the Revision to Browse (Ctrl (Meta on Mac OS) + Alt + Shift + B)** - Context-dependent action that allows you to change the selected repository revision using the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.
    - ✕ **Remove Repository Location (Ctrl (Meta on Mac OS) + Alt + Shift + R)** - 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.
    - 🔄 **Refresh** - Refreshes the resource selected in the **Repositories** view.

- **Check out** (**Ctrl (Meta on Mac OS) + Alt + Shift + C**) - Allows you to create a working copy from a repository directory, on your local file system. To read more about this operation, see the section *Check out a working copy*.

- **Export** - Exports a folder from the repository to the local file system.

- **Import** sub-menu:

  - **Import Folder Content** (**Ctrl (Meta on Mac OS) + Alt + Shift + M**) - Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.

    📑 **Note:** The difference between **Import Folder Content** and **Share project** actions is that the latter one also converts the selected directory into a working copy.

  - **Import File(s)** (**Ctrl (Meta on Mac OS) + Alt + I**) - Imports the files selected from the files system into the selected folder from the repository.

- **Working Copy** menu:

  - 📁 (📁 on Mac OS X) **Working Copies Manager** - Opens dialog with a list of working copies that the Apache Subversion[TM] client is aware of. In this dialog you can add existing working copies or remove no longer needed ones.

  - **Switch to** - Selects one of the following view modes: ⊟ **All Files**, ⟷ **Modified**, ⟵ **Incoming**, ⟶ **Outgoing**, or ⟷ **Conflicts**.

  - 🔄 **Refresh** (**F5**) - Refreshes the state of the selected resources or of the entire working copy if there is no selection.

  - 🔄 **Synchronize** (**Ctrl (Meta on Mac OS) + Shift + S**) - Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.

  - **Update** (**Ctrl (Meta on Mac OS) + U**) - Updates all the selected resources that have incoming changes to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.

  - **Update to revision/depth** - 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.

  - **Commit** - Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.

  - 📂 **Update all** (**Ctrl (Meta on Mac OS) + Shift + U**) - Updates all resources from the working copy that have incoming changes. It performs a recursive update on the synchronized resources.

  - 📤 **Commit all** - Commits all the resources with outgoing changes. It is disabled when **Incoming** mode is selected or the synchronization result does not contain resources with outgoing changes. It performs a recursive commit on the synchronized resources.

  - ↩ **Revert** (**Ctrl (Meta on Mac OS) + Shift + V**) - Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Apache Subversion[TM] pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.

  - **Edit conflict** (**Ctrl (Meta on Mac OS) + E**) - Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see *Edit conflicts*.

  - ✔ **Mark Resolved** (**Ctrl (Meta on Mac OS) + Shift + R**) - Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.

- ✔ **Mark as Merged (Ctrl (Meta on Mac OS) + Shift + M)** - Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.
- **Override and Update** - Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.
- **Override and Commit** - Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.
- **Mark as copied** - You can use this action to mark an item from the working copy as a copy of an other item under version control, when the copy operation was performed outside of an SVN client. The **Mark as copied** action is available when you select two items (both the new one and the source one) and depends on the state of the source item.

  > **Note:** If you use an SVN 1.6 working copy or older, this action does not apply for directories.

- **Mark as moved** - You can use this action to mark an item from the working copy as being moved from another location of the working copy, when the move operation was performed outside of an SVN client. The **Mark as moved** action is available when you select two items from different locations (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

  > **Note:** If you use an SVN 1.6 working copy or older, this action does not apply for directories.

- **Mark as renamed** - You can use this action to mark an item from the working copy as being renamed outside of an SVN client. The **Mark as renamed** action is available when you select two items from the same directory (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

  > **Note:** If you use an SVN 1.6 working copy or older, this action does not apply for directories.

- **Add to "svn:ignore" (Ctrl (Meta on Mac OS) + Alt + I)** - Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the *svn:ignore* property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.
- ➕ **Add to version control (Ctrl (Meta on Mac OS) + Alt + V)** - Allows you to schedule for addition resources that are not under version control. For further details, see *Add Resources to Version Control* section.
- **Remove from version control** - Schedules selected items for deletion from repository upon the next commit. The items are not removed from the file system after committing.
- 🧹 **Clean up (Ctrl (Meta on Mac OS) + Shift + C)** - 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.
- ⊞ **Expand all (Ctrl (Meta on Mac OS) + Alt + X)** - Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.
- ⊟ **Collapse all (Ctrl (Meta on Mac OS) + Alt + Z)** - Collapses all descendants of the selected folder. The same behavior is obtained by double-clicking on a expanded folder.

- **Compare** menu:

  - 🔲 **Perform Files Differencing** - performs a comparison between the source and target files;

  - ⬇ **Next Block of Changes** - jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

  - ⬆ **Previous Block of Changes** - jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

- ⬇ **Next Change** - jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change;

- ⬆ **Previous Change** - jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change;

- ⬇ **Last Change** - jumps to the last change from the current file;

- ⬆ **First Change** - jumps to the first change from the current file;

- **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 (Ctrl (Meta on Mac OS) + Shift + Comma)** - This action copies the selected change from the right editor to the left editor.

- **Show Word Level Details** - provides a word-level comparison of the selected change;

- **Show Character Level Details** - provides a character-level comparison of the selected change.

- **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.

- **History** menu:

  - 🕐 **Show History (Ctrl (Meta on Mac OS) + H)** - Displays the history for a SVN resource at a given revision. The resource can be one selected from the **Repositories** view, **Working Copy** view, or from the **Affected Paths** table from the **History** view, depending on which view was last focused when this action was invoked.

  - **Show Annotation** (**Ctrl (Meta on Mac OS) + Shift + A**) - Complex action that does the following operations:

    - opens the selected resource in the **Annotations** editor;
    - displays corresponding annotations list in the **Annotations** view;
    - displays the history of the selected resource.

  - This operation is available for any resource selected from **Repositories** view, **Working Copy** view, **History** view or **Directory Change Sets** view, depending on which view was last focused when this action was invoked.

  - **Revision Graph (Ctrl (Meta on Mac OS) + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

- **Tools** menu:

  - **Share project** - Allows you to *share a new project* using an SVN repository. The local project is automatically converted into an SVN working copy.
  - **Branch / Tag** - Allows you to copy the selected resource from the **Repositories** view or **Working Copy** view to a branch or tag into the repository. To read more about this operation, see the section *Creating a Branch / Tag*.
  - **Merge (Ctrl (Meta on Mac OS) + J)** - Allows you to merge the changes made on one branch back into the trunk, or vice versa, using the selected resource from the working copy. To read more about this operation, see the section *Merging*.
  - **Switch (Ctrl (Meta on Mac OS) + Alt + W)** - Allows you to change the repository location of a working copy or only of a versioned item of the working copy within the same repository. It is available when the selected item of the working copy is a versioned resource, except an external folder. To read more about this action, see the section *Switching the Repository Location*.
  - **Relocate** - Allows you to change the base URL of the root folder of the working copy to a new URL, when the base URL of the repository changed, for example the repository itself was relocated to a different server. This

operation is available for a selected item of the working copy tree that is a versioned folder. To read more about this operation, see the section *Relocate a Working Copy*.

- ✖ **Create patch (Ctrl (Meta on Mac OS) + Alt + P)** - Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see *the section about patches*.

- **Working copy format** - this submenu contains the following two operations:

  - 🔁 **Upgrade** - Allows you to upgrade the format of the current working copy to the newest one known by Syncro SVN Client, to allow you to benefit of all the new features of the client.

  - 🔁 **Downgrade** - Allows you to downgrade the format of the current working copy to an older format. The formats allowed to downgrade to are SVN 1.5 and SVN 1.4. This is useful in case you wish to use older SVN clients with the current working copy, or, by mistake, you have upgraded the format of an older working copy by using a newer SVN client.

    📄 **Note:** SVN 1.7 working copies cannot be downgraded to older formats.

    See the section *Working Copy Format* to read more about this subject.

- **Options** menu:

  - **Preferences** - Opens the **Preferences** dialog.
  - **Menu Shortcut Keys** - Opens the **Preferences** dialog directly on the **Menu Shortcut Keys** option page, where users can configure in one place the keyboard shortcuts available for menu items available in Syncro SVN Client.
  - **Global Run-Time Configuration** - Allows you to configure SVN general options, that should be used by all the SVN clients you may use:

    - **Edit 'config' file** - In this file you can configure various SVN client-side behaviors.
    - **Edit 'servers' file** - In this file you can configure various server-specific protocol parameters, including HTTP proxy information and HTTP timeout settings.

  - **Export Options** - Allows you to export the current options to a file.
  - **Import Options** - Allows you to import options you have previously exported.
  - **Reset Options** - Resets all your options to the default ones.
  - **Reset Authentication** - Resets the Subversion authentication information.

- **Window** menu:

  - **Show View** - Allows you to select the view you want to bring to front.
  - **Show Toolbar** - Allows you to select the toolbar you want to be visible.
  - **Enable flexible layout** - Toggles between a fixed and a flexible layout. When the flexible layout is enabled, you can move and dock the internal views to adapt the application to different viewing conditions and personal requirements.
  - **Reset Layout** - Resets all the views to their default position.

- **Help** menu:

  - **Help (F1)** - Opens the **Help** dialog.
  - **Dynamic Help** - Shows the **Dynamic Help** view.
  - **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.
  - **Improvement Program Options** - Allows you to activate or deactivate the Syncro Soft Product Improvement Program.
  - **Report Problem** - Opens a dialog that allows the user to write the description of a problem that was encountered while using the application.
  - **Support Center** - Opens the Support Center web page in a browser.
  - **About** - Opens the **About** dialog.

Main Toolbar

The toolbar of the SVN Client SVN Repositories window contains the following actions:

*   **Check out** - Checks out a working copy from a repository. The repository URL and the working copy format must be specified.

*   **Synchronize** - Synchronizes the current working copy with the repository.

*   **Update All** - Updates all resources of the working copy that have an older revision that repository.

*   **Commit All** - Commits all resources of working copy that have a newer version compared to that of the repository.

*   **Refresh** - Refreshes the whole content of the current working copy from disk starting from the root folder. At the end of the operation, the modified files and folders that were not committed to repository yet, are displayed in the **Working Copy** view.

*   **Compare** - The selected resource is compared with:

    *   the *BASE* revision, when the selected resource is:

        *   locally modified and the **All Files** view mode is currently selected (no matter if there are incoming changes);
        *   locally modified and there are no incoming changes when any other view mode is selected.

    *   the remote version of the same resource, when remote information is available after a **Synchronize** operation (only when one of **Modified**, **Incoming**, **Outgoing** and **Conflicts** view modes is selected).
    *   the working copy revision, when the selected resource is from the **History** view;

*   **Show History** - Displays the history of the selected resource (from the **Working Copy** or **Repository** views) in the **History** view.

*   **Show Annotation** - Displays the annotations of the selected resource. The selected resource can be in the **Working Copy** or the **History** views.

*   **Revision Graph** - Displays the revision graph of the selected resource. The selected resource can be in the **Working Copy** or the **Repositories** views.

*   **Enable/Disable flexible layout** - Toggles between a fixed and a flexible layout. When the flexible layout is enabled, you can move and dock the internal views to adapt the application to different viewing conditions and personal requirements.

Status Bar

The status bar of the Syncro SVN Client window displays important details of the current status of the application. This information is available only in the **Working Copy** view.
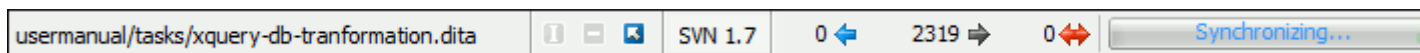
Figure 3: Status bar

The status bar is composed of the following areas:

*   the path of the currently processed file from the current working copy (during an operation like **Check out** or **Synchronize**) or the result of the last operation;
*   the current status of the following working copy options:

- Show ignored files (   );
- Show deleted files (   );
- Process `svn:externals` definitions (   ).

The options for ignored and deleted files are switched on and off from *the Settings menu* of the **Working Copy** panel;
- the format of the currently loaded working copy;
- the current numbers of incoming changes (   ), outgoing changes (   ) and conflicting changes (   );
- a progress bar for the currently running SVN operation and a button (   ) that allows you to stop it.

Getting Started

This section explains the basic operations that can be done in Syncro SVN Client.

SVN Repository Location

This section explains how to add and edit the repository locations in Syncro SVN Client.

Add / Edit / Remove Repository Locations

Usually, team members do all of their work separately, in their own working copy, and must share their work. This is done using an Apache Subversion^{TM} repository. Syncro SVN Client supports versions 1.4, 1.5, 1.6, 1.7 and 1.8 of the SVN repository format.

 Note: SVN 1.8 FSFS-backed repositories (accessed through `file://` protocol) are not supported.

Before you can begin working with a Subversion repository, define a repository location in the *Repositories view*.

To create a 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 on a right click with the mouse or with the keyboard by pressing the special context menu key available on Windows keyboards. This action opens the **Add SVN Repository** dialog which prompts 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 prompts you for a new password only when your password changes.

Once you enter the repository URL Syncro SVN Client tries to contact the server and get the content of the repository for displaying it in the *Repositories view*. If the server does not respond in the timeout interval set in preferences, an error is reported. If you do not want to wait until the timeout expires, you can end the waiting process with the   **Stop** button from the toolbar of the view.

To edit a repository location, click the **Edit Repository Location** toolbar button or right click inside the view on a repository root 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 shows the previously defined repositories URLs and it allows 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 is displayed to make sure that you do not accidentally remove locations.

The order of the repositories can be changed in the **Repositories** 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 up the selected repository in the list with one position.

To set the reference revision number of an SVN repository right-click on the repository in the list displayed in the *Repositories view* and select the **Change the Revision to Browse...** action. The revision number of the repository is used for displaying the contents of the repository when it is viewed in the *Repositories 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 for all the file open operations executed directly from the *Repositories view*.

Authentication

Five protocols are supported: *HTTP*, *HTTPS*, *SVN*, *SVN + SSH* and *FILE*. If the repository that you are trying to access is password protected, the **Enter authentication data** dialog requests a user name and a password. If the **Store authentication data** checkbox is checked, the credentials are stored in Apache Subversion™ default directory:

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

There is 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 causes Subversion to forget all your credentials. When you reset the authentication data, restart Syncro SVN Client for the change to take effect.

> **Tip:** The *FILE* protocol is recommended if the SVN repository and Syncro SVN Client are located on the same computer as it ensures faster access to the SVN repository compared with other protocols.

For HTTPS connections where client authentication is required by your SSL server, you must choose the certificate file and enter the corresponding certificate password which is used to protect your certificate.

When using a secure HTTP (HTTPS) protocol for accessing a repository, a **Certificate Information** dialog pops up and asks you whether you accept the certificate permanently, temporarily or simply deny it.

If the repository has SVN+SSH protocol, the SSH authentication can also be made with a private key and a pass phrase.
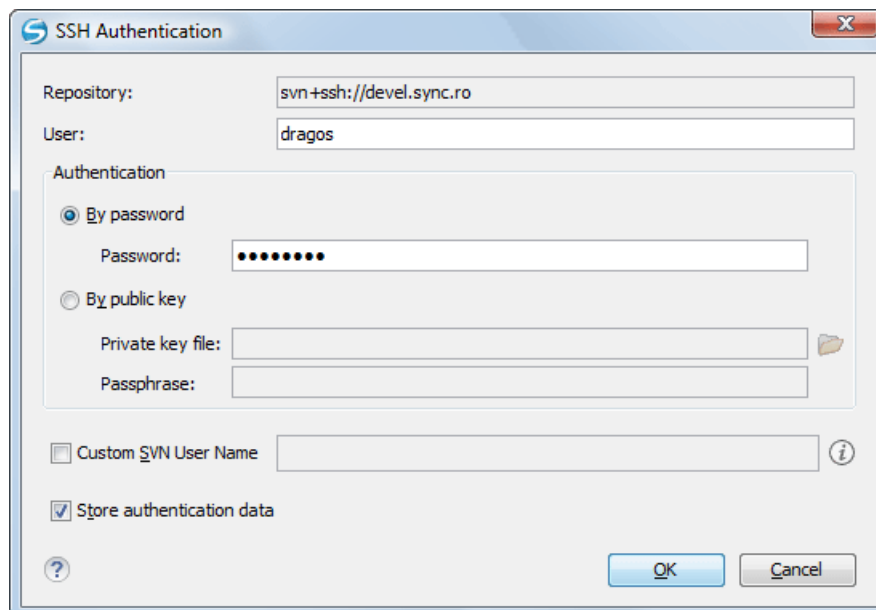
Figure 4: User & Private key authentication dialog

After the SSH authentication dialog, another dialog pops up for entering the SVN user name that accesses the SVN repository. The SVN user name is recorded as the *committer* in SVN operations.

When connecting for the first time to a Subversion repository through SVN+SSH protocol, you will be asked to confirm if you trust the SSH host. The same dialog box is also displayed when the server changed the SSH key or when the key was deleted from the local Subversion cache folder.
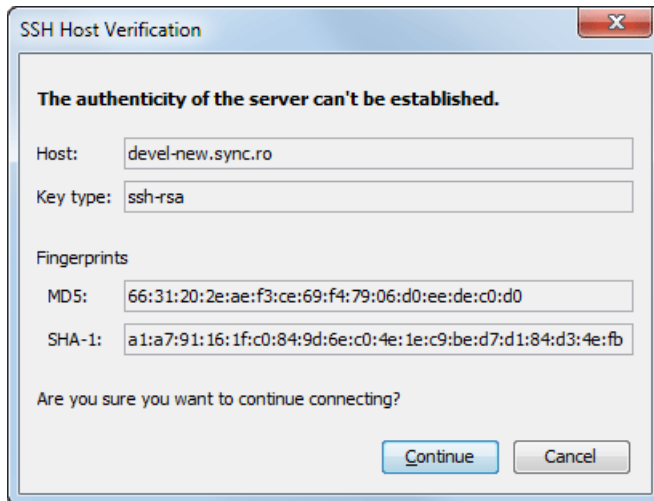
Figure 5: SSH server name and key fingerprint

Share a Project

Even if you start developing a new project or you want to migrate an existing one to Subversion, Syncro SVN Client allows you to share it easily with the rest of your team. The **Share project** action helps you do this by providing a simplified interface:
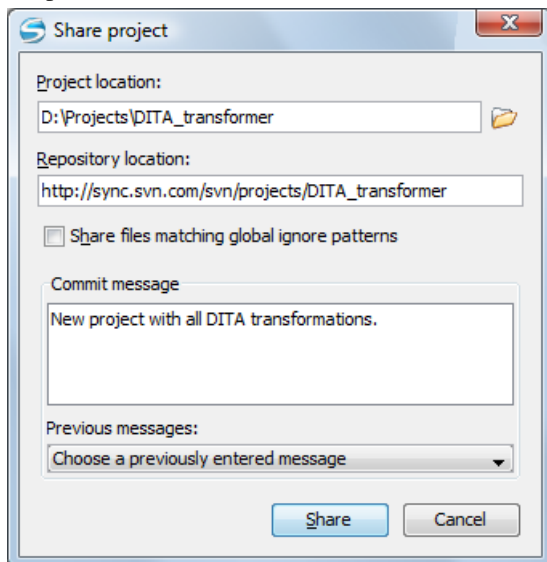
Figure 6: Share Project Dialog Box
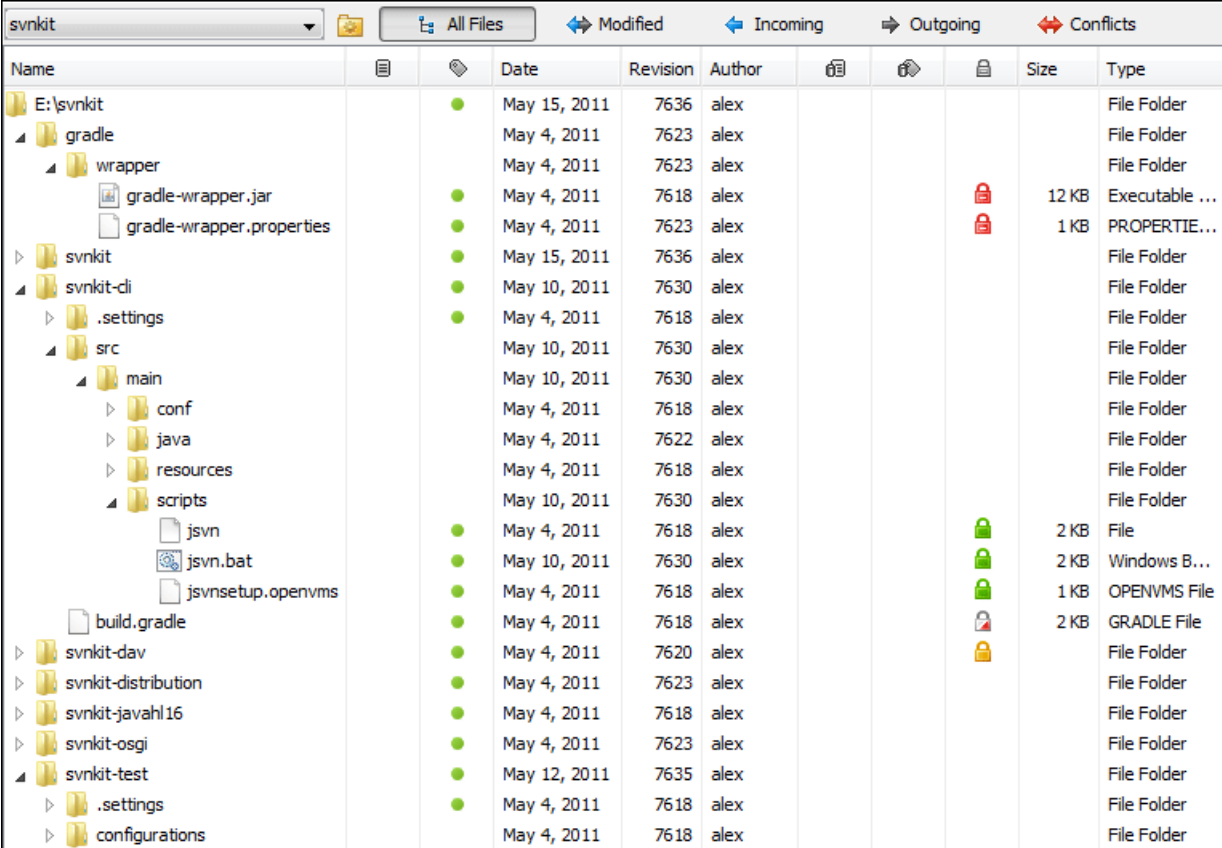
To share a project, set the following:

- **Project location** box - enter the location of the project on the local disk.

  📄 **Note:** The root directory of the project is not imported in the repository, only its content is imported.

- **Repository location** box - type the repository address of the new project. If the address does not exist, it is created automatically. If it does exist, make sure that it is an empty location to avoid mixing your project content with other files.
- **Share files matching global ignore patterns** check box - file names that match the patterns defined under the `global-ignores` property, in the Subversion clients configuration file, are also imported in the repository.

The shared project directory is automatically converted to a working copy and added under Syncro SVN Client management.

Defining a Working Copy

An Apache Subversion^{TM} 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 7: 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 usually corresponds to a particular subtree of the repository.

Check Out a Working Copy

Check Out means making a copy of a project from a repository into your local file system. This copy is called a *working copy*. An Apache Subversion^{TM} working copy is a specially formatted directory structure which contains additional `.svn` directories that store Subversion information, as well as a pristine copy of each item that is checked out.

To check out a working copy, go to the *Repositories view*. In case no connections to your repository are available, *add a new repository location* and follow these steps to check out a working copy:

1. Navigate to the desired repository directory in the **Repositories** view.
2. Right click on the directory and select **Check Out...** from the popup menu.

The **Check Out** dialog box is displayed:
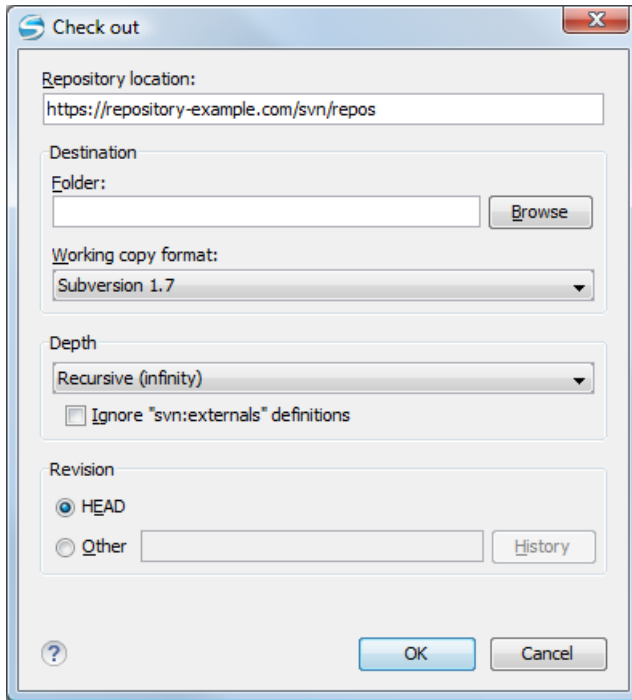
Figure 8: Check Out Dialog

3. Click the **Browse** button.
4. Select the location where the working copy is created.
5. Select the version of the working copy format: SVN 1.4, SVN 1.5, SVN 1.6 or SVN 1.7.
6. Select the depth to be used when checking out the working copy;

   This allows you to specify the recursion level into child resources. The depth is used if you want to check out only a part of the selected repository directory and bring in a future update the rest of the files and subdirectories. You can find out more about the check out depth in the *sparse checkouts* section.

7. Select the revision number that is checked out.

   By default the last (HEAD) revision is checked out. If you need another revision, you have to specify the revision number or click the **History** button which opens *the History dialog*.

After a check out, the new working copy will be added to the list in the ***Working Copy** view* and loaded automatically.

The History Dialog

The **History** dialog presents a list of revisions for a resource. It is opened from the dialogs that require setting an SVN revision number like *the Check Out dialog* or *the Branch / Tag dialog* to name just a few. It presents information about revision, commit date, author, and commit comment.
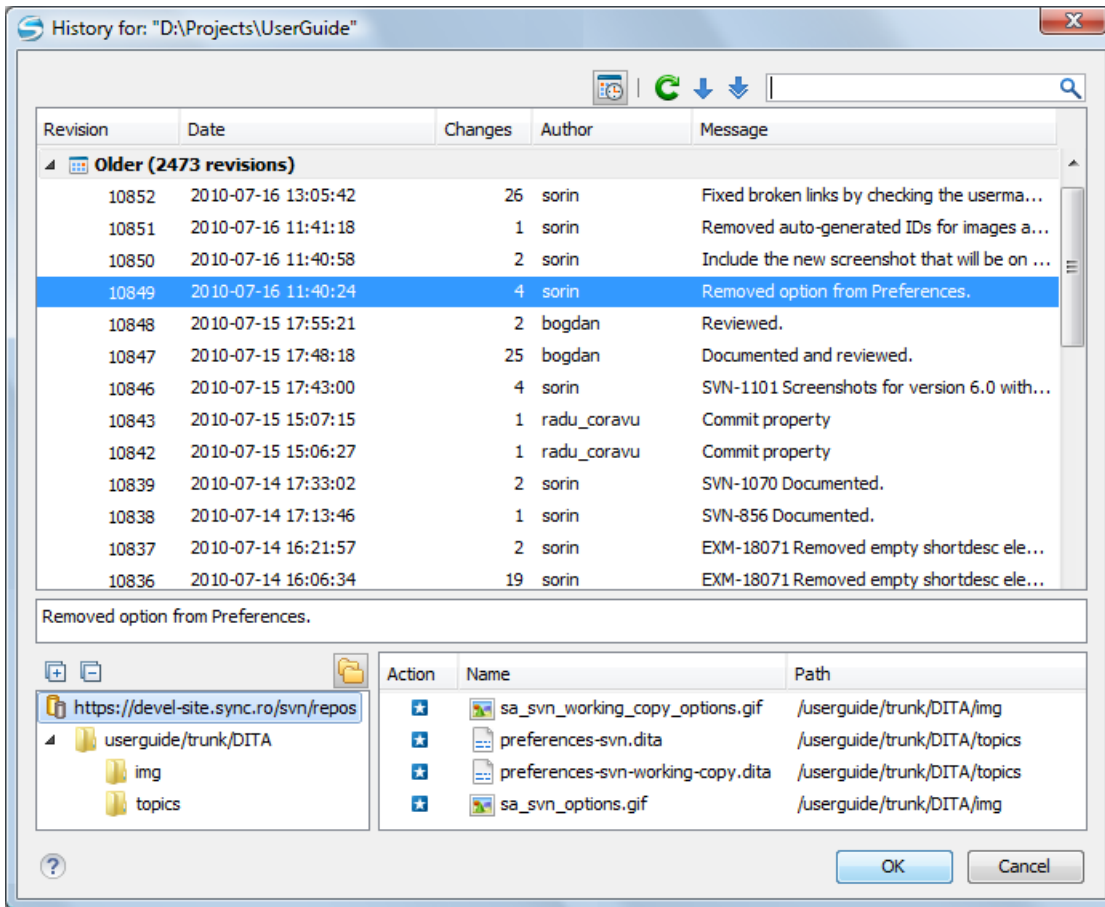
Figure 9: History Dialog

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. You can group revisions in predefined time frames (today, yesterday, this week, this month), by pressing the 🗓 **Group by date** button from the toolbar.

The **Affected Paths** area displays all paths affected by the commit of the revision selected in history. You can see the changes between the selected revision and the file's previous state using the **Compare with previous version** action, available in the contextual menu.

Use an Existing Working Copy

Using an existing working copy is the process of taking a working copy that exists on your file system and connecting it to Apache Subversion^TM repository. If you have a brand new project that you want to import into your repository, then see the section *Import resources into the repository*. The following procedure assumes that you have an existing valid working copy on your file system.

1. Click the **Working Copies Manager** toolbar button 📁 (📁 on Mac OS X) in the *Working Copy view*.
   This action opens the **Working copies list** dialog.
2. Press the **Add** button.
3. Select the working folder copy from the file system.

   📄 **Note:**

   With SVN 1.6 and earlier, you can use any directory located in a working copy as a standalone working copy. This helps you avoid loading a whole working copy when needing to work only on a small part of it.

Starting with SVN 1.7, all the internal information of a working copy is kept only in the root directory. To track outside changes properly, Syncro SVN Client needs to load the whole working copy.

4. Optionally you can press the **Edit** button to change the name of the working copy that is displayed in the **Working Copy** view.

   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.

   To change the working copies order, use the two arrow buttons which move the selected working copy one position up or down.

5. Press the **OK** button.

The selected working copy is loaded and presented in the *Working Copy view*.

Manage Working Copy Resources

This section explains how to work with the resources that are displayed in the **Working Copy** view.

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 contextual menu.

Please note that only one file can be edited at a time. If you try to open another file, it is opened in the same editor window. The editor has syntax highlighting for known file types, meaning that a different color is used for each type of recognized token in the file. If the selected file is an image, then it is previewed in the editor, with no access to modifying it.

After modifying and saving a file from a working copy, a modified marker - an asterisk (*) - will be added to the file's icon in the *Working Copy view*. The asterisk marks the files that have local modifications that were not committed to the repository.

Add Resources to Version Control

To share new files and folders (created in your working copy), add them to version control using the **Add to version control** option from the *Working Copy view*.

You can easily spot resources not under version control by the (*unversioned*) icon displayed in the   **Local file status** column. Resources scheduled for addition (*added*) are displayed with this icon in the **Working Copy** view and are added in the repository after you commit them.

> **Note:** Do not make a confusion between and icons. The former icon stands for resources that are actually copies of resources already committed in the repository, meaning they are *scheduled for addition with history*.

When you use the **Add to version control** option on a directory, its entire structure is scanned and all the resources that can be added under version control are presented.

Though it is not mandatory to add resources under version control explicitly, it is recommended. If you forgot to add a resource, when you *commit your changes*, the resource is presented in the commit dialog, but not selected. When you commit and *unversioned* resource, it is automatically added under version control before starting the commit operation.
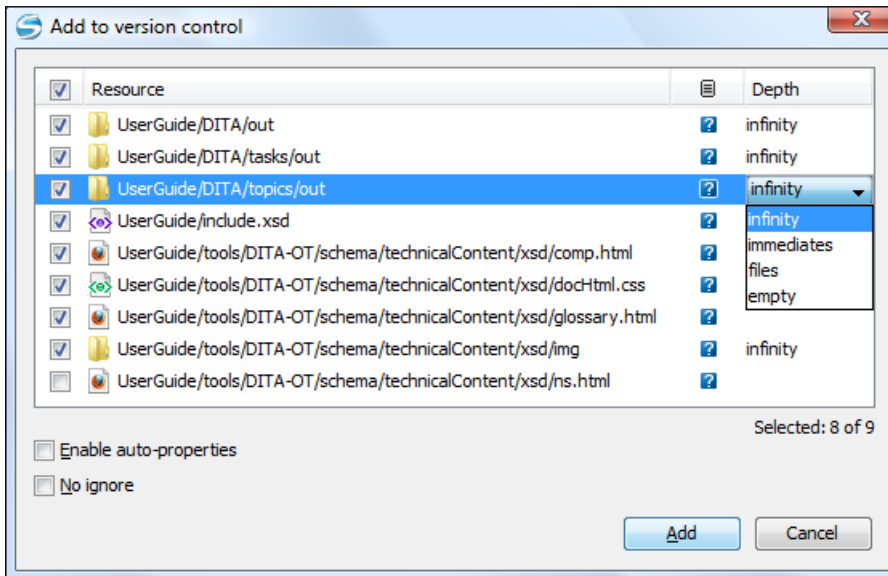
Figure 10: "Add to version control" dialog box

📄 **Note:** *Ignored* (🔲) items can also be added under version control.

The **Depth** column is displayed only when directories are also presented in the dialog. For any directory, you can use one of the available values to instruct Subversion to limit the scope of the operation to a particular tree depth.

📄 **Note:** The initial value of the **Depth** field can have the following values, depending on the *listing mode of the items in the working copy view*:

- *infinity* - when the working copy items are presented as a tree;
- *files* - when the working copy items are presented compressed;
- *empty* - when the working copy items are presented flat.

When you add unversioned or ignored directories, the initial value of the **Depth** field also depends on the state of the **Show unversioned directories content** and **Show ignored directories content** options. In case these options are enabled, the value is based on the listing mode of the items in the working copy view. When they are disabled, the value is *empty* .

The following options are available in this dialog box:

- **Enable auto-properties** or **Disable auto-properties** - enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in the `config` file of the Subversion configuration directory.

  📄 **Note:**

  This option is available only when there are defined properties to be applied automatically for resources newly added under version control. You can define these properties in the `config` file of the Subversion configuration directory, in the `auto-props` section.

  Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable auto-properties**, or **Disable auto-properties**.

- **No ignore** - when you enable this option, file-name patterns defined to ignore *unversioned* resources do not apply. Resources that are located inside an *unversioned* directory selected for addition, and match these patterns, are also scheduled for addition in the repository.

  📄 **Note:**

  This option is available only when directories are also presented in the dialog.

You can define file-name patterns to ignore *unversioned* resources in one of the following locations:

- in the `config` file of the Subversion configuration directory (the `global-ignores` option from the `miscellany` section);
- in the Syncro SVN Client options: **Options** > **Preferences** > **SVN** > **Working copy** > **Application global ignores**.

Each of the above two options is activated only when you select an item for which the option can be applied.

Ignore Resources Not Under Version Control

Some resources inside your working copy do not need to be subject to version control. These resources can be files created by the compiler, `*.obj`, `*.class`, `*.lst`, or output folders used to store temporary files. Whenever you *commit changes*, Apache Subversion^{TM} shows your modified files but also the unversioned files, which fill up the file list in the commit dialog. Though the unversioned files are 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 are never displayed in the commit dialog and only genuine unversioned files which must be committed are shown.

You can choose to ignore a resource by using the **Add to svn:ignore** action in the contextual menu of the ***Working Copy*** *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 the following wildcard characters:

- `*` - Matches any string of characters of any size, including the empty string.
- `?` - Matches any single character.

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

The action **Add to svn:ignore** 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 is visible in the **Working Copy** view. The ignored resources are represented with grayed icons.

Delete Resources

The ✖ **Delete** action is available in the contextual menu of the ***Working Copy view***. When you delete an item from the working copy, it is marked as *deleted* (scheduled for deletion from repository upon the next commit) and removed from the file system. Depending on the state of each item, you are prompted to confirm the operation.

> **Note:** For SVN 1.6 working copies, the delete action does not remove from the file system the directories under version control, it only marks them as *deleted*. This is because they also contain the pristine copy of that directory content. In the ***Working Copy view***, this behaviour is transparent as all resources have the deleted mark (■). The directories are removed from the file system when you *commit* them to the repository.

If a resource is deleted from the file system without Subversion's knowledge, the resource is marked as *missing* (■) in your working copy. You can decide what you want to do with a *missing* item:

- in case of a commit, any *missing* item is first deleted automatically and then committed.

  > **Note:** Not any *missing* item can be committed as *deleted*, and removed from the repository. For example, you cannot commit an item that no longer exists on the disk and that was scheduled for addition (■) previously, since this item does not exist in the repository, but you can use the **Delete** action instead.

- in case you want to recover *missing* items, either *update* the items themselves or one of their parent directories. This fetches their latest version from the repository.

You can also delete conflicting items (file content conflicts, property conflicts, tree-conflicts) and Syncro SVN Client automatically marks them as resolved.

> **Note:** It is recommended that you resolve conflicts manually to avoid loosing any important remote modifications.

Finally, you can change your mind and *revert* the deleted items to their initial, pristine, state.

Copy Resources

You can copy several resources from different locations of the working copy. You select them in the *Working Copy view* and then use **Copy to** from the contextual menu. This is not a simple file system copy, but an Apache 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.

Based on the selected items, the **Copy to** action is enabled only if it can be performed. Even if the operation would not normally be possible in SVN (due to some invalid local file states against copy), Syncro SVN Client performs the copy operation as a simple file system operation. This means no SVN versioning meta-data is affected.

> **Note:**
> - In case you copy an item to a directory that is *not under version control* (*unversioned* or *ignored*), the history of the item is not preserved. For example, when copying directories, all items inside them will also be copied without history.
> - In case you copy a directory that contains *external* items, these are not copied. This is specific for SVN 1.7 working copies only. To fetch the *external* items, use the **Update** operation on the copied directory.

In the **Copy to** dialog you can navigate through the working copy directories in order to choose a target directory, to copy inside it. If you try to copy a single resource you are also able to change that resource's name. For *versioned* items, you can select **Ignore resource history** to copy them without their history (similar to a simple file system copy).

> **Note:** The **Copy to** dialog only presents all the local directories that are a valid destination against the copy operation, based on their local file status. Also, the *working copy settings* are taken into account.

In 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 move several resources at once. Select the resources in the *Working Copy view* and choose the **Move to** action from the contextual 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*.

> **Note:** *External* items cannot be moved using the **Move to** action, because they cannot be deleted. Instead, you should edit the *svn:externals* property defining the *external* item.

Rename Resources

The **Rename** action is available in the contextual menu of the *Working Copy view* and can be performed on a single resource. This action acts as a move command with the destination directory being the same as the original location of the resource. A copy of the original item is created with the new name, also keeping its history. The original item is marked as *deleted*.

> **Note:** For SVN 1.6 working copies, renaming items by changing only letter cases is not permitted on case-insensitive platforms (like Windows and some Mac OS file systems).

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 modify 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.

An SVN 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 demands 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 is not 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.

Path	State	Owner	Comment	Creation date	Expiration Date
Sample/css/sample1.css	🔒	mihai	Modified the input style.	11/30/11 6:03 PM	
Sample/css/sample1.xhtml	🔒	mihai	Updated the W3C details	11/30/11 6:04 PM	
Sample/personal-schema.css	🔒	dragos	reverted	7/5/11 4:35 PM	
Sample/personal.dtd	🔒	marius	test busy cursor	6/22/11 11:35 AM	
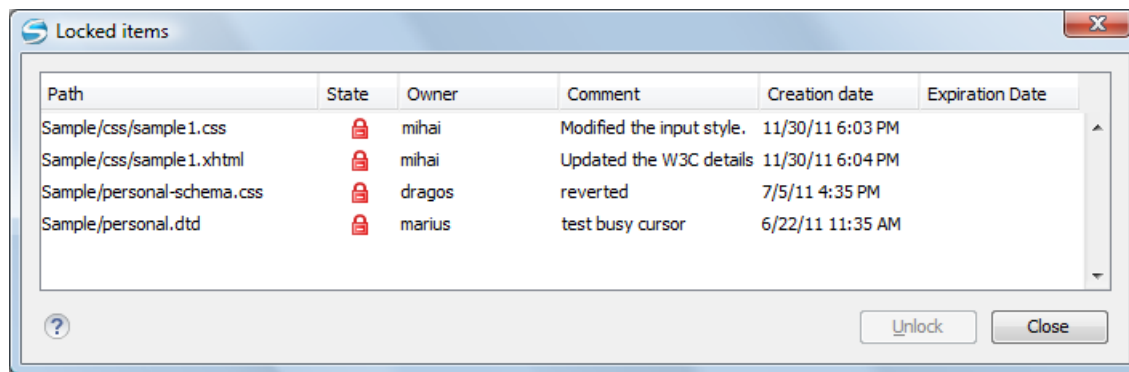
Figure 11: The locked items dialog

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:

- 🔒 - shown when:

  - another user has locked the file in the repository;
  - the file was locked by the same user from another working copy;
  - the file was locked from the **Repositories** view.

- 🔒 - displayed after you have locked a file from the current working copy.
- 🔓 - a file already locked from your working copy is no longer locked in the repository (it was unlocked by another user).
- 🔒 - a file already locked from your working copy is being locked by another user. Now the owner of the file lock is the user who stole the lock from you.

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

Locking a File

By locking a file you have exclusive write access over it in the repository.

You can lock a file from your working copy or directly from the **Repositories** view. Note that you can lock only files, but no directories. This is a restriction imposed by Apache Subversion[TM].

The **Lock** dialog allows you to write a comment when you set a lock or when you steal an existing one. Note that you should steal a lock only 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, as it may not allow you to do this if certain conditions are not met.

The lock stays in place until you unlock the file or until someone breaks it. There is also the possibility that the lock expires after a period of time specified in the Subversion server policy.

Unlocking a File

A file can be unlocked from the contextual menu of 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 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 **Incoming** and **Outgoing** modes of *the Working Copy view* was designed for, to help you send and receive modifications from the repository.

The **Incoming** and **Outgoing** modes of this view focus on incoming and outgoing changes. The incoming changes are the changes that other users have committed in the repository 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** action from *the Working Copy 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*.
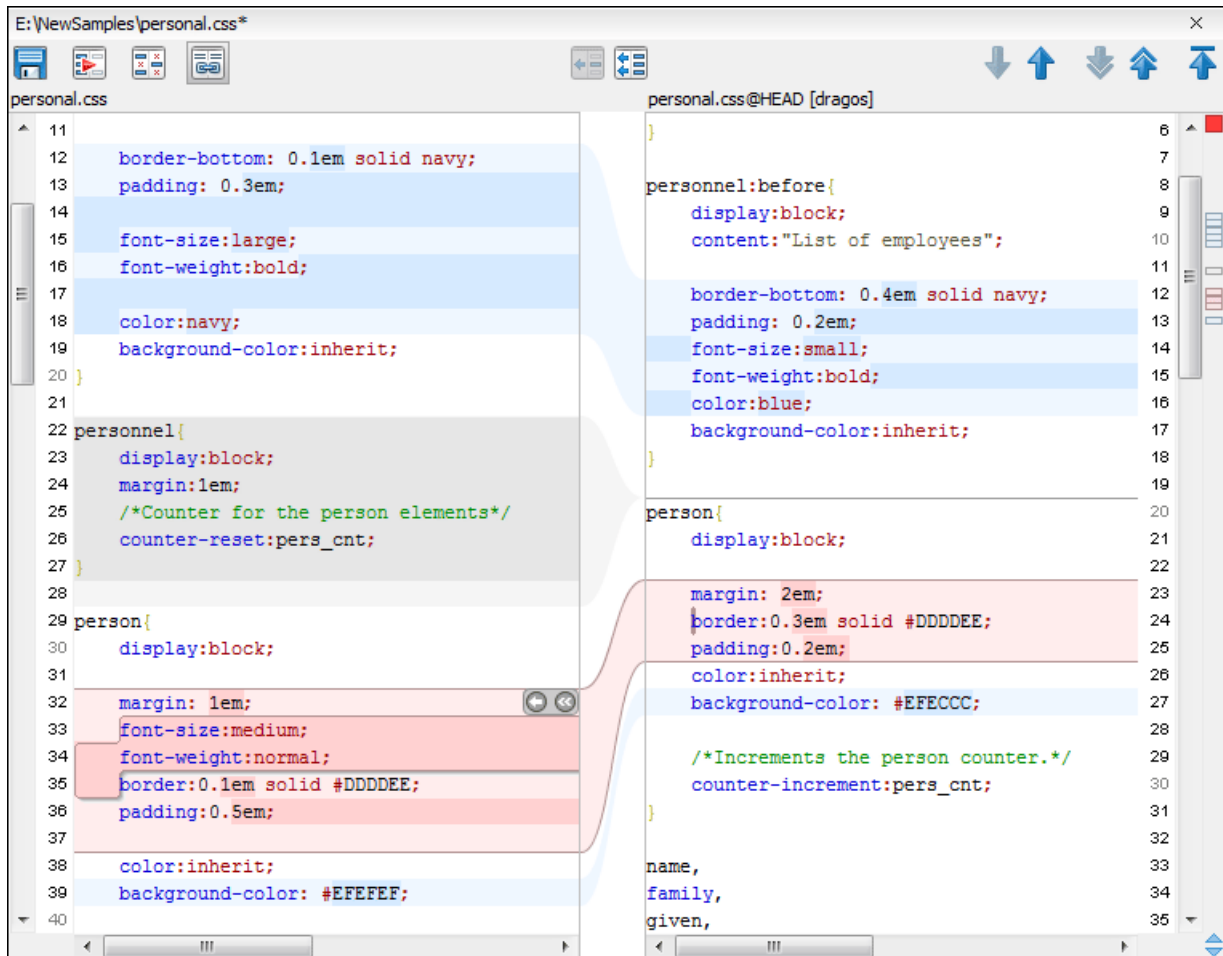
Figure 12: Compare View

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

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** and **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 action **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 when you double click on the middle divider of a difference: **Word Details** and **Character Details**.

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 in the *SVN Preferences* which allows you to enable or disable the white space ignoring feature of the compare algorithm.

Conflicts

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:

- *real conflict* (decorator in *Name* column) - Syncro SVN Client considers the following resource states to be real conflicts:

  - *conflicted* state - a file reported by SVN as being in this state is obtained after it was updated/merged while having incoming and outgoing content or property changes at the same time, changes which could not be merged. A content conflict (symbol in *Local file status* column) is reported when the modified file has binary content or it is a text file and both local and remote changes were found on the same line. A properties conflict (symbol in *Local properties status* column) is reported when a property's value was modified both locally and remotely;
  - *tree conflicted* state (symbol in *Local file status* column) - obtained after an update or merge operation, while having changes at the directory structure level (for example, file is locally modified and remotely deleted or locally scheduled for deletion and remotely modified);
  - *obstructed* state (symbol in *Local file status* column) - obtained after a resource was versioned as one kind of object (file, directory, symbolic link), but has been replaced outside Syncro SVN Client by a different kind of object.

- *pseudo-conflict* (decorator in *Name* column) - a file is considered to be in *pseudo-conflict* when it contains both incoming and outgoing changes. When incoming and outgoing changes do not intersect, an update operation may automatically merge the incoming file content into the existing locally one. In this case, the *pseudo-conflict* marker is removed. This marker is used only as a warning which should prevent you to run into a real conflict.

  Note:

  - A conflicting resource cannot be committed to repository. You have to resolve it first, by using **Mark Resolved** action (after manually editing/merging file contents) or by using **Mark as Merged** action (for pseudo-conflicts).
  - and decorators are presented only when one of the following view modes is selected: **Modified**, **Incoming**, **Outgoing**, **Conflicts**.
  - The marker is used also for folders to signal that they contain a file in real conflict or pseudo-conflict state.

Content Conflicts vs Property Conflicts

A *Content 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, Apache Subversion[TM] 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.

A *Property conflict* is 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 you 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*. Resolving the conflict means deciding 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.

The **Compare** view (or the external diff application *set in Preferences*) is opened with the action **Edit Conflict** which is available on the contextual menus of *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** - Saves the modifications of the local version of the file displayed in the left side of the view.
- **Perform Files Differencing** - Applies 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** - Scrolls the view to the topmost difference.
- **Go to Previous Modification** - Scrolls the view to the previous difference. The current difference is painted with a darker color than the other ones.
- **Go to Next Modification** - Scrolls the view to the next difference. The current difference is painted with a darker color than the other ones.
- **Go to Last Modification** - Scrolls 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** - Copies 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** - Applies the previous operation for all the differences.
- **Show Modification Details at Word Level** - Displays a more detailed version of the current difference computed at word level.

- **Show Modification Details at Char Level** - Displays 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 should run:

- either 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 contextual 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 the repository file. This can be achieved either by editing directly the file or by using the action **Copy change from right to left** from the *Compare view toolbar*. After editing the conflicting file you have to run the action **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.
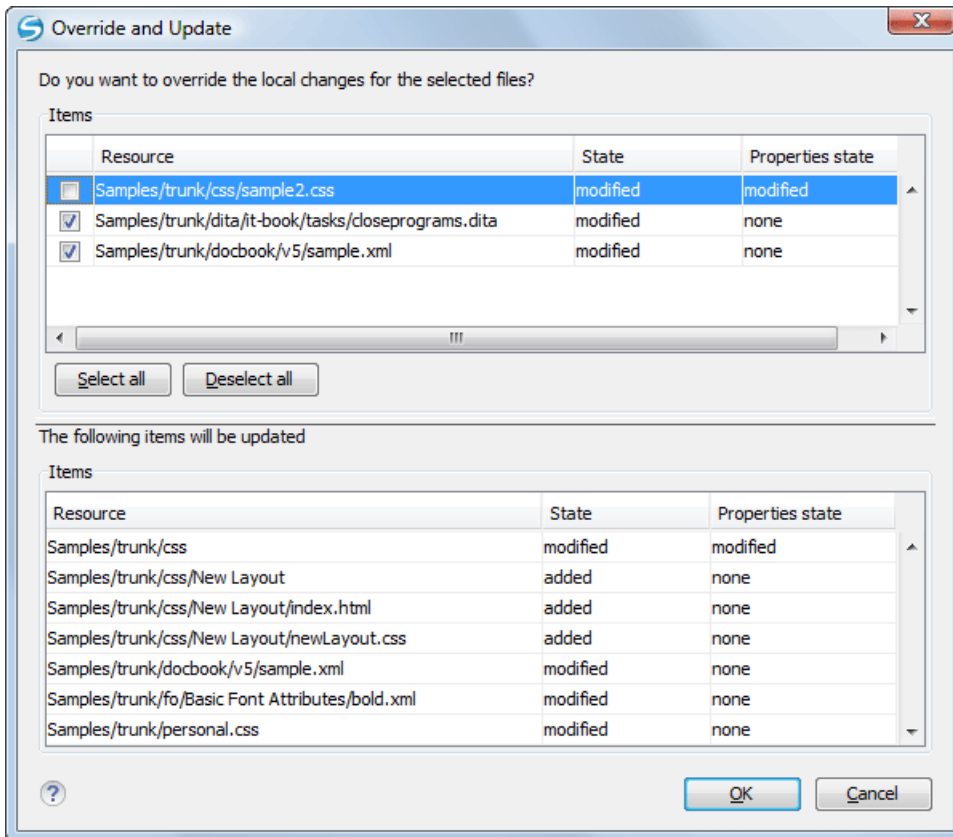
Figure 13: Override and update dialog

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 are 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 should open the file in the compare editor and decide which incoming changes need to be copied locally and 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 use the **Mark as merged** action. When you want to override completely the remote file with the local file you should run the action **Override and commit** which drops any remote changes and commits your file.

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

Tree Conflicts

A *tree conflict* is a conflict at the directory tree structure level and occurs when the user runs an update action on a resource that:

• it is locally modified and the same resource was deleted from the repository (or deleted as a result of being renamed or moved);
• it was locally deleted (or deleted as a result of being renamed or moved) and the same resource is incoming as modified from the repository.

The same conflict situation can occur after a merge or a switch action. The action ends with an error and the folder containing the file that is now in the tree conflict state is also 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 conflicting resource or when running the **Edit conflict** action:
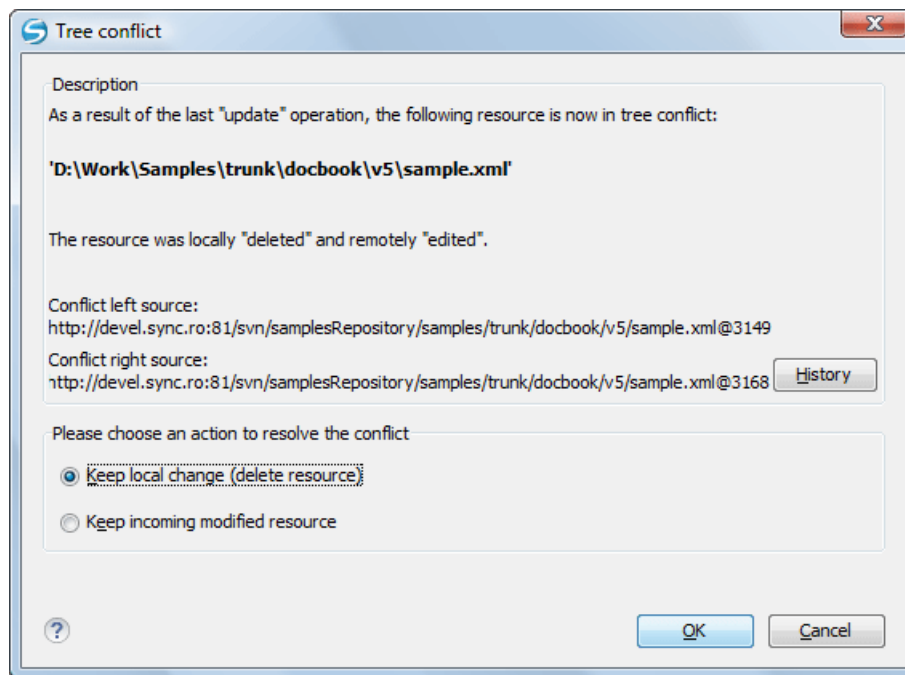
Figure 14: Resolve a tree conflict

• Keep 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 - Keeps 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 from **Working Copy** *view*. It updates the selected resources to the last synchronized revision (if remote information is available) or to the *HEAD* revision of the repository.

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 of text. Not applicable to binary resources (for example multimedia files, PDFs, executable program files)

- *Conflicting* - A conflicting change occurs when one or more of the same lines of a text file have been changed both remotely and locally.

If the resource contains only incoming changes or the outgoing changes do not intersect with incoming ones then the update will end normally and the Subversion system will merge incoming changes into the local file. In the case of a 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 the **Update to revision/depth** action from the **Working Copy** view (**All Files** view mode) or the **Update to revision** action from the *History view* contextual menu.

If you select multiple files and folders and then you perform an **Update** operation, all of those files and folders are updated one by one. The Subversion client makes sure that all files and 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 check out a newly versioned file, and found that an unversioned file with the same name already exists in your working folder. Subversion will never overwrite an unversioned file unless you specifically do this with an **Override and update** action. 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 from your local working copy to the repository. The **Commit** dialog box presents all the items that you are able to commit.
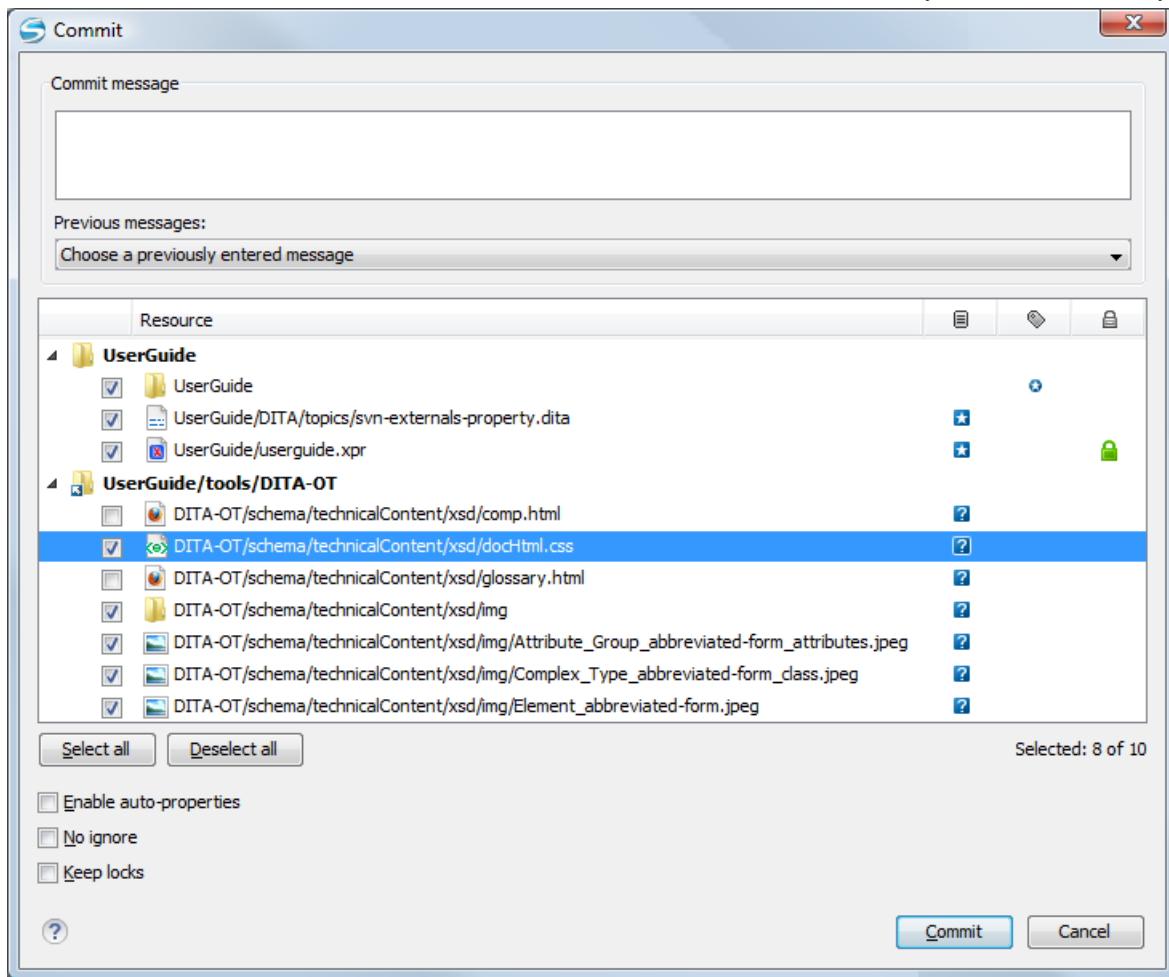
Figure 15: Commit dialog box

Enter a message to associate with the commit, or choose a previous message from the **Previous messages** list (the last 10 commit messages will be remembered even after restarting the SVN client application).

An item that can be committed has one of the following states: *added*, *modified* (content or properties), *replaced*, and *deleted*. All items that have one of these states are selected in the dialog box by default. If you do not want to commit one of the items, uncheck it.

Besides the items that have one of the mentioned states, Syncro SVN Client also includes the files being *unversioned* or *missing*. In order to be committed, these items are handled automatically:

• *unversioned* items are added under version control;
• *missing* items are deleted.

📋 **Note:** In case the **Show unversioned directories content** is disabled, the **Commit** dialog box does not display the items inside an *unversioned*directory.

Unversioned or *missing* items are not selected by default in the **Commit** dialog box, unless you have selected them explicitly when issuing the commit command.

📋 **Note:** In some cases, items that have one of the above states are not presented in the **Commit** dialog.

For example:

• items that have been *added* or *replaced* previously, but now are presented as *missing* after being removed from the file system, outside of an SVN client. Such items do not exist in the repository and you should use the **Delete** action to remove them from your working copy;

- items that have incoming changes from the repository, after a synchronization. You need to have your working copy up-to-date before committing your changes;
- files that, after a synchronization, appear as locked by other users or from other locations than the current working copy.

> **Note:** Due to dependencies between items, when you select or clear an *unversioned* (🔲) or *added* (🔲) item in the **Commit** dialog box, other items with one of these states can be selected or cleared automatically.

The modifications that will be committed for each file can be reviewed in the compare editor window by double clicking a file in the **Commit** dialog box, or by right clicking and selecting the **Show Modifications** action from the contextual menu. This option is available to review only file content changes, not property changes.

The 🗎 **Local file status** column indicates the actual state of the items and the 🏷 **Local properties status** column indicates whether the properties of an item are modified.

The 🔒 **Lock information** column is displayed in case at least one of the files in the **Commit** dialog box has lock information associated with it, valid against the commit operation.

The following options are available in this dialog box:

- **Enable auto-properties** or **Disable auto-properties** - enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in the `config` file of the Subversion configuration directory.

  > **Note:**
  >
  > This option is available only when there are defined properties to be applied automatically for resources newly added under version control. You can define these properties in the `config` file of the Subversion configuration directory, in the `auto-props` section.
  >
  > Based on the value of the `enable-auto-props` runtime configuration directive, the presented option is either **Enable auto-properties**, or **Disable auto-properties**.

- **Keep locks** - selecting the **Keep locks** option preserves any locks you set on various files.

  > **Note:** This option is available only when files that you locked are presented in the dialog box.

Each of the above options is activated only when you select an item for which the option can be applied.

Your working copy must be up-to-date with respect to the resources you commit. 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.

Committing to Multiple Locations

Although Subversion does not support committing to different locations at once, Syncro SVN Client offers this functionality regarding *external* items.

If items to be committed belong to different *external* definitions found in the working copy, they are grouped under the corresponding item that indicates their repository origin. Each parent item is rendered bold and its corresponding repository location is presented when hovering it. Parent items are decorated with a small arrow (🔗) if they are *external* definitions. The working copy root directory is never decorated and is not presented if there are no *external* items listed (all items belong to the main working copy). Each child item is presented relative to the parent item.

> **Note:** When an *external* directory has modifications of its own, it is presented both as a parent item and as an item that you can select and commit. This is always the case for *external* files.

For SVN 1.6 working copies, each set of items belonging to an *external* definition is committed independently, resulting multiple revisions (one for each set).

For SVN 1.7 working copies, the sets of items belonging to *external* definitions from the same repository are committed together, resulting a single revision. So, the number of revisions can be smaller than the number of *externals*. External

definitions are considered from the same repository if they have the same protocol, server address, port, and repository address within the server.

> 📄 **Note:** *External* files are always from the same repository as the parent directory which defines them, so they are always committed together with the changes from their parent directory.

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. The only requirement is that the user includes the bug ID in the commit message that he enters in the **Commit** dialog. 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*, the 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. If the property is set to *true* or 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 regular expression string, for 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 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

This section explains how to obtain information for a SVN 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 want more detailed information about a given resource you can use the **Information** action from the **Working Copy** 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.

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.

Request History for a Resource

In Apache Subversion[TM], 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 *Repositories view*, *Working Copy view*, *Revision Graph*, or *Directory Change Set view*. From the **Working copy view** you can display the history of local versioned resources.

Management of SVN Properties

In the *Properties view* you can read and set the Apache Subversion[TM] 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.

If you want to change the properties of a given resource you need to select that resource from the *Working Copy view* and run 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 containing:

- **Name** - 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** - 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.

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

- **Name** - Property name (cannot be changed).
- **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 the **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.

Branches and 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 Apache Subversion^{TM} 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* or in the *Repositories view*, select the resource which you want to copy to a branch or tag, then select the command *Branch / Tag...* from the **Tools** menu.
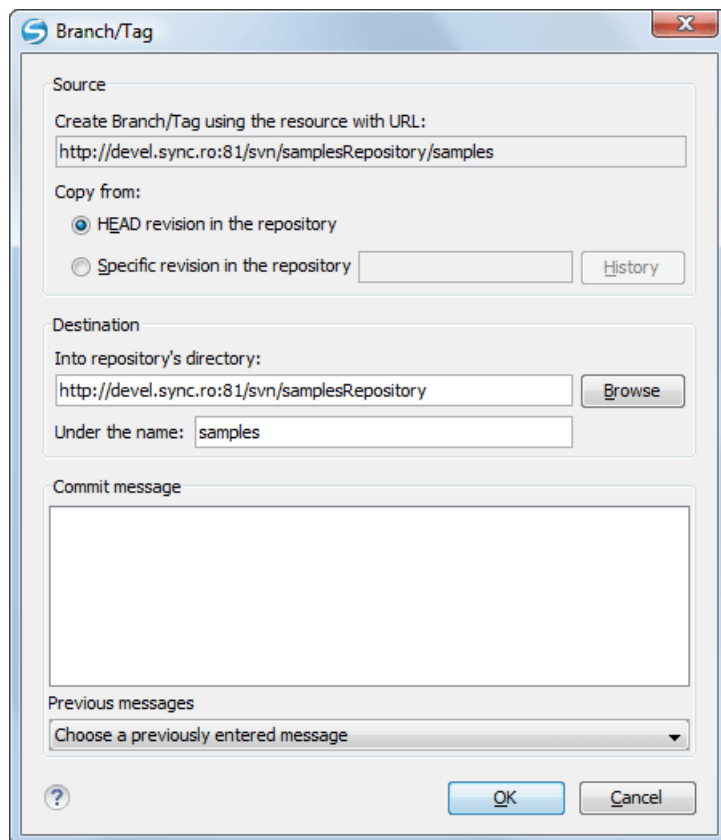

Figure 16: The Branch / Tag dialog

The default target URL for the new branch / tag will be the repository URL of the selected resource from your working copy, divided in two: the URL of the parent and the selected resource's name. You may specify other resource name if you want to make a branch / tag using a different name than the one of the selected resource, by modifying the field labeled **Under the name:**. The new branch / tag will be created as child of the specified repository directory URL and having the new provided name. To change the parent directory URL to the new path for your branch / tag, click on the **Browse** button and choose a repository target directory for your resource.

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** action can be found in the **Tools** menu of Syncro SVN Client. 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:

- merge revisions - integrate the modifications from a branch into the trunk or a different branch, when the branches are created from the same trunk;
- reintegrate a branch - integrate the modifications from a branch into the trunk;
- merge two different trees - the general case of integrating some changes between two different branches.

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.

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:
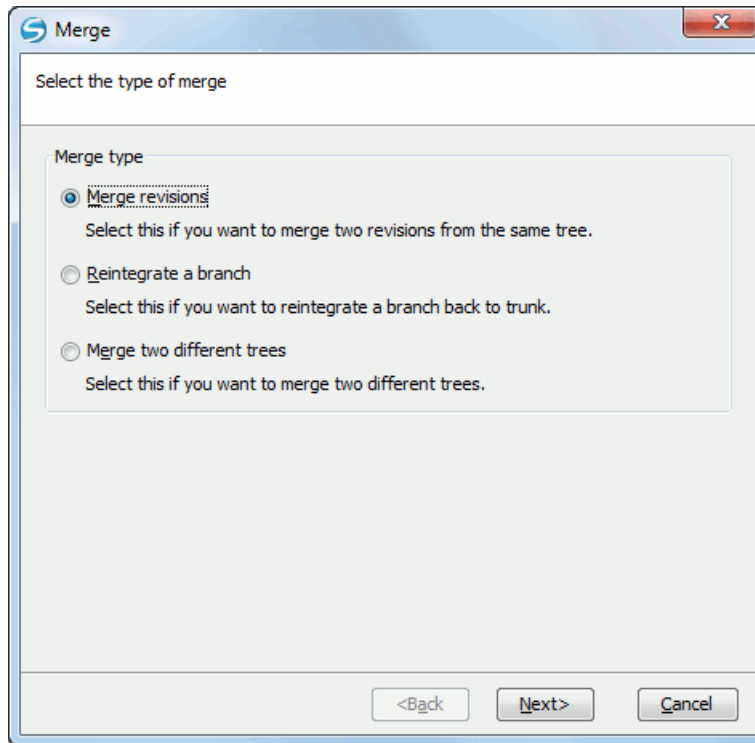
Figure 17: The Merge Wizard - The Merge Type

2. Select the option **Merge revisions**.

3. Press the **Next** button.
The second step of the **Merge** wizard is displayed:

Figure 18: The Merge Wizard - Revisions Range

4. In the **From URL** field enter the folder URL of the branch or tag containing the changes that you want to port into your working copy.

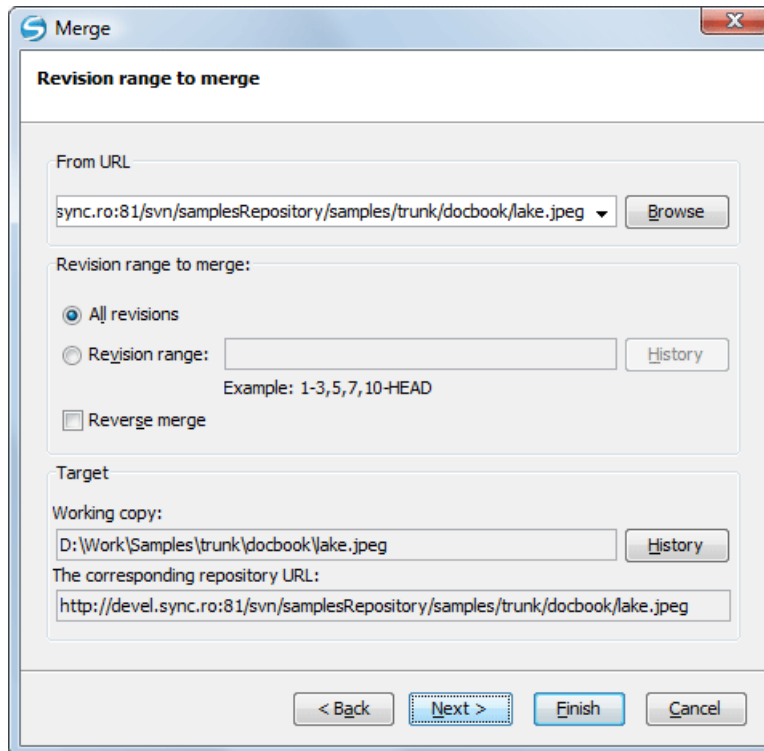   You may also click the **Browse** button to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

5. Select the revision range.
   a) Choose between all revisions and a revision range.

      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.

      The **History** button opens the *the History dialog* which allows selecting the list of revisions to be merged in the easiest way. One or several revisions can be selected in that dialog.

      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.

   b) Check the **Reverse merge** box (optional).

      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.

6. Specify the target where the changes of the branch will be integrated.
   a) Enter the working copy folder in the **Working copy** field.

      By default the root folder of the current working copy from the **Working Copy** view is set in this field.

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

   b) Specify the URL of the target branch that will receive the changes.

      By default the URL of the repository of the current working copy from the **Working Copy** view is set in this field.

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

7. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:
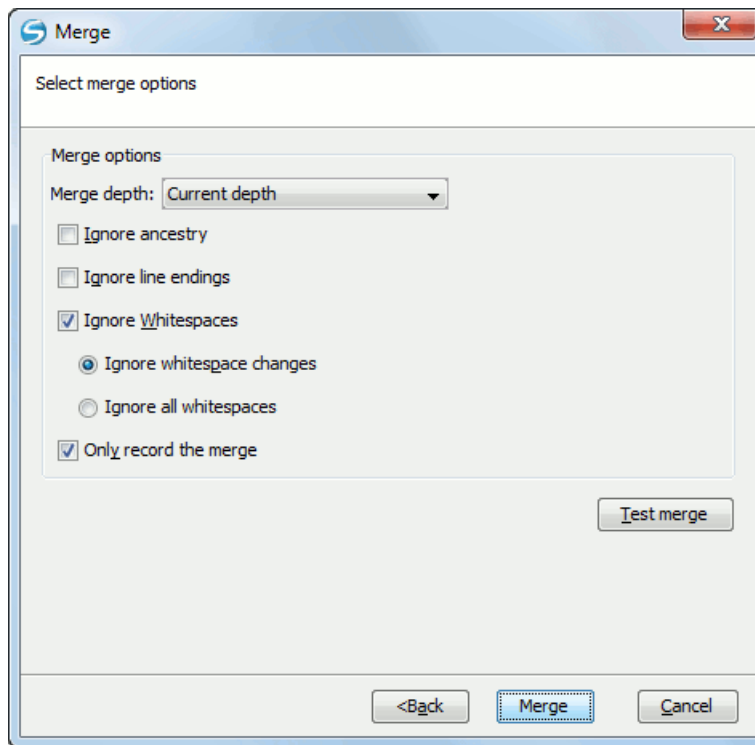
Figure 19: The Merge Wizard - Advanced Options

8. Set advanced options if necessary before starting the merge process.

a) Set the depth of the merge operation in the **Merge depth** combo box.

You can specify how far down into your working copy the merge should go by selecting one of the following values:

- Current depth
- Recursive (infinity)
- Immediate children (immediates)
- File children only (files)
- This folder only (empty)

The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

b) Check the **Ignore ancestry** checkbox (optional).

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.

c) Check the **Ignore line endings** checkbox (optional).

d) Check the **Ignore Whitespaces** checkbox (optional).

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.

e) Check the **Only record the merge** checkbox (optional).

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.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you 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.

9. Press the **Merge** button.
The merge operation is executed.

10. Optionally resolve the conflicts that were created by the merge operation.

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, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.
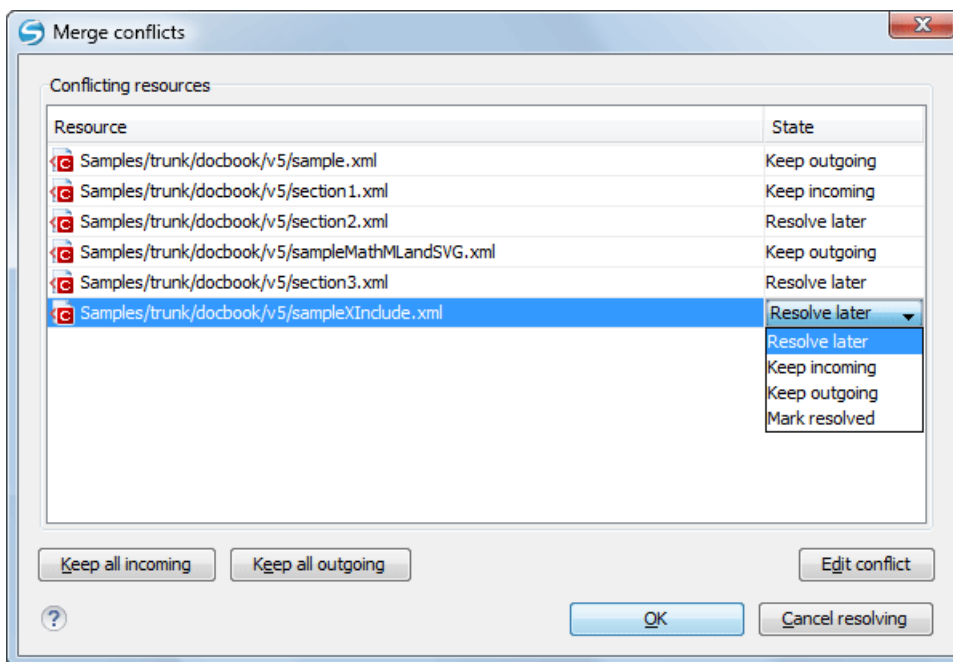

Figure 20: Merge Conflicts Dialog

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.

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

Reintegrate a Branch

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 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).

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 Apache Subversion^{TM} to calculate the correct revision ranges and to perform additional checks which ensure that the branch has been fully updated with trunk changes. The range of revisions to merge will be calculated automatically. 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.
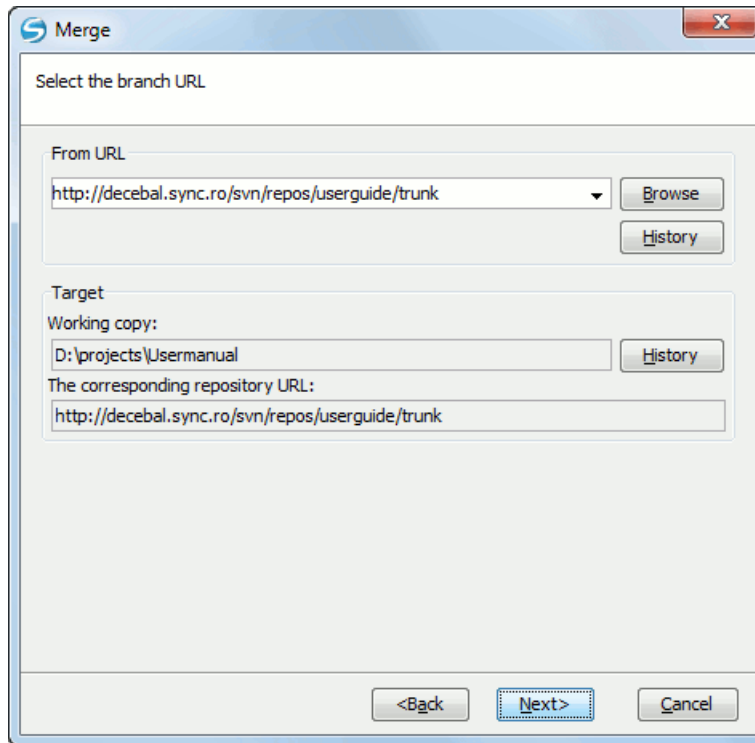
1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:
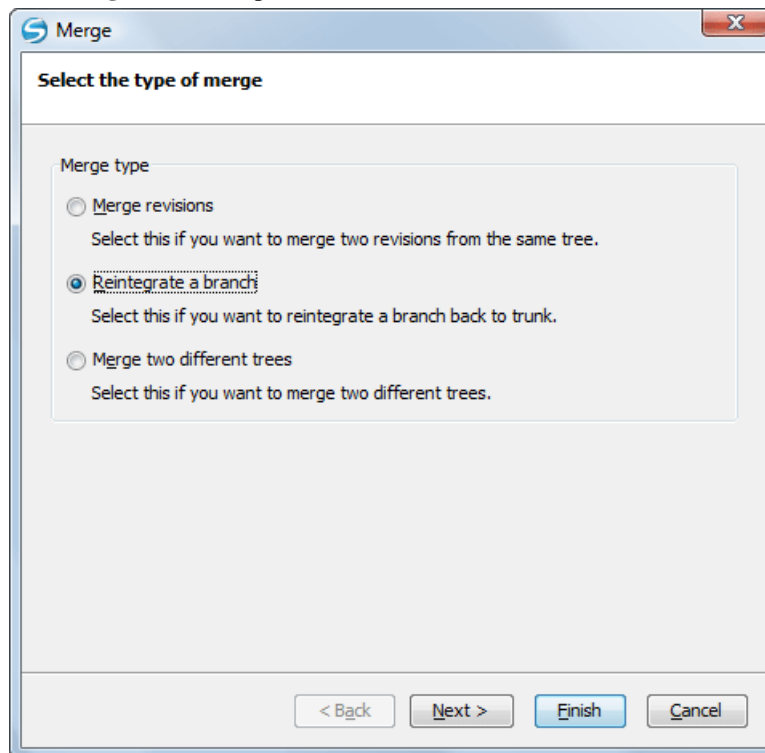
Figure 21: The Merge Wizard - The Merge Type

2. Select the option **Reintegrate a branch**.
3. Press the **Next** button.
   The second step of the **Merge** wizard is displayed:

Figure 22: The Merge Wizard - Reintegrate Branch

4. In the **From URL** field enter the folder URL of the branch or tag containing the changes that you want to integrate.

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.

The History button opens *the History dialog* which allows you to select a revision number of the repository with the changes.

5. Select the target of the operation.
   a) Select the path of the working copy.
   b) Select the URL of the repository corresponding to the working copy.

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.

6. Press the **Next** button.
The **Merge Options** step of the wizard is opened:
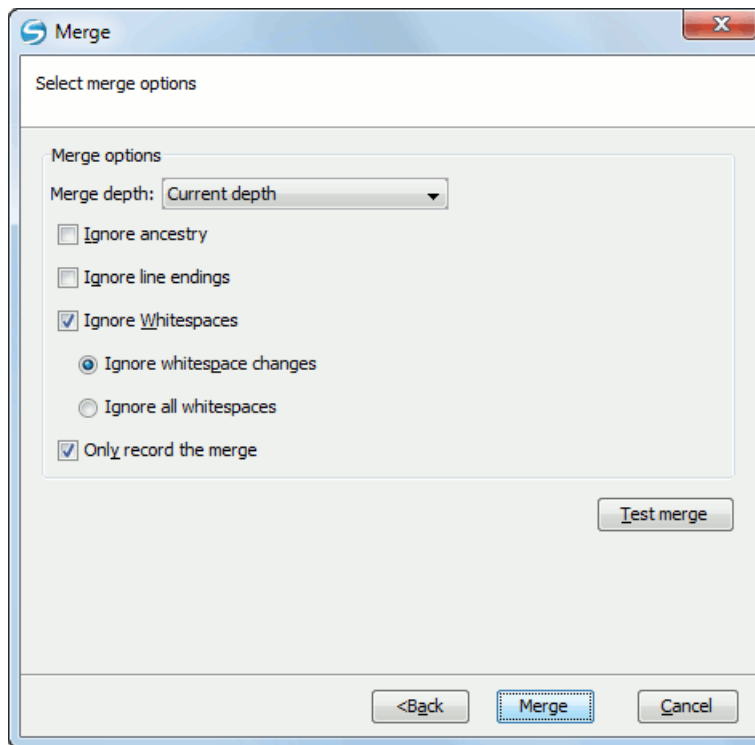
Figure 23: The Merge Wizard - Advanced Options

7. Set advanced options if necessary before starting the merge process.

a) Set the depth of the merge operation in the **Merge depth** combo box.

You can specify how far down into your working copy the merge should go by selecting one of the following values:

- Current depth
- Recursive (infinity)
- Immediate children (immediates)
- File children only (files)
- This folder only (empty)

The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

b) Check the **Ignore ancestry** checkbox (optional).

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.

c) Check the **Ignore line endings** checkbox (optional).

d) Check the **Ignore Whitespaces** checkbox (optional).

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.

e) Check the **Only record the merge** checkbox (optional).

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.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you 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.

8. Press the **Merge** button.
   The merge operation is executed.

9. Optionally resolve the conflicts that were created by the merge operation.

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, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.
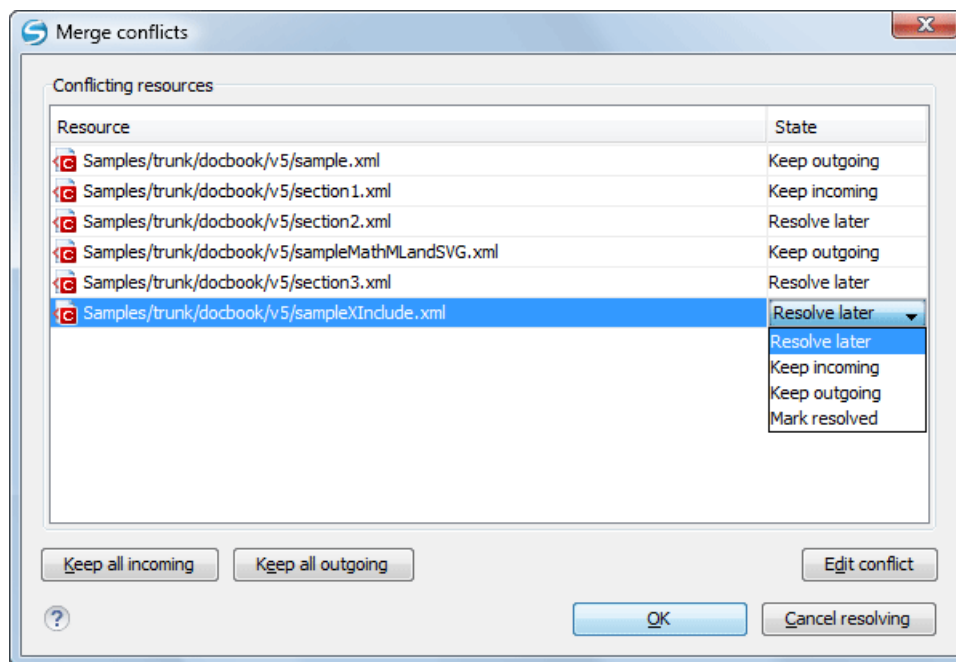
Figure 24: Merge Conflicts Dialog

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.

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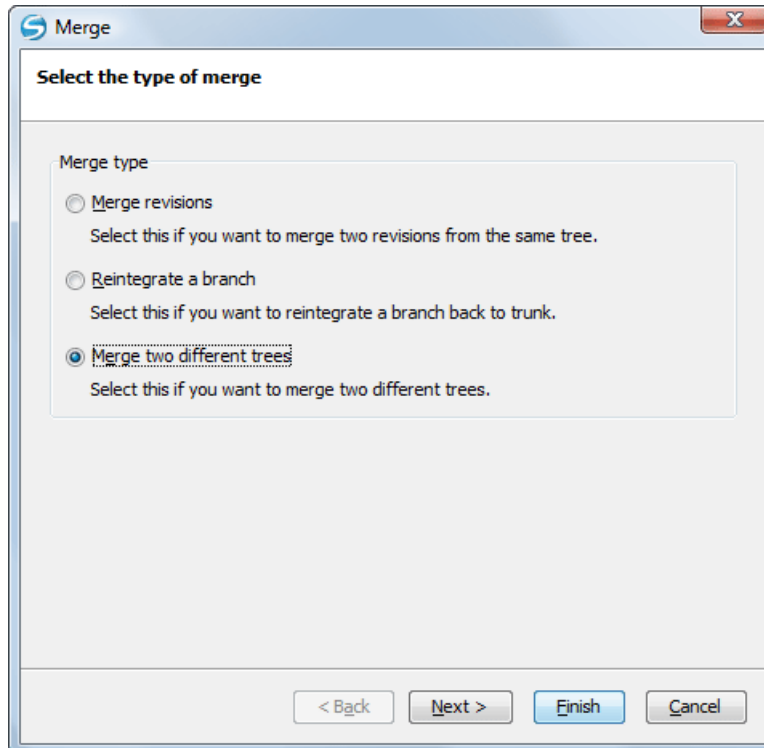
Merge Two Different Trees

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.

📋 **Note:** If the server does not support merge-tracking then this is the only way to merge a branch back to trunk.
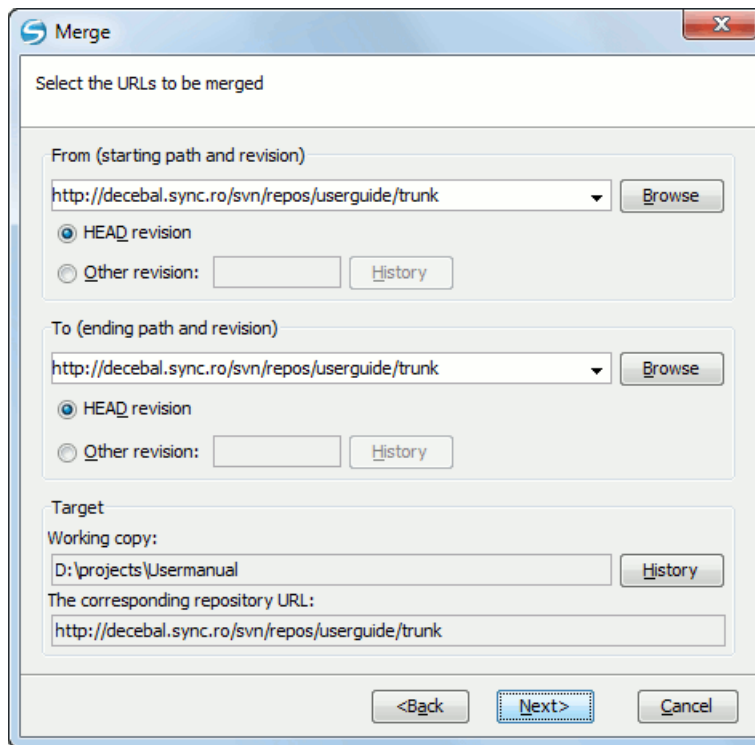
1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:

Figure 25: The Merge Wizard - The Merge Type

2. Select the option **Merge two different trees**.
3. Press the **Next** button.
   The second step of the **Merge** wizard is displayed:

Figure 26: The Merge Wizard - Merge Two Different Trees

4. Specify the URL of the first tree in the **From** field.

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.

By default the start URL will be the URL of the selected file in the working copy. If you want to specify a different URL you should browse the repository and select a start URL and a revision.

a) Select a URL in the **From** field.

b) Select a revision of the repository from the **From** field.

In the **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.

By default the HEAD revision is selected. If you want a previous revision you have to select the **Other revision** option and press *the History button* to see a list of all revisions.

5. Specify the URL of the second tree in the **To** field.

a) Select a URL in the **To** field.

b) Select a revision of the repository from the **To** field.

By default the HEAD revision is selected. If you want a previous revision you have to select the **Other revision** option and press *the History button* to see a list of all revisions.

6. Specify the target of the merge operation in the **Target** panel.

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.

a) Specify the working copy path in the **Working copy** field.

b) Specify the repository URL corresponding to the working copy.

7. Press the **Next** button.

The **Merge Options** step of the wizard is opened:
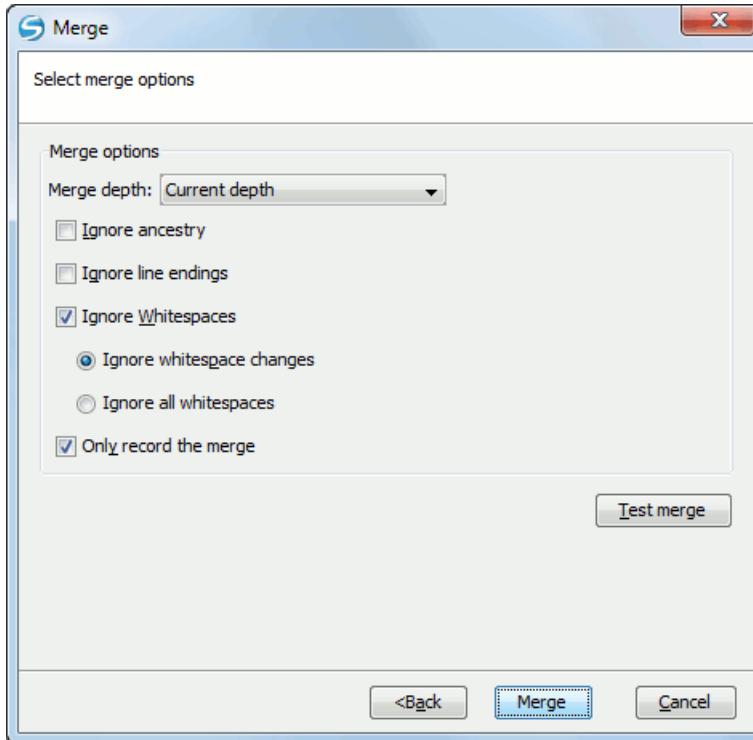
Figure 27: The Merge Wizard - Advanced Options

8. Set advanced options if necessary before starting the merge process.

a) Set the depth of the merge operation in the **Merge depth** combo box.

You can specify how far down into your working copy the merge should go by selecting one of the following values:

- Current depth
- Recursive (infinity)
- Immediate children (immediates)
- File children only (files)
- This folder only (empty)

The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.

b) Check the **Ignore ancestry** checkbox (optional).

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.

c) Check the **Ignore line endings** checkbox (optional).

d) Check the **Ignore Whitespaces** checkbox (optional).

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.

e) Check the **Only record the merge** checkbox (optional).

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.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you 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.

9. Press the **Merge** button.
The merge operation is executed.

10. Optionally resolve the conflicts that were created by the merge operation.

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, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.


Figure 28: 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.

Merge Options

1. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:
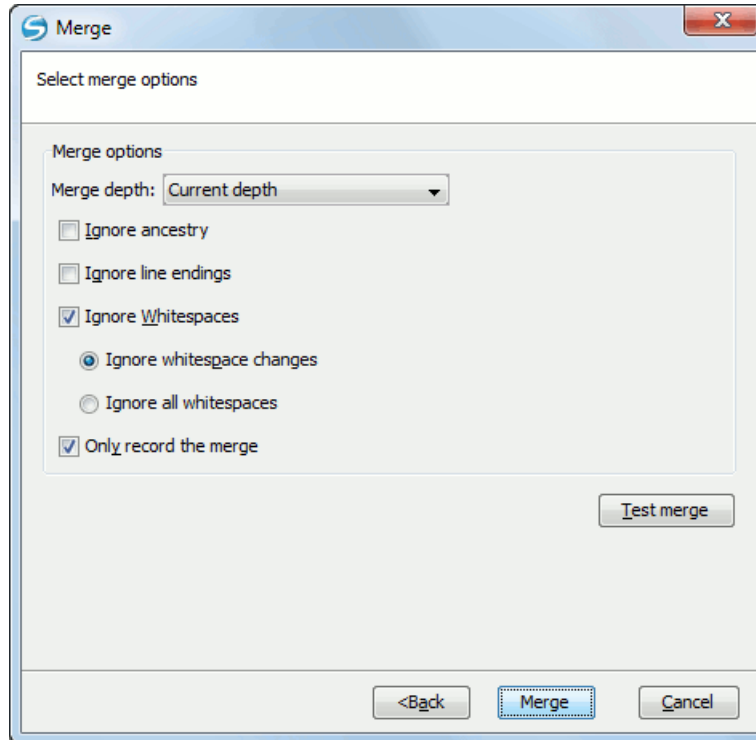
Figure 29: The Merge Wizard - Advanced Options

2. Set advanced options if necessary before starting the merge process.
   a) Set the depth of the merge operation in the **Merge depth** combo box.

   You can specify how far down into your working copy the merge should go by selecting one of the following
   values:

   - Current depth
   - Recursive (infinity)
   - Immediate children (immediates)
   - File children only (files)
   - This folder only (empty)

   The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working
   copy.

   b) Check the **Ignore ancestry** checkbox (optional).

   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.

   c) Check the **Ignore line endings** checkbox (optional).
   d) Check the **Ignore Whitespaces** checkbox (optional).

   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.

e) Check the **Only record the merge** checkbox (optional).

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.

f) Press the **Test merge** button (optional).

By pressing the **Test merge** button you 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.

3. Press the **Merge** button.
The merge operation is executed.

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

Merge Branches Task - First Step

Go to menu **Tools** > **Merge ...**
The **Merge** wizard is opened:
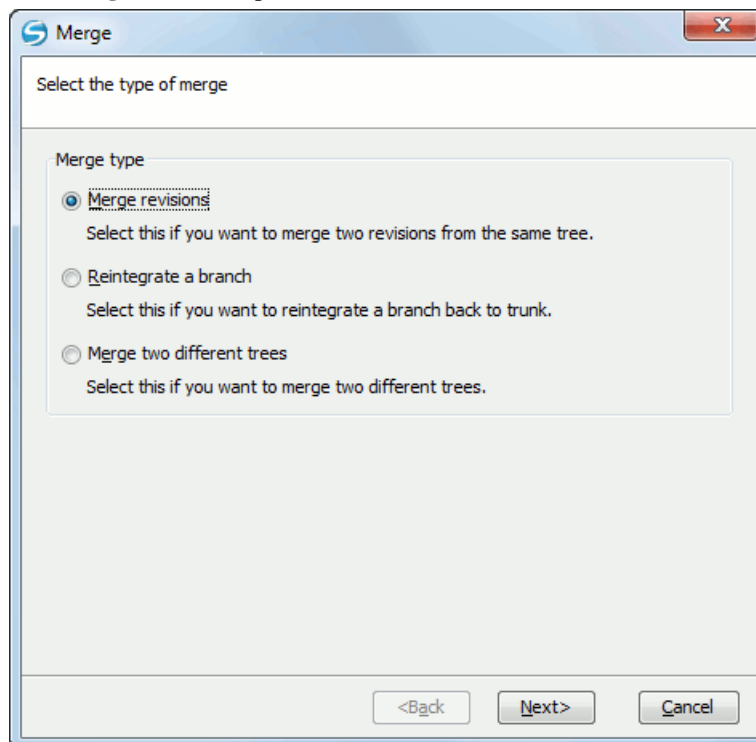
Figure 30: The Merge Wizard - The Merge Type

Resolve Merge Conflicts

Optionally resolve the conflicts that were created by the merge operation.

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, the following dialog will appear presenting you the resources that are in conflict. In this dialog you can choose a way in which every conflict should be resolved.
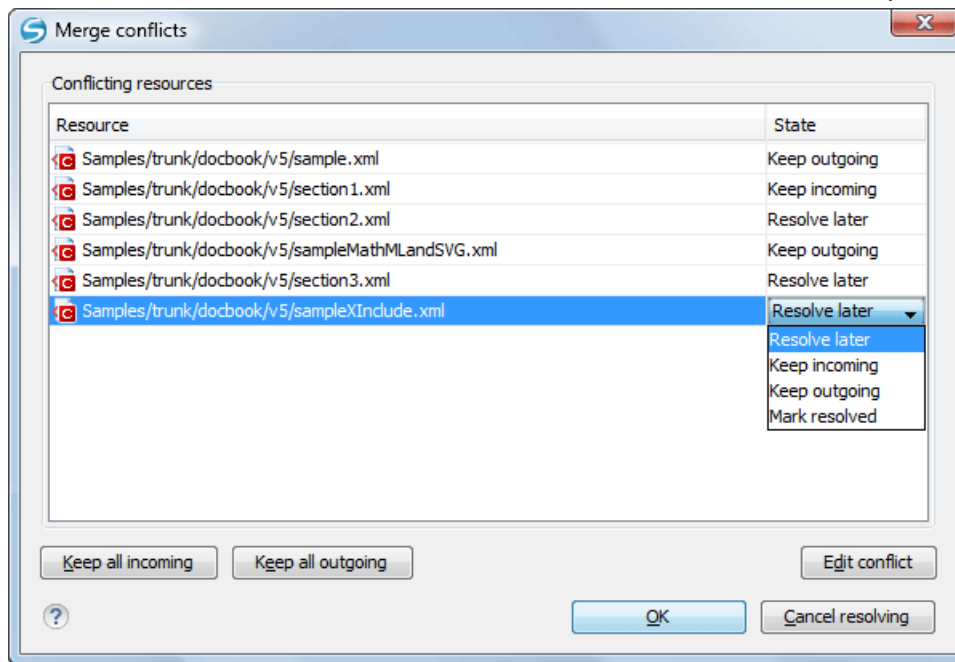
Figure 31: Merge Conflicts Dialog

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. The action is available on the **Tools** menu when a versioned resource is selected in the current working copy that is displayed in *the Working Copy view*.

Relocate a Working Copy

Sometimes the base URL of the repository is changed after a working copy is checked out from that URL, for example the repository itself is relocated to a different server. In such cases 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 called **Relocate** and is available on the **Tools** menu when the selected item in *the Working Copy view* is a versioned folder.

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.

Patches

This section explains how to work with patches in Syncro SVN Client.

What Is a Patch

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*.

Creating a patch in Apache Subversion$^{\text{TM}}$ implies the access to two states (revisions) of a project:

- the current working copy and a revision from the repository - if you have not committed yet your current working copy and prefer not to do it, you create a patch between the current working copy and a revision from the repository;
- two repository revisions - if both states are two revisions already committed to the repository.

Create a Patch Between Working Copy and Repository Revision

When the changes that must be included in a patch were not committed to the repository the patch should be created with the differences between the selected item(s) of the working copy and a repository revision. The steps are the following:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard:
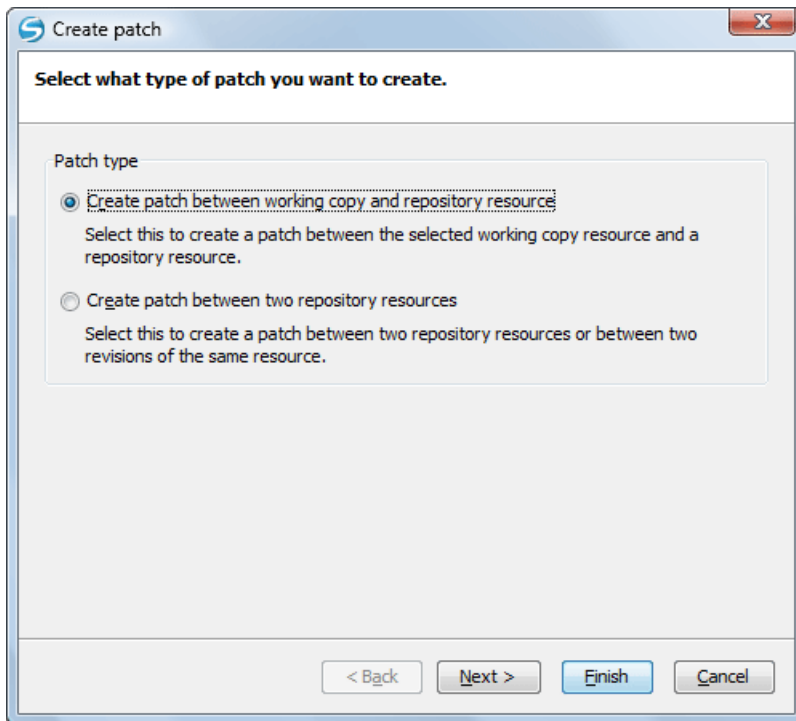
Figure 32: The Create Patch Wizard - Patch Type

2. Select the first option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:
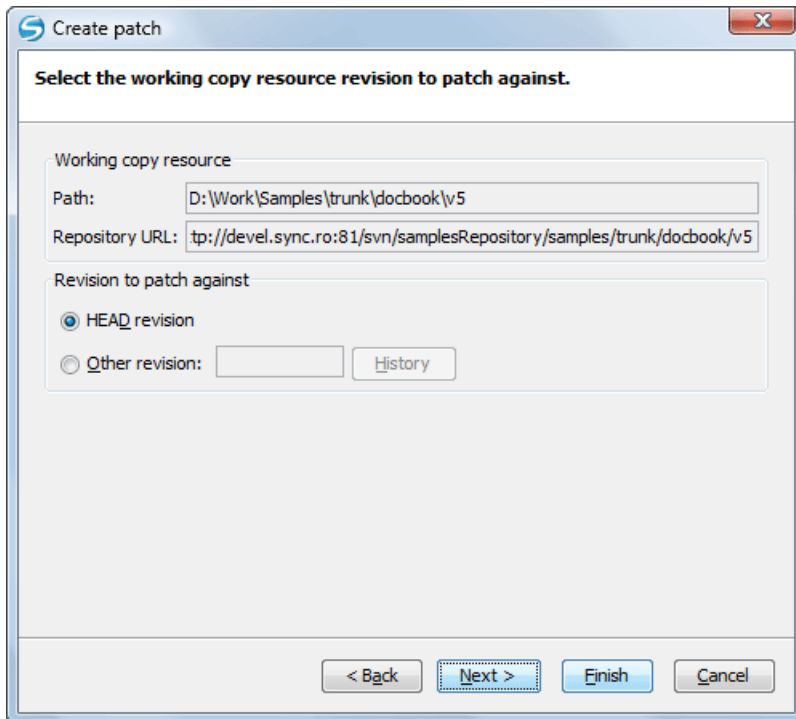
Figure 33: The Create Patch Wizard - Step 2

4. Specify the working copy resource.

  a) Specify the local file path of the working copy resource.

  b) Specify the repository URL corresponding to the working copy resource.

5. Select the repository revision.

  You have the option of choosing between the HEAD revision and a specific revision number. For the second option you should press *the History button* to display a list of the repository revisions.

6. Press the **Next** button.
  The next step of the wizard is displayed:
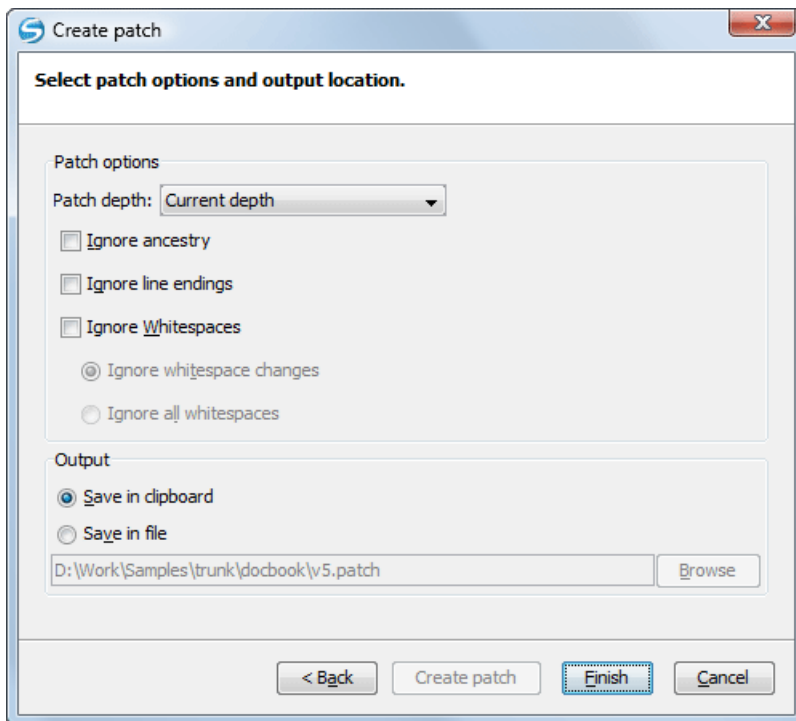
Figure 34: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

In case of a file resource the depth is always zero. In case of a folder resource the depth has one of the following 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).

8. Select the **Ignore ancestry** checkbox (optional).

If checked, the SVN ancestry that exists when the two URLs have a common SVN history is ignored.

9. Select the **Ignore line endings** checkbox (optional).

If checked, the differences in line endings are ignored when the patch is created.

10. Select the **Ignore whitespaces** checkbox (optional).

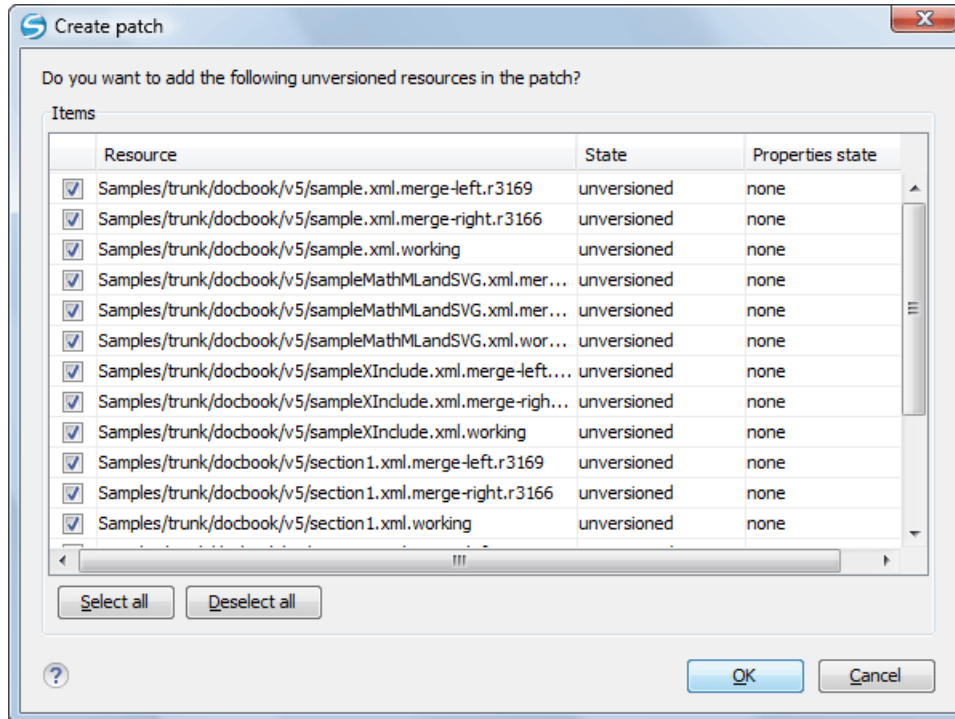If checked, the differences in whitespaces are ignored when the patch is created.

11. Select the output location.

- **Save in clipboard** - The patch will be created and saved in clipboard. This is useful when you do not want to save the patch in a file on disk.
- **Save in file** - The patch will be created and saved in the specified file.

12. Press the **Next** button.
This will go to the final step of the wizard:

Figure 35: The Patch Wizard - Add Unversioned Resources

13. Select the unversioned files that will be included in the 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.

14. Press the OK button.
The patch is created by applying all the specified options.

Create a Patch Between Two Repository Revisions

When a patch must include the differences between two repository revisions, in the same repository or in two different repositories, the steps for creating the patch are the following:

1. Go to menu **Tools** > **Create patch**.
This opens the **Create patch** wizard:
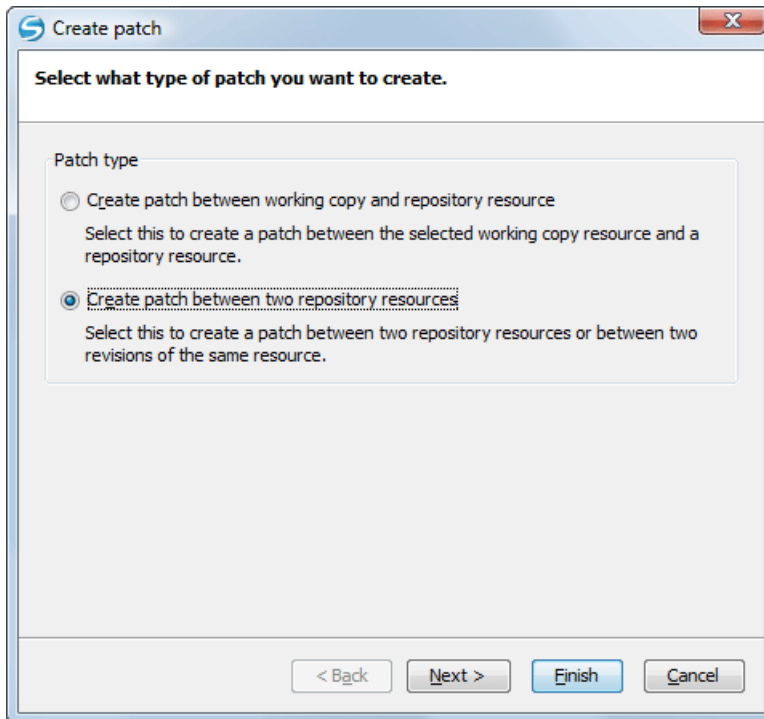
Figure 36: The Create Patch Wizard - Patch Type

2. Select the second option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:
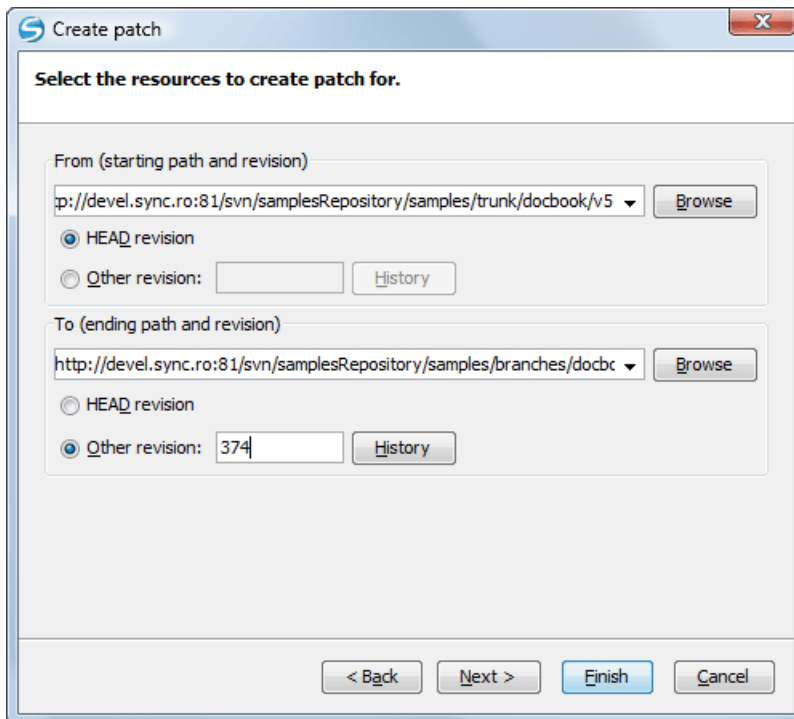
Figure 37: The Create Patch Wizard - Step 2

4. Select the first repository revision in the **From** panel.
5. Select the second repository revision in the **To** panel.

For both revisions you have the option of choosing between the HEAD revision and a specific revision number. For a specific number you should press *the History button* to display a list of the repository revisions.

6. Press the **Next** button.

The next step of the wizard is displayed:
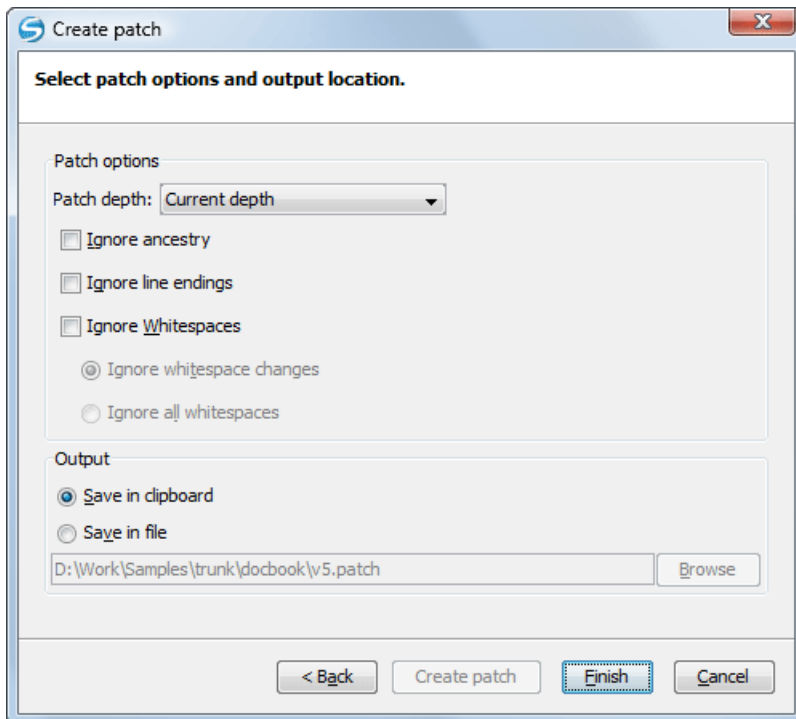
Figure 38: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

In case of a file resource the depth is always zero. In case of a folder resource the depth has one of the following 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).

8. Select the **Ignore ancestry** checkbox (optional).

If checked, the SVN ancestry that exists when the two URLs have a common SVN history is ignored.

9. Select the **Ignore line endings** checkbox (optional).

If checked, the differences in line endings are ignored when the patch is created.

10. Select the **Ignore whitespaces** checkbox (optional).

If checked, the differences in whitespaces are ignored when the patch is created.

11. Select the output location.

- **Save in clipboard** - The patch will be created and saved in clipboard. This is useful when you do not want to save the patch in a file on disk.
- **Save in file** - The patch will be created and saved in the specified file.

12. Press the **Next** button.

This will go to the final step of the wizard:

Figure 39: The Patch Wizard - Add Unversioned Resources

13. Select the unversioned files that will be included in the 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.

14. Press the OK button.

The patch is created by applying all the specified options.

Working with Repositories

This section explains how to locate and browse SVN repositories in Syncro SVN Client.

Importing Resources Into a Repository

This is the process of taking a project and importing it into a repository so that it can be managed by an Apache Subversion[TM] server. 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 *using an existing working copy*.

This process is started from menu **Repository** > **Import** > **Import Folder Content**. The same action is available in the **Repositories** view contextual menu. 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.

Exporting Resources From a 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.

The export dialog is very similar to the check out dialog:
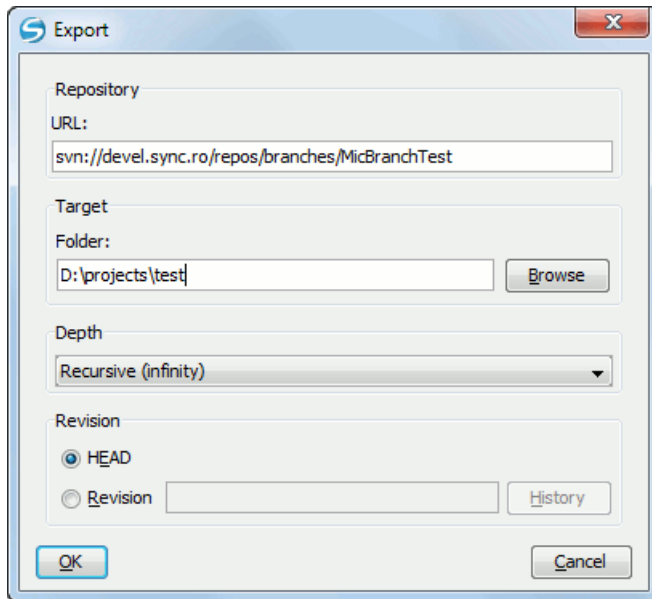
Figure 40: Export from Repository

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 a Repository

Once you have a location defined in the *Repositories 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 to** and **Move to** action allows you to copy and move individual or multiple resources to a specific directory from the *HEAD* revision of the repository.
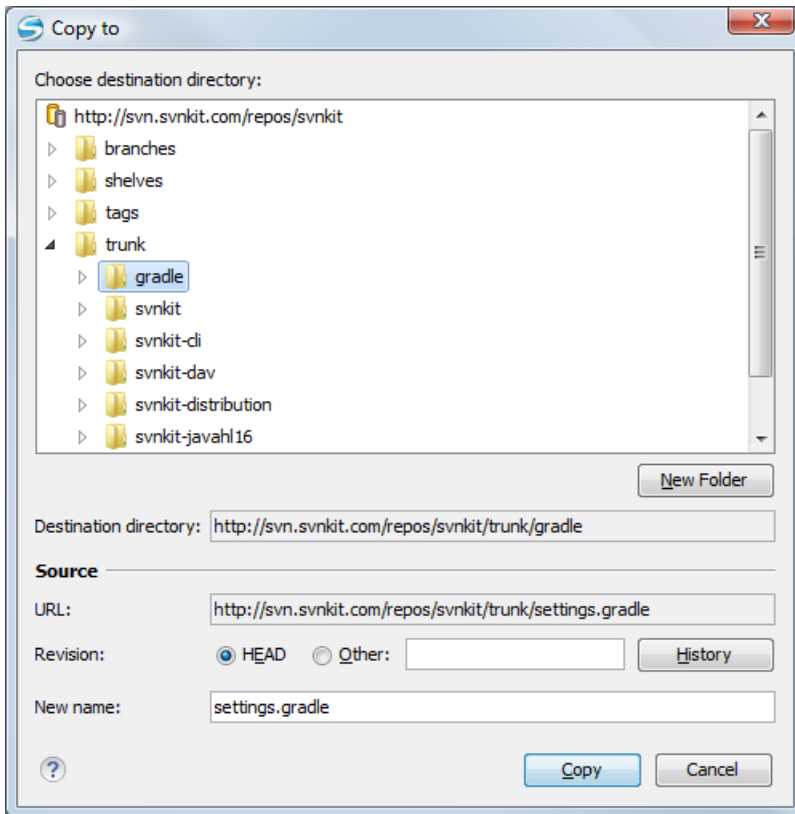
Figure 41: Copy/Move Items in Repository

The dialog box used to copy or move items allows you to browse the *HEAD* revision of the repository and select the destination of the items, presenting its repository URL below the tree view.

The **Source** section presents relevant options regarding the item(s) that you move or copy:

• **URL** - this field is displayed only if you copy/move a single item;
• **Revision** - presents the revision from which you copy one or more items, allowing you to also choose another revision;

> **Note:** Since only items from the HEAD revision can be moved, the **Revision** options are not presented for the **Move to** action.

> **Note:** When you copy a single item while browsing a revision other than *HEAD*, the **Revision** options present this revision but does not allow you to change it. The same applies if copying multiple items.

• **New name** - This option is presented when you copy or move a single item, allowing you to also rename it.

Another useful action is **Delete**, allowing you to erase resources directly from the repository.

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

Sparse Checkout

Sometimes you need to check out only certain parts of a directory tree. For this you can check out the top directory (*the action **Check out** from the **Repositories** view*) and then update recursively only the needed directories (*the action **Update** from the **Working Copy** view*). Now, each directory has a depth set to it, which has four possible values:

• **Recursive** (*infinity*) - Updates all descendant directories and files recursively.
• **Immediate children** (*immediates*) - Updates the directory, including direct child directories and files, but does not populate the child directories.
• **File children only** (*files*) - Updates the directory, including only child files without the child directories.

- **This folder only** (*empty*) - Updates only the selected directory, without updating any children.

For some operations, you can use as depth the current depth registered on the directories from the working copy (the value **Current depth**). This is the depth value defined in a previous check out or update operation.

The sparse checked out directories are presented in the *Working Copy view* with a marker corresponding to each depth value, in the top left corner, as follows:

- - **Recursive** (*infinity*) - This is the default value and it is has no mark. The directory has no limiting depth.
- - **Immediate children** (*immediates*) - The directory is limited to direct child directories (without contents) and files.
- - **File children only** (*files*) - The directory is limited to direct child files only.
- - **This folder only** (*empty*) - The directory has *empty* depth set.

A depth set on a directory means that some operations process only items within the specified depth range. For example, **Synchronize** on a working copy directory reports the repository modified items within the depth set on the directory and those existing in the working copy outside of this depth.

The depth information is also presented in the **SVN Information** dialog box and in the tool tip displayed when hovering a directory in the **Working Copy** view.

This feature requires the SVN client to support SVN 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.

Syncro SVN Client Views

The main working area occupies the center of the application window, which contains the most important views:

- *Repositories View*
- *Working Copy View*
- *History View*
- *Console View*

The other views that support the main working area are also presented in this section.

Repositories View

The **Repositories** view allows you to define and manage Apache Subversion[TM] repository locations and browse repositories. 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 repository. More information about each resource is displayed in a tabular form:

- **Date** - Date when the resource was last modified;
- **Revision** - The revision number at which the resource was last time modified;
- **Author** - Name of the person who made the last modification on the resource;
- **Size** - Resource size on disk;
- **Lock information** - Information about the lock status of a file. When a repository file is locked by a user the icon is displayed in this column. If no icon is displayed the file is not locked. The tooltip of this column displays the details about lock:

  - owner - the name of the user who created the lock;
  - date - the date when the user locked the file;
  - expires on - date when the lock expires. Lock expiry policy is set in the repository options, on the server side;
  - comment - the message attached when the file was locked.

- **Type** - Contains the resource type or file extension.

Name	Date	Revision	Author	🔒	Size	Type
Repositories						
▲ http://devel-new.sync.ro/svn/svnrepos						Repository
▲ EclipseEditorArea3X	Jun 13, 2011	73635	mircea			Folder
▷ .settings	May 25, 2011	73008	mircea			Folder
▷ lib	Jun 13, 2011	73635	mircea			Folder
▷ src	Jun 13, 2011	73635	mircea			Folder
▷ tools	May 25, 2011	73008	mircea			Folder
.classpath	May 25, 2011	73008	mircea		1 KB	File
.project	May 25, 2011	73008	mircea		1 KB	File
ant	May 25, 2011	73008	mircea		1 KB	File
ant.bat	Jun 10, 2011	73594	mircea	🔒	1 KB	bat
build.xml	Jun 10, 2011	73594	mircea	🔒	2 KB	xml
▷ branches	Jun 20, 2011	73893	radu_coravu			Folder
▷ step1	Today 10:39 AM	74029	serban			Folder
▷ tags	Jun 1, 2011	73165	sorin			Folder
▷ trunk	Today 12:14 PM	74037	radu_coravu			Folder
▷ https://tei.svn.sourceforge.net/svnroot/tei...						Repository
▲ https://saxon.svn.sourceforge.net/svnroot...						Repository
▷ latest8.8	Jul 22, 2008	287	mhkay			Folder
▷ latest8.9	Jul 22, 2008	286	mhkay			Folder
▷ latest9.0	Dec 9, 2008	347	mhkay			Folder
▷ latest9.1	Dec 22, 2010	594	mhkay			Folder
▷ latest9.2	Dec 22, 2010	594	mhkay			Folder
▷ latest9.3	Jun 22, 2011	621	mhkay			Folder
▷ tags	Oct 30, 2010	579	mhkay			Folder
▲ trunk	Sep 5, 2006	4	mhkay			Folder
▷ bj	Sep 5, 2006	4	mhkay			Folder

Figure 42: Repositories View

Toolbar

The **Repositories** view's toolbar contains the following buttons:

- **New Repository Location** - Allows you to enter a new repository location by means of the **Add SVN Repository** dialog.

- **Move Up** - Move the selected repository up one position in the list of repositories in the **Repositories** view.

- **Move Down** - Move the selected repository down one position in the list of repositories in the **Repositories** view.

- **Collapse all** - Collapses all repository trees.

- **Stop** - Stops the current repository browsing operation executed when a repository node is expanded. This is useful when the operation takes too long or the server is not responding.

- **Settings** - Allows you to configure the resource table appearance.

Contextual Menu Actions

The **Repositories** view contextual menu contains different actions depending on the selected item. If a repository location is selected, the following management actions are available:

- **New Repository Location** (**Ctrl (Meta on Mac OS) + Alt + N**) - Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.
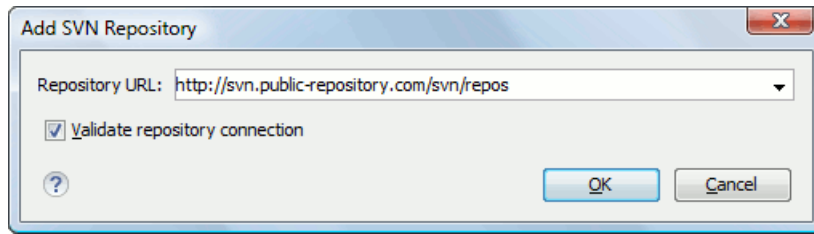
Figure 43: Add SVN Repository Dialog Box

If the **Validate repository connection** option is selected, the URL connection is validated before being added to the **Repositories** view.

- 🔧 **Edit Repository Location** (**Ctrl (Meta on Mac OS) + Alt + E**) - Context-dependent action that allows you to edit the selected repository location using the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.
- **Change the Revision to Browse** (**Ctrl (Meta on Mac OS) + Alt + Shift + B**) - Context-dependent action that allows you to change the selected repository revision using the **Change the Revision to Browse** dialog. It is active only when a repository location root is selected.
- ✖ **Remove Repository Location** (**Ctrl (Meta on Mac OS) + Alt + Shift + R**) - 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.

The following actions are common to all repository resources:

- **Open** - Opens the selected file in the Editor view in read-only mode.

- **Open with** - Displays the *Open with...* dialog to specify the editor in which the selected file is opened. In case multiple files are selected, only external applications can be used to open the files.

- **Save as** - Saves the selected files locally, as they are in the browsed revision.

- 🔄 **Refresh** - Refreshes the resource selected in the **Repositories** view.
- 📥 **Check out** (**Ctrl (Meta on Mac OS) + Alt + Shift + C**) - Allows you to create a working copy from a repository directory, on your local file system. To read more about this operation, see the section *Check out a working copy*.
- **Branch/Tag** - Allows you to create a branch or a tag from the selected folder in the repository. To read more about how to create a branch/tag, see the *Creation and management of Branches/Tags* section.
- **Share project** - Allows you to *share a new project* using an SVN repository. The local project is automatically converted into an SVN working copy.
- **Import** sub-menu:

  - **Import Folder Content** (**Ctrl (Meta on Mac OS) + Alt + Shift + M**) - Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.

    📝 **Note:** The difference between **Import Folder Content** and **Share project** actions is that the latter one also converts the selected directory into a working copy.

  - **Import File(s)** (**Ctrl (Meta on Mac OS) + Alt + I**) - Imports the files selected from the files system into the selected folder from the repository.

- **Export** - Exports a folder from the repository to the local file system.

- 🕐 **Show History** (**Ctrl (Meta pn Mac OS) + H**) - Displays the history of the selected resource. At the start of the operation, you can set filtering options.

- 📇 **Show Annotation** (**Ctrl (Meta on Mac OS) + Shift + A**) - Complex action that does the following operations:

  - opens the selected resource in the **Annotations** editor;
  - displays corresponding annotations list in the **Annotations** view;

- displays the history of the selected resource.

- **Revision Graph** (**Ctrl (Meta on Mac OS) + Shift + G**) - This action allows you to see the graphical representation of a resource history. For more details about a resource revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.

- **Copy URL Location (Ctrl (Meta on Mac OS) + Alt + U)** () - Copies to clipboard the URL location of the selected resource.

- **Copy to** - Copies to a specified location the currently selected resource(s). This action is also available when you browse other revisions than the latest one (*HEAD*), to allow restoring previous versions of an item.

- **Move to** (**Ctrl + (Meta on Mac OS) + M**) - Moves to a specified location the currently selected resource(s).

- **Rename** (**(F2)**) - Renames the selected resource.

- **Delete** (**(Delete)**) - Deletes selected items from the repository via an immediate commit.

- **New Folder** - Allows you to create a folder in the selected repository path (available only for folders).

- **Locking** (available only for files):

    - **Lock** (**Ctrl + (Meta on Mac OS) + K**) - Allows you to lock certain files for which you need exclusive access. For more details on the use of this action, see *Locking a file*.

    - **Unlock** (**Ctrl + (Meta on Mac OS) + Shift + K**) - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

- **Show SVN Properties** (**Ctrl + (Meta on Mac OS) + Shift + P**) - Brings up the *Properties view* displaying the SVN properties for the selected resource. This view does not allow adding, editing, or removing SVN properties of a repository resource. These operations are allowed only for working copy resources.

- **File Information** (**Ctrl + (Meta on Mac OS) + I**) - provides additional information for the selected resource. For more details, see the section *Information view*.

Assistant Actions

When there is no repository configured, the **Repositories** view mode lists the following two actions:
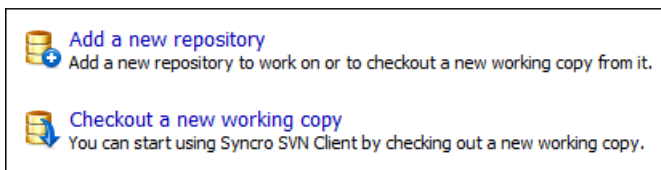
Drag and Drop Operations

The structure of the files tree can be changed with drag and drop operations inside the **Repositories** view. These operations behave in the same way with the **Copy to**/**Move to** operations.

Working Copy View

The **Working Copy** view allows you to manage the content of an SVN working copy.

The toolbar contains:

- the list of defined working copies
- a set of view modes that allow you to filter the content of the working copy based on the resource status (like incoming or outgoing changes)
- **Settings** menu

If you click any of the view modes (**All Files**, **Modified**, **Incoming**, **Outgoing**, **Conflicts**), the information displayed changes as follows:

- **All Files** - Resources (files and folders) are presented in a hierarchical structure 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 44: Working Copy View - All Files View Mode

- **Modified** - The resource tree presents resources modified locally (including those with conflicting content) and remotely. Decorator icons are used to differentiate between various resource states:

  - - incoming modification from repository:

    - - file content or properties modified remotely;

    - - new file added remotely;

    - - file deleted remotely;

  - - outgoing modification to repository:

    - - file content or properties modified locally;

    - - new file added locally;

    - - file deleted locally;

  - - pseudo-conflict state - a resource being locally and remotely modified at the same time, or a parent directory of such a resource.

  - - real conflict state - a resource that had both incoming and outgoing changes and not all the differences could be merged automatically through the update operation (manually editing the local file is necessary for resolving the conflict).

UserGuide ▾		🗔	🏗 All Files	⬌ Modified	⬅ Incoming	⮕ Outgoing	⬌ Conflicts		⚙
Name	📄	🏷	🗄	🗝	Remote date	Remote revision	Remote author	Size	Type
📁 UserGuide		●	🔷		Today 11:44 AM	12368	sorin		Folder
▲ 📁 DITA									Folder
▷ 📁 img									Folder
▷ 📁 references									Folder
▷ 📁 tasks									Folder
▲ 📁 topics									Folder
📄 ignore-resources-working-copy.dita	⭐							2 KB	dita
📄 oxygenEntitiesDictionary.dita			🔷		Today 11:44 AM	12368	sorin	2 KB	dita
📄 preferences-svn-working-copy.dita	⭐							4 KB	dita
📄 properties-view.dita	⭐							2 KB	dita
📄 revert-changes.dita	⭐							3 KB	dita
📄 show-history.dita	⭐							1 KB	dita
📄 svn-main-menu.dita	⭐							26 KB	dita
📄 svn-main-toolbar.dita	⭐							3 KB	dita
📄 svn-toolbar.dita	⭐							2 KB	dita
📄 tree-conflict.dita	⭐							1 KB	dita
📄 update-working-copy.dita	⭐							4 KB	dita
📄 views.dita	⭐							3 KB	dita
📄 working-copy-menu.dita	⭐							20 KB	dita
📄 working-copy-settings.dita			🔷		Today 11:28 AM	12366	sorin	2 KB	dita
📄 working-copy-view.dita	⭐							15 KB	dita
📄 EditorUserManual.ditamap	⭐							102 KB	ditamap
📄 .project			🔴			12368		1 KB	File
📄 build.xml		●	🔷		Nov 17, 2010	12337	sorin	14 KB	xml
📄 build_part.xml			🔷		Nov 16, 2010	12323	sorin	18 KB	xml
📄 userguide.xpr	⭐							2 MB	xpr

Figure 45: Working Copy View - Modified View Mode

- ⬅ **Incoming** - The resource tree presents only incoming changes.
- ⮕ **Outgoing** - The resource tree presents only outgoing changes.
- ⬌ **Conflicts** - The resource tree presents only conflicting changes (real conflicts, pseudo-conflicts, and files in *the Name conflict state*).

The following columns provide information about the resources:

- **Name** - Resource name. Resource icons can have the following decorator icons:
  - Additional status information:
    - 📁 **Propagated modification marker** - A folder marked with this icon indicates that the folder itself presents some changes (like modified properties) or a child resource has been modified.
    - 📁 **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 and it indicates a working copy not directly related with the parent working copy that defines it.
    - 📁 **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.
    - 👁 *Grayed* - A resource with a grayed icon but no overlaid icon is an ignored resource. It is obtained with the **Add to svn:ignore** action.
  - Current SVN depth of a folder:

- **Immediate children (immediates)** (a variant of *sparse checkout*) - The directory contains only direct file and folder children. Child folders ignore their content.

- **File children only (files)** (a variant of *sparse checkout*) - The directory contains only direct file children, disregarding any child folders.

- **This folder only (empty)** (a variant of *sparse checkout*) - The directory discards any child resource.

  Note:
  - Any folder not marked with one of the depth icons, has recursive depth (*infinity*) set by default (presents all levels of child resources).
  - Although folders not under version control can have no depth set, Syncro SVN Client presents *unversioned* and *ignored* folders with *empty* depth when **Show unversioned directories content** or **Show ignored directories content** options are disabled.

- **Local file status** - Shows the changes of working copy resources that were not committed to the repository yet. The following icons are used to mark resource status:

  - - Resource is *not under version control* (*unversioned*).
  - - Resource is being *ignored* because it is not under version control and its name matches a file name pattern defined in one of the following places:

    - *global-ignores* section in the SVN client-side *'config' file*;
    - *Application global ignores option* of Syncro SVN Client;
    - the value of a *svn:ignore property* set on the parent folder of the resource being ignored.

  - - Marks a newly created resource, *scheduled for addition* to the version control system.
  - - Marks a resource *scheduled for addition*, created by copying a resource already under version control and inheriting all its SVN history.
  - - The content of the resource has been *modified*.
  - - Resource has been *replaced* in your working copy (the file was scheduled for deletion, and then a new file with the same name was scheduled for addition in its place).
  - - Resource is *deleted*(scheduled for deletion from **Repository** upon the next commit).
  - - The resource is *incomplete* (as a result of an interrupted *check out* or *update* operation).
  - - The resource is *missing* because it was moved or deleted without using an SVN-aware application.
  - - The contents of the resource is in *real conflict state*.
  - - Resource is in *tree conflict* state after an update operation because:

    - Resource was locally modified and incoming deleted from repository;
    - Resource was locally scheduled for deletion and incoming modified.

  - - Resource is *obstructed* (versioned as one kind of object: file, directory, or symbolic link, but has been replaced outside Syncro SVN Client by a different kind of object).
  - - Resource is in *name conflict* state (only on case insensitive systems like Windows). This happens when two files having the same name (ignoring letter-cases) exist in the same repository folder (for example `file.txt` and `File.txt`). The two files were added to repository from a case-sensitive operating system like Linux or Mac OS X. This state is reported after an *update* operation and most of the Apache Subversion[TM] operations cannot be performed over such files. The solution is to rename one of the files from a case-sensitive operating system or directly from the **Repositories** view in order to have the files with completely different names.

- **Local properties status** - Marks the resources that have SVN properties, with the following possible states:

  - - The resource has SVN properties set.
  - - The resource properties have been modified.
  - - Properties for this resource are in *real conflict* with property updates received from the repository.

- **Revision** - The current revision number of the resource.
- **Date** - Date when the resource was last time modified on the disk.
- **BASE Revision** - The revision number of the pristine version of the resource.
- **BASE Date** - Date when the pristine version of the resource was last time committed in the repository.
- **Author** - Name of the person who made the last modification on the pristine version of the resource.
- ⊞ **Remote file status** - Shows changes of resources recently modified in the repository. The following icons are used to mark incoming resource status:

  - ◁ - Resource is newly added in repository.
  - ◁ - The content of the resource has been modified in repository.
  - ◁ - Resource was replaced in repository.
  - ◀ - Resource was deleted from repository.

- ◇ **Remote properties status** - Resources marked with the ○ icon have incoming modified properties from the repository.
- **Remote revision** - Revision number of the resource latest committed modification.
- **Remote date** - Date of the resource latest modification committed on the repository.
- **Remote author** - Name of the author who committed the latest modification on the repository.
- 🔒 **Lock information** - Shows the lock state of a resource. The lock mechanism is a convention intended to help you signal other users that you are working with a particular set of files. It minimizes the time and effort wasted in solving possible conflicts generated by clashing commits. A lock gives you exclusive rights over a file, only if other users follow this convention and they do not try to bypass the lock state of a file.

  A folder can be locked only by the SVN client application, completely transparent to the user, if an operation in progress was interrupted unexpectedly. As a result, folders affected by the operation are marked with the 🔒 symbol. To clear the locked state of a folder, use the **Clean up** action.

  > 📄 **Note:** Users can lock only files.

  The following lock states are displayed:

  - *no lock* - the file is not locked. This is the default state of a file in the SVN repository;
  - *remotely locked* (🔒) - shown when:

    - another user has locked the file in the repository;
    - the file was locked by the same user from another working copy;
    - the file was locked from the **Repositories** view.

    If you try to commit a new revision of the file to the repository, the server does not allow you to bypass the file lock.

    > 📄 **Note:** To commit a new revision, you need to wait for the file to be unlocked. Ultimately, you might try to *break* or *steal* the lock, but this is not what other users expect. Use these actions carefully, especially when you are not the file lock owner.

  - *locked* (🔒) - displayed after you have locked a file from the current working copy. Now you have exclusive rights over the corresponding file, being the only one who can commit changes to the file in the repository.

    > 📄 **Note:** Working copies keep track of their locked files, so the locks are presented between different sessions of the application. Synchronize your working copy with the repository to make sure that the locks are still valid (not *stolen* or *broken*).

  - *stolen* (🔒) - a file already locked from your working copy is being locked by another user. Now the owner of the file lock is the user who stole the lock from you.
  - *broken* (🔒) - a file already locked from your working copy is no longer locked in the repository (it was unlocked by another user).

> **Note:** To remove the *stolen* or *broken* states from your working copy files, you have to **Update** them.

If one of your working copy files is locked, hover the mouse pointer over the lock icon to see more information:

- lock type - current file lock state;
- owner - the name of the user who created the lock;
- date - the date when the user locked the file;
- expires on - date when the lock expires. Lock expiry policy is set in the repository options, on the server side;
- comment - the message attached when the file was locked.

- **Size** - Resource size on disk;
- **Type** - Contains the resource type or file extension.

> **Note:** The working copy table allows you to show or hide any of its columns and also to sort its contents by any of the displayed columns. The table header provides a contextual menu which allows you to customize the displayed information.

The toolbar allows you to switch between two working copies:

- Drop down list - Contains all the working copies Syncro SVN Client is aware of. When you select another working copy from the list, the newly selected working copy content is scanned and displayed in the **Working Copy** view.

- (on Mac OS X) **Working Copies Manager** - opens a dialog box that displays the working copies Syncro SVN Client is aware of. In this dialog box, you can add existing working copies or remove those you no longer need. If you try to add a folder which is not a valid Subversion working copy, Syncro SVN Client warns you that the selected directory is not under version control.

> **Note:** Removing a working copy from this dialog does NOT remove it from your file system; you will have to do that manually.

Working Copy Settings

The **Settings** button from the toolbar of the **Working Copy** view provides the following options:

- **Show unversioned directories content** - displays the content of unversioned directories;

> **Note:** In case this option is disabled, it will be ignored for items that, after a synchronize, are reported as incoming from the repository. This applies for all working copy modes, except **All Files**.

- **Show ignored items** - displays the ignored resource when **All Files** mode is selected;
- **Show ignored directories content** - displays the content of ignored directories when **All Files** mode is selected;

> **Note:** Although *ignored* items are not presented in the **Modified**, **Incoming**, and **Conflicts** modes, they will be if, after a synchronize, they are reported as incoming from the repository.

- **Show deleted items** - displays the deleted resource when **All Files** mode is selected. All other modes always display deleted resources, disregarding this option;
- **Tree** / **Compressed** / **Flat** - affect the way information is displayed inside the **Modified**, **Incoming**, **Outgoing**, and **Conflicts** view modes;
- **Configure columns** - allows you to customize the structure of the **Working Copy** view data. This action opens the following dialog box:
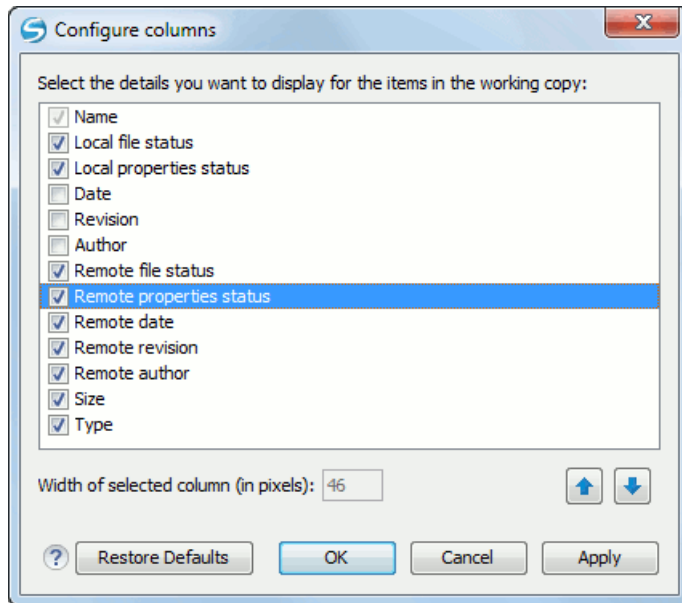
Figure 46: Configure Columns of Working Copy View

The order of the columns can be changed with the two arrow buttons. The column size can be edited in the **Width of selected column** field. The **Restore Defaults** button reverts all columns to the default order, width and enabled/disabled state from the installation of the application.

Working Copy Format

When an SVN working copy is loaded, Syncro SVN Client first checks the format of the working copy. If the format is older than SVN 1.7, a dialog box asks you whether you want to upgrade to the latest known working copy format - that is SVN 1.7.

In case you select the **Never ask me again** check box, your choice is used as default for similar situations. You can change how working copy formats are handled from **Options** > **Preferences** > **SVN** > **Working copy**, in the *Administrative area* section.

> **Note:**
> - The format of the working copy can be downgraded or upgraded at any time with the **Upgrade** and **Downgrade** actions available in the **Tools** menu. These actions allow switching between SVN 1.4, SVN 1.5 and SVN 1.6 working copy formats.
>
> - SVN 1.7 working copies cannot be downgraded to older formats.

Refresh a Working Copy

A refresh is a frequent operation triggered automatically when you switch between two working copies using the toolbar selector of the **Working Copy** view and when you switch between Syncro SVN Client and other applications.

The **Working Copy** view features a fast refresh mechanism: the content is cached locally when loading the working copy for the first time. Later on, when the same working copy is displayed again, the application uses this cache to detect the changes between the cached content and the current content found on disk. The refresh operation is run on these changes only, thus improving the response time. improvement is noticeable especially when working with large working copies.

Contextual Menu Actions

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

- **Edit conflict (Ctrl (Meta on Mac OS) + E)** - Opens the **Compare** editor, allowing you to modify the content of the currently conflicting resources. For more information on editing conflicts, see *Edit conflicts*.

- **Open in Compare Editor (Ctrl (Meta on Mac OS) + Alt + C)** - Displays changes made in the currently selected file.

- **Open (Ctrl (Meta on Mac OS) + O)** - Opens the selected resource from the working copy. Files are opened with an internal editor or an external application associated with that file type, while folders are opened with the default file system browsing application (e.g. Windows Explorer on Windows, Finder on Mac OS X, etc).

- **Open with (Ctrl (Meta on Mac OS) + Shift + O)** - Displays the *Open with...* dialog to specify the editor in which the selected file will be opened. If multiple files are selected, only external applications can be used to open the files.

- **Show in Explorer**/**Show in Finder** - Opens the parent directory of the selected working copy file and selects the file.

- ⊞ **Expand all (Ctrl (Meta on Mac OS) + Alt + X)** - Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.

- ⟳ **Refresh (F5)** - Re-scans the selected resources recursively and refreshes their status in the working copy view.

- ⟳ **Synchronize (Ctrl (Meta on Mac OS) + Shift + S)** - Connects to the repository and determines the working copy and repository changes made to the selected resources. The application switches to **Modified** view mode if the *Always switch to 'Modified' mode* option is selected.

- **Update (Ctrl (Meta on Mac OS)+ U)** - Updates the selected resources to the *HEAD* revision (latest modifications) from the repository. If the selection contains a directory, it will be updated depending on its *depth*.

- **Update to revision/depth** - 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.

- **Commit** - Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.

- ⤶ **Revert (Ctrl (Meta on Mac OS) + Shift + V)** - Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Apache Subversion^TM pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.

- **Override and Update** - Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.

- **Override and Commit** - Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.

- ✓ **Mark Resolved (Ctrl (Meta on Mac OS) + Shift + R)** - Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.

- ✓ **Mark as Merged (Ctrl (Meta on Mac OS) + Shift + M)** - Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.

- ✖ **Create patch (Ctrl (Meta on Mac OS) + Alt + P)** - Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see *the section about patches*.

- **Compare with**:

  - **Latest from HEAD (Ctrl (Meta on Mac OS) + Alt + H)** - Performs a 3-way diff operation between the selected file and the *HEAD* revision from the repository and displays 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 (Ctrl (Meta on Mac OS) + Alt + C)** - Compares the working copy file with the BASE revision file (the so-called *pristine copy*).

- **Revision (Ctrl (Meta on Mac OS) + Alt + R)** - Shows the **History view** containing the log history of that resource.

- **Branch/Tag** - Compares 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).

- **Each other** - Compares two selected files with each other.

These *compare* actions are enabled only if the selected resource is a file.

- **Replace with**:

  - **Latest from HEAD** - Replaces the selected resources with their versions from the *HEAD* revision of the repository.

  - **BASE revision** - Replace the selected resources with their versions from the pristine copy (the BASE revision).

  - **Note:** In some cases it is impossible to replace the current selected resources with their versions from the *BASE/HEAD* revision:

    - for **Replace with BASE revision** action, the resources being unversioned or added have no *BASE* revision, so they cannot be replaced. However, they will be deleted if the action is invoked on a parent folder. The action will never work for missing folders or for obstructing files (folders being obstructed by a file), because you cannot recover a tree of folders;

    - for **Replace with latest from HEAD** action, you must be aware that there are cases when resources will be completely deleted or reverted to BASE revision and after that updated to HEAD revision, in order to avoid conflicts. These cases are:

      - the resource is *unversioned*, *added*, *obstructed* or *modified*;
      - the resource is affected by a *svn:ignore* or *svn:externals* property which is locally added on the parent folder and not yet committed to the repository.

- **Show History (Ctrl (Meta on Mac OS) + H)** - Displays the **History view** where the log history for the selected resource will be presented. For more details about resource history see the sections about *the resource history view* and *requesting the history for a resource*.

- **Show Annotation (Ctrl (Meta on Mac OS) + Shift + A)** - 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 *Annotations View*.

- **Revision Graph (Ctrl (Meta on Mac OS) + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see *Revision Graph*.

- **Copy URL Location (Ctrl (Meta on Mac OS) + Alt + U)** - Copies the encoded URL of the selected resource from the Working Copy to the clipboard.

- **Mark as copied** - You can use this action to mark an item from the working copy as a copy of an other item under version control, when the copy operation was performed outside of an SVN client. The **Mark as copied** action is available when you select two items (both the new one and the source one) and depends on the state of the source item.

  - **Note:** If you use an SVN 1.6 working copy or older, this action does not apply for directories.

- **Mark as moved** - You can use this action to mark an item from the working copy as being moved from another location of the working copy, when the move operation was performed outside of an SVN client. The **Mark as**

moved action is available when you select two items from different locations (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

> **Note:** If you use an SVN 1.6 working copy or older, this action does not apply for directories.

- **Mark as renamed** - You can use this action to mark an item from the working copy as being renamed outside of an SVN client. The **Mark as renamed** action is available when you select two items from the same directory (both the new one and the source one (usually reported as *missing*)) and depends on the state of the source item.

  > **Note:** If you use an SVN 1.6 working copy or older, this action does not apply for directories.

- **Copy to** - Copies to a specified location the currently selected resource(s).

- **Move to (Ctrl (Meta on Mac OS)+ M)** - Moves to a specified location the currently selected resource(s).

- **Rename (F2)** - You can only rename one 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.

- **Delete (Delete)** - Schedules selected items for deletion upon the next commit, removing them from the disk. Depending on the state of each item, you are prompted to confirm the operation.

- **New**:

  - **New File** - Creates a new file inside the selected folder. The newly created file will be added under version control only if the parent folder is already versioned.

  - **New Folder (Ctrl (Meta on Mac OS)+ Shift + F)** - Creates a child folder inside the selected folder. The newly created folder will be added under version control only if its parent is already versioned.

  - **New External Folder (Ctrl (Meta on Mac OS) + Shift + W)** - Creates a new folder inside the selected folder, having the contents of a target folder from the current working copy's repository. The application does this by setting a *svn:externals* property on the selected folder and updating the folder in order to bring all the newly pointed resources inside your current working copy. The operation shows a dialog which allows you to specify the new folder's name and easily select the target folder by browsing the repository's contents.

    Subversion 1.5 and higher clients 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.

- **Add to version control (Ctrl (Meta on Mac OS) + Alt + V)** - Allows you to schedule for addition resources that are not under version control. For further details, see *Add Resources to Version Control* section.

- **Remove from version control** - Schedules selected items for deletion from repository upon the next commit. The items are not removed from the file system after committing.

- **Add to "svn:ignore" (Ctrl (Meta on Mac OS) + Alt + I)** - Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the *svn:ignore* property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.

- **Clean up (Ctrl (Meta on Mac OS) + Shift + C)** - 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.

- **Locking**:

  - **Scan for locks (Ctrl (Meta on Mac OS) + L)** - 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 *Scanning for locks*.

  - 🔒 **Lock (Ctrl (Meta on Mac OS) + K)** - 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 see *Locking a file*.

  - 🔓 **Unlock (Ctrl (Meta on Mac OS) + Alt + K)** - Releases the exclusive access to a file from the repository. You can also choose to unlock it by force (*break the lock*).

- 🏷️ **Show SVN Properties (Ctrl (Meta on Mac OS) + P)** - Brings up the *Properties view* and displays the SVN properties for the selected resource.

- ℹ️ **File Information (Ctrl (Meta on Mac OS) + I)** - Provides additional information for the selected resource from the working copy. For more details please see *Obtain information for a resource*.

Drag and Drop Operations

The structure of the files tree can be changed with drag and drop operations inside the **Working Copy** view. These operations behave in the same way with the **Copy to**/**Move to** operations.

Also, files and folders can be added to the file tree of the view as *unversioned* resources by drag and drop operations from other applications (for example from Windows Explorer or Mac OS X Finder). In this case, the items from the file system are only copied, without removing them from their original location.

⚠️ **Attention:** When you drag items from the working copy to a different application, the performed operation is controlled by that application. This means that the moved items are left as *missing* in the working copy (items are moved in the file system only, but no SVN versioning meta-data is changed).

Assistant Actions

To ensure a continuous and productive work flow, when a view mode has no files to present, it offers a set of guiding actions with some possible paths to follow.

Initially, when there is no working copy configured the **All Files** view mode lists the following two actions:

Check out a new working copy
You can start using Syncro SVN Client by checking out a new working copy.

Add a working copy
If you already have a working copy on disk, you can add it to Syncro SVN Client and start to work.
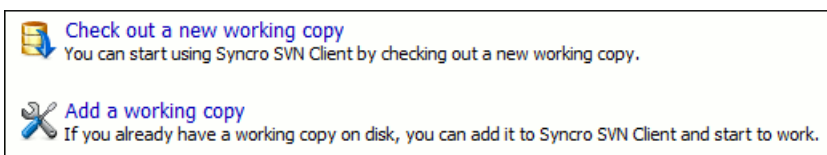
Figure 47: All Files Panel

For **Modified**, **Incoming**, **Outgoing**, **Conflicts** view modes, the following actions may be available, depending on the current working copy state in different contexts:

- ℹ️ Information message - Informs you why there are no resources presented in the currently selected view mode;

- 🔄 **Synchronize with Repository** - Available only when there is nothing to present in the **Modified** and **Incoming** view modes;

- ⬅️ **Switch to Incoming** - Selects the **Incoming** view mode.

- ➡️ **Switch to Outgoing** - Selects the **Outgoing** view mode.

- 🔻 **Switch to Conflicts** - Selects the **Conflicts** view mode.

- **Show all changes/incoming/outgoing/conflicts** - Depending on the currently selected view mode, this action presents the corresponding resources after a synchronize operation was executed only on a part of the working copy resources.

History View

In Apache Subversion^{TM}, 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 *Repositories view*, *Working Copy view*, *Revision Graph*, or *Directory Change Set view*. From the **Working copy view** you can display the history of local versioned resources.

The view consists of four distinct areas:

- The table showing details about each revision, like: revision number, commit date and time, number of changes (more details available in the tooltip), author's name, and a fragment of the commit message.

  Some revisions may be highlighted to emphasize:

  - the current revision of the resource for which the history is displayed - a bold font revision;
  - the last revision in which the content or properties of the resource were modified - blue font revision.

    Note: Both font highlights may be applied for the same revision.

- The complete commit message for the selected revision;

- A tree structure showing the folders where the modified resources are located. You can compress this structure to a more compact form that focuses on the folders that contain the actual modifications;

- The list of resources modified in the selected revision. For each resource, the type of action done against it is marked with one of the following symbols:

  - ⊞ - A newly created resource;
  - ⊞ - A newly created resource, copied from another repository location;
  - ⊞ - The content/properties of the resource were *modified*;
  - ℝ - Resource was *replaced* in the repository;
  - ⊟ - Resource was deleted from the repository.
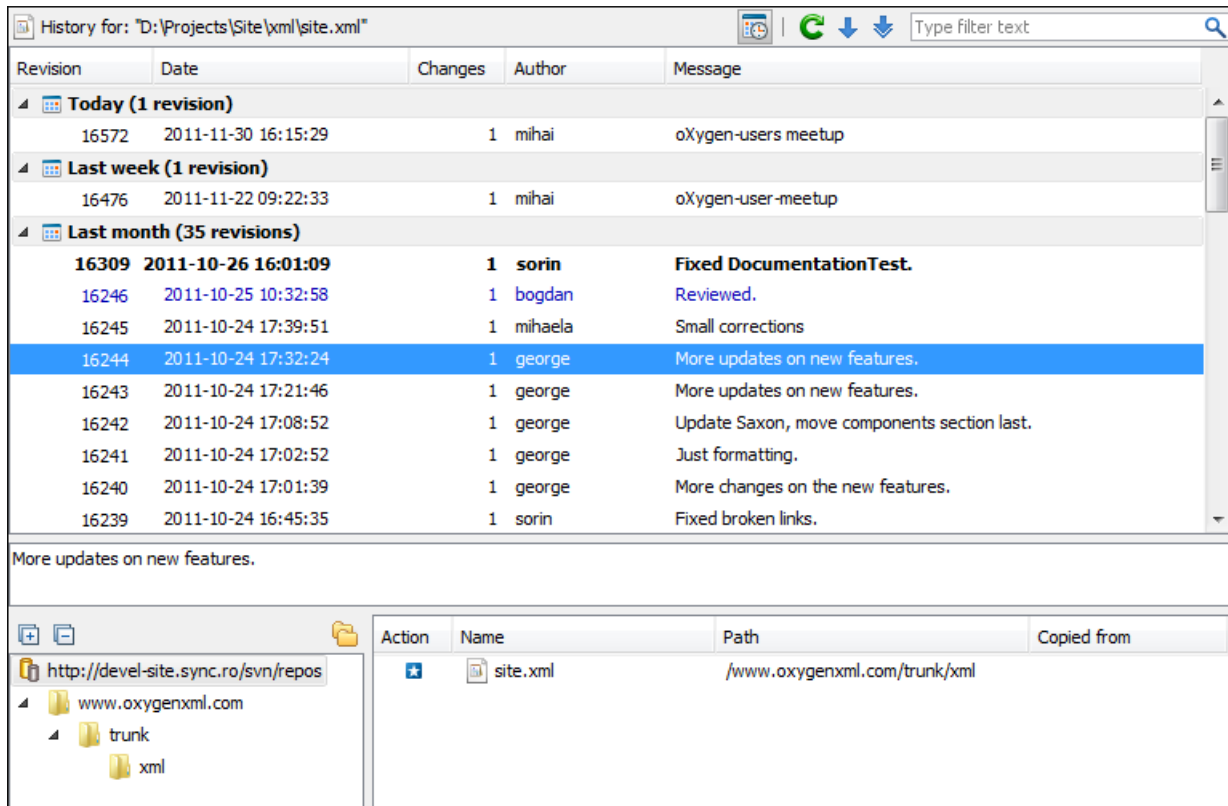
Figure 48: History View

You can group revisions in predefined time frames (today, yesterday, this week, this month), by pressing the ▣ **Group by date** button from the toolbar.

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.
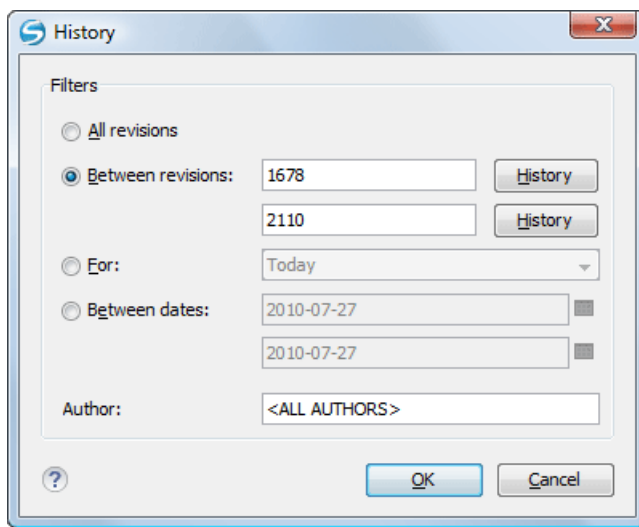
Figure 49: History Filters Dialog

Options for the set of revisions presented in the History view are:

• all revisions of the selected resource;

• only revisions between a start revision number and an end revision number;

• only revisions added in a period of time like today, last week, last month, etc.;

• only revisions between a start and an end date;

• only 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**.

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 are 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:

• **Compare with working copy** - Compares the selected revision with your working copy file. It is enabled only when you select a file.

• **Open** - Opens the selected revision of the file into the Editor. This is enabled only for files.

• **Open with** - Displays the *Open with...* dialog to specify the editor in which the selected file will be opened.

• **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.

• **Save as** - Allows you to save the contents of a file as it was committed at a certain revision. This option is available only when you access the history of a file.

• **Copy to** - Copies to the repository the item whose history is displayed, using the selected revision. This option is active only when presenting the history for a repository item (URL).

  📄 **Note:** This action can be used to resurrect deleted items also.

• **Revert changes from this revision** - Reverts changes which were made in the selected revision. The changes are reverted only in your working copy file, so it does not affect the repository file. It does not replace your working copy file with the entire file at the earlier revision, but only rolls-back earlier changes when other unrelated changes have been made since the date of the revision. This action is enabled when the resource history was launched for a local working copy resource.

• **Update to revision** - Updates your working copy resource to the selected revision. This is useful if you want your working copy to 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 is inconsistent and you are unable to commit your changes.

• 🔽 **Check out** - Checks out a new working copy of the directory for which the history is presented, from the selected revision.

• 🔳 **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** - Allows you to change commit data for a file:
    - *Author* - Changes the name of the SVN user that committed the selected revision.
    - *Message* - Changes the commit message of the selected revision.

When two resources are selected in the **History** view, the contextual menu contains the following 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.

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*

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. The changed resources can be contained in the folder or in a subfolder of that folder. These resources are presented in a tree format. For each changed resource all the revisions committed between the two reference revision numbers are presented.
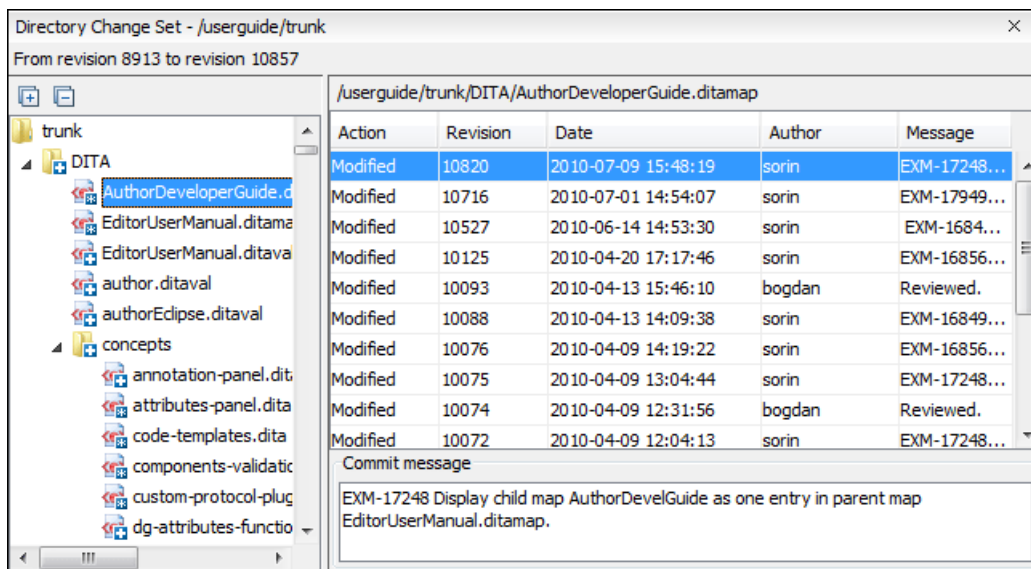
Figure 50: Directory Change Set View

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 left side tree. These revisions 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. 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:

- **Compare with previous version** - Performs a diff operation between the selected revision in the list and the previous one.
- **Open** - Opens the selected revision in the associated editor type.
- **Open with** - *Displays a dialog* with the available editor types and allows the user to select the editor type for opening the selected revision.
- **Save as** - Saves the selected file as it was in the selected revision.
- **Copy to** - Copies to the repository the item whose history is displayed, using the selected revision.

  > **Note:** This action can be used to resurrect deleted items also.

- **Check out** - Checks out a new working copy of the selected directory, from the selected revision.
- **Show Annotation** - Requests the annotations of the file and *displays them in the **Annotations** view*.

The Editor Panel of SVN Client

You can open a file for editing in:

- an internal built-in editor;

- the default external application associated in the operating system with the file's type;

- 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 **Open with** action available in *the Repository view*, *the Working Copy view* and *the History view* which opens the following dialog:
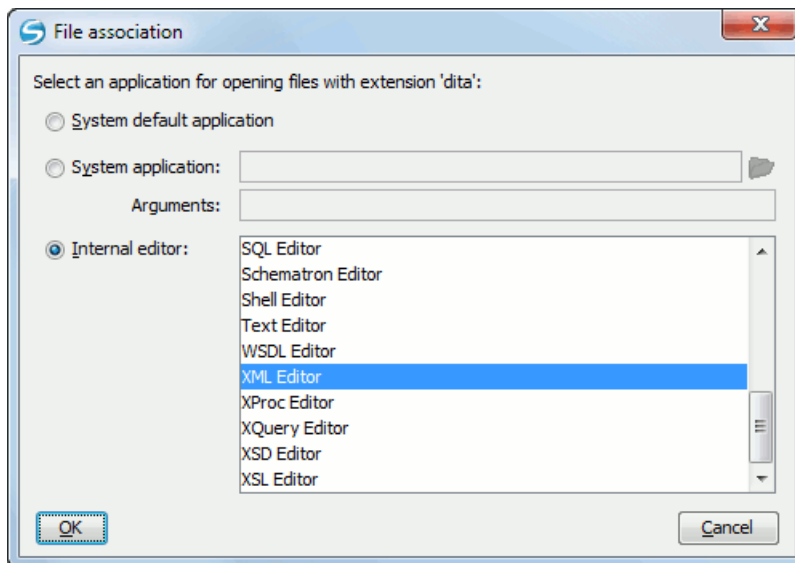
Figure 51: Open with Dialog

The internal editor can be accessed either from the *Working copy view* or from the *History view*. No actions that modify the content are allowed when the editor is opened with a revision from history.

Only one file at a time can be edited in an internal editor. If you try to open another file it will be opened in the same editor window. The editor provides syntax highlighting for known file types. This means that a different color will be used for each recognized token type found in the file. If the file's content type 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 (Meta on Mac OS) + S** key shortcut. After saving your file you can see the file changed status in *the Working Copy 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 file's encoding.

Annotations View

Sometimes you need to know not only what was changed in a file, but also who made those changes. 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 52: The Annotations View

The annotations of a file are computed with the **Show Annotation** action 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*.

Compare View

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**.
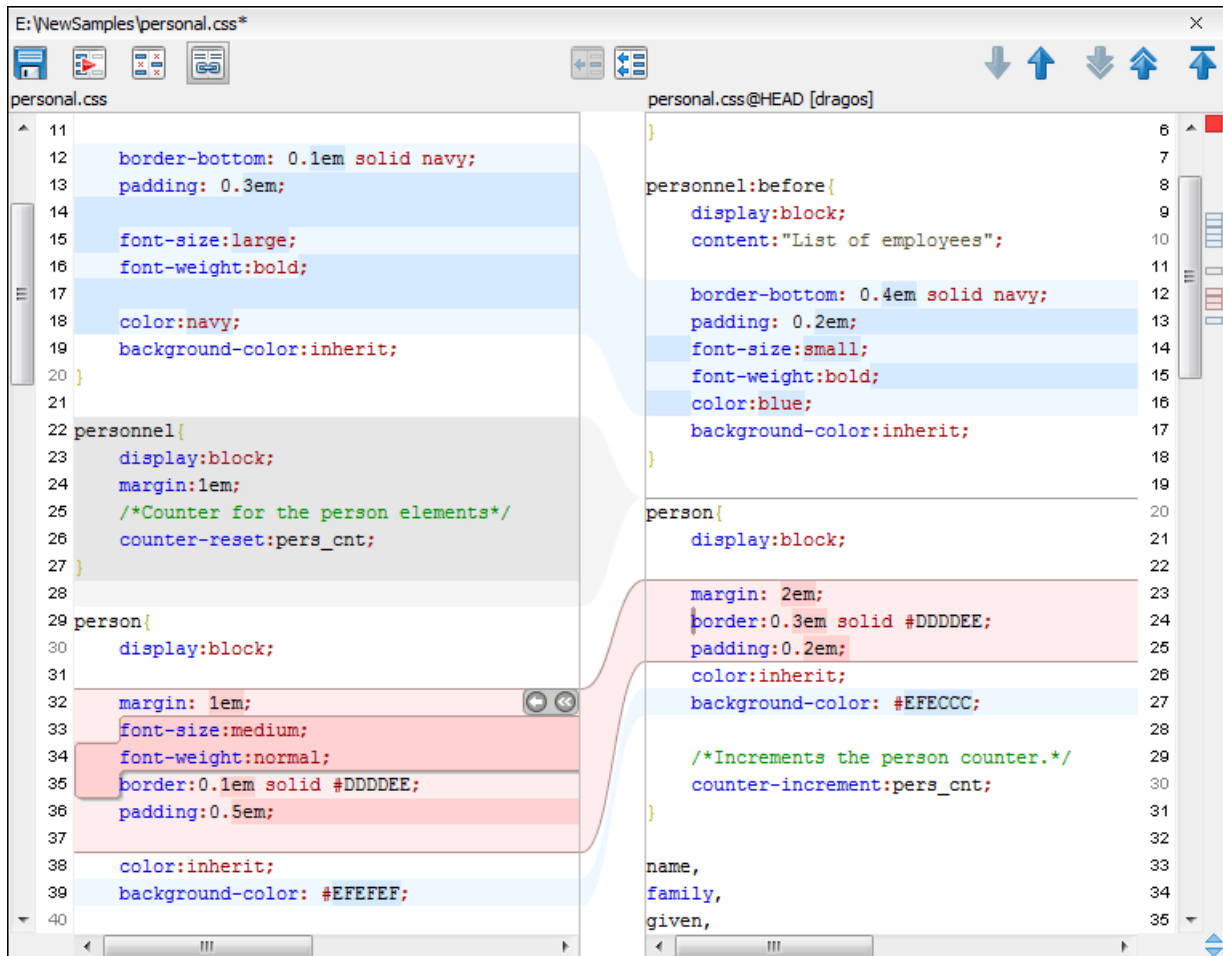
Figure 53: Compare View

At the top of each of the two editors, there are presented the name of the opened file, the corresponding SVN revision number (for remote resources) and the author who committed the associated revision.

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.

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 see *View differences*.

Toolbar

The list of actions available in the toolbar consists of:

- 🖫 **Save action** - Saves the content of the left editor when it can be edited.

- 🖹 **Perform Files Differencing** - performs a comparison between the source and target files;

- **Ignore Whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.

- **Synchronized scrolling** - Synchronizes scrolling of the two open files, so that a selected difference can be seen on both sides of the application window. This action enables/disables the previous described behavior.

- **Next Block of Changes** - jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

- **Previous Block of Changes** - jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

- **Next Change** - jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change;

- **Previous Change** - jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change;

- **First Change** - jumps to the first change from the current file;

- **Copy change from right to left** - Copies the selected change from the right editor to the left editor.

- **Copy all non-conflicting changes from right to left** - 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.

These actions are available also from the *Compare* menu.

Image Preview

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

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).

Compare Images View

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

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

Properties View

The properties view presents Apache Subversion[TM] properties for the currently selected resource from either the **Working Copy** view or the **Repositories** view.
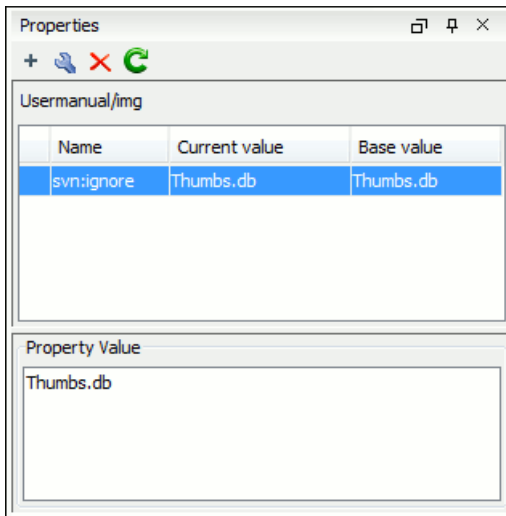
Figure 54: The Properties View

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 sign) - new property;

  - -(minus sign) - 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 external file 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.

  ⚠ **Attention:**

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 check out 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

The **Console View** shows the traces of all the actions performed by the application. Part of the displayed messages mirror the communication between the application and the Apache Subversion[TM] server. 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**.

The view has a simple layout, with most of its space occupied by a message area. On its right side, there is a toolbar holding the following buttons:

- ✸ **Clear** - Erases all the displayed messages;
- 🔒 **Lock scroll** - Disables the automatic scrolling when new messages are appended in the view.

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

Dynamic Help View

Dynamic Help view is a help window that changes its content to display the help section referring to the currently selected view. As you change the focused view, you are 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 very easy to follow.

The graphical representation of a resource history is invoked with the 🔴 **Revision graph** action available on the right click menu of a SVN resource in *the Working Copy view* and *the Repository view*.
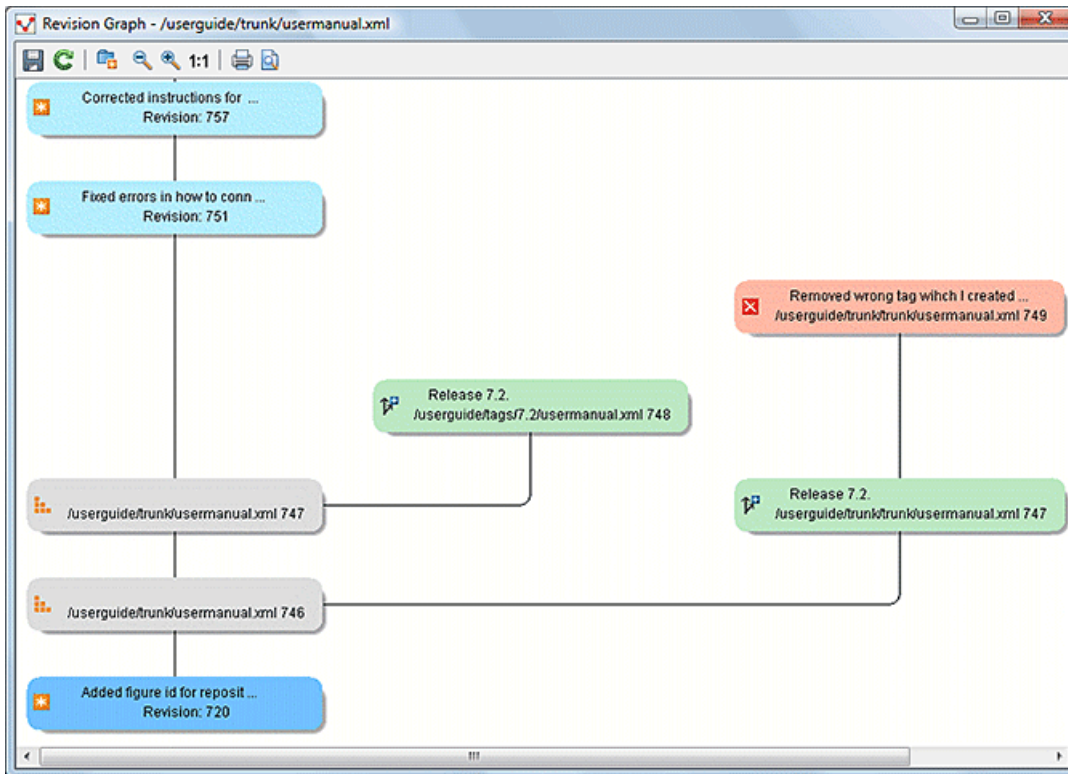
Figure 55: The Revision Graph of a File Resource

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 *Modification type* 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.
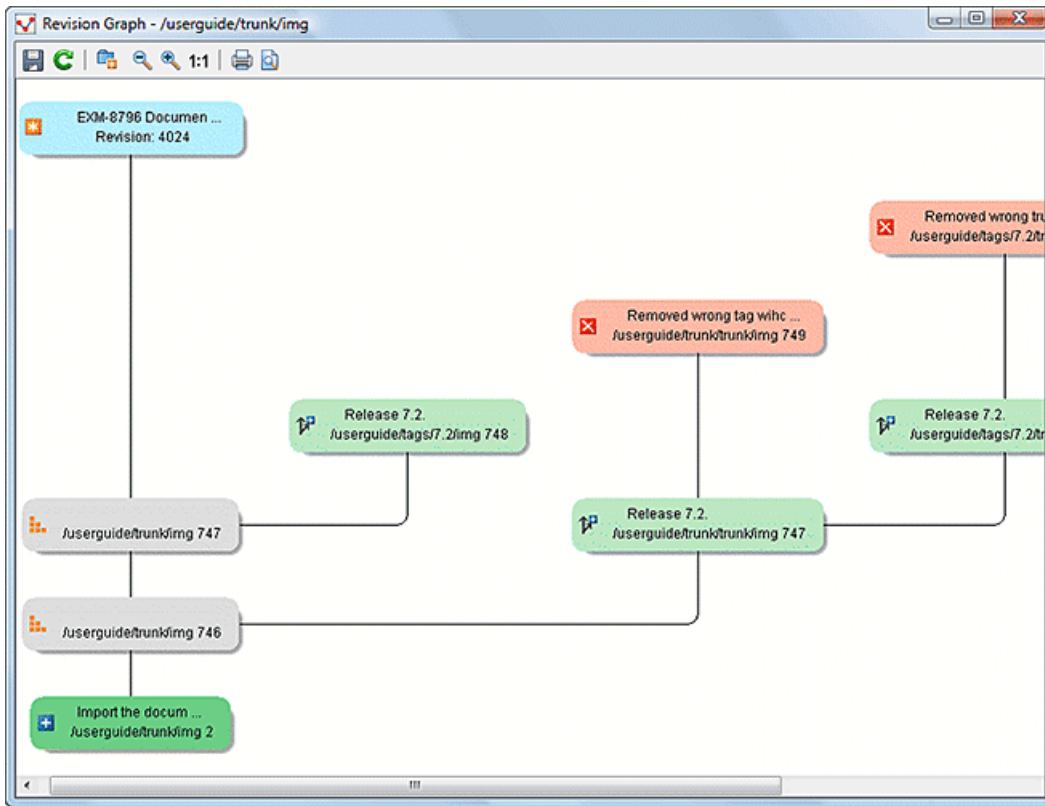
Figure 56: The Revision Graph of a Directory (Direct Changes)
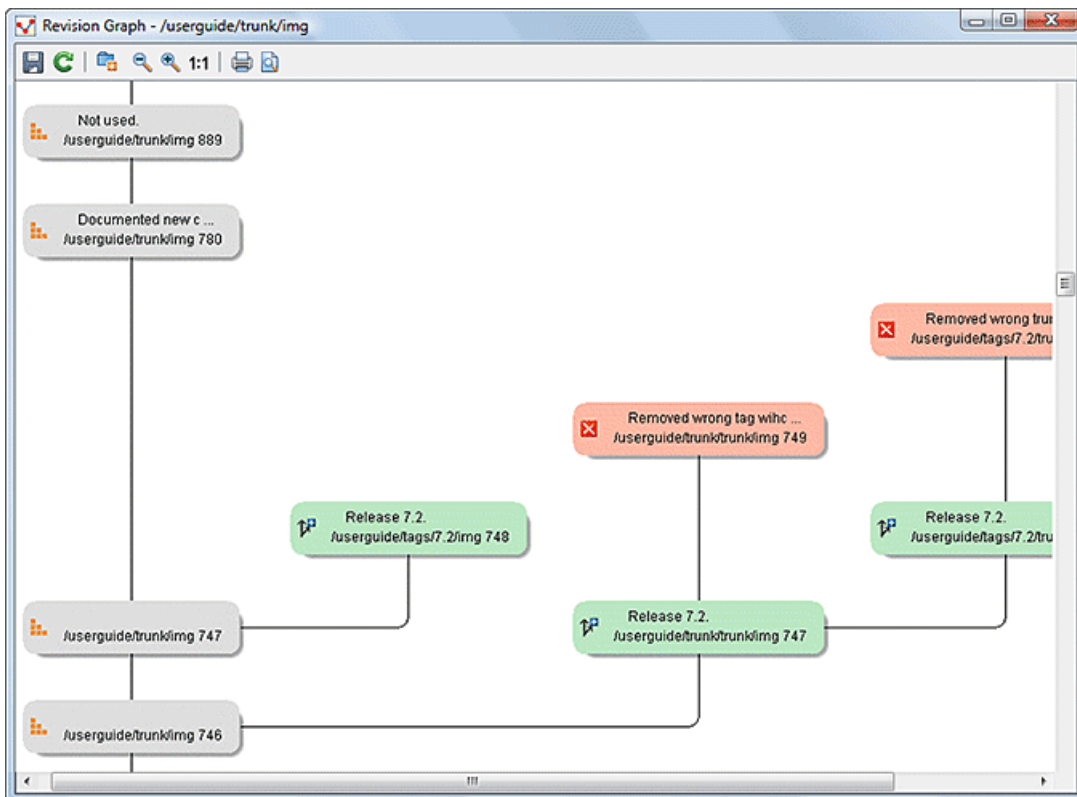
Figure 57: The Revision Graph of a Directory (Also Indirect Changes)

The **Revision graph** dialog toolbar contains the following actions:

- 🖫 **Save as image** - Saves the graphical representation as image. For a large revision graph you have to *set more memory in the startup script*. The default memory size is not enough when there are more than 100 revisions that are included in the graph.
- 🗔 **Show/Hide indirect modifications** - Switches between simplified and complete graph.
- 🔍 **Zoom In** - Zooms in the graph.
- 🔍 **Zoom Out** - Zooms out the graph. When the font reaches its minimum size, the graph nodes will display only the icons, leading to a very compact representation of the graph.
- 1:1 **Reset scale** - Resets the graphical scale to a default setting.
- 🖨 **Print** - Prints the graph.
- 🖺 **Print preview** - Offers a preview of the graph to allow you to check the information to be printed.

Right clicking any of the graph nodes display a contextual menu containing the following actions:

- **Open** - Opens the selected revision in the editor panel. Available only for files.
- **Open with** - Opens the selected revision in the editor panel. Available only for files.
- **Save as** - Saves the file for which the revision graph was generated, based on the selected node revision.
- **Copy to** - Copies to the repository the item whose revision graph is displayed, using the selected revision.

    📄 **Note:** This action can be used to resurrect deleted items also.

- **Compare with HEAD** - Compares the selected revision with the HEAD revision and displays the result in the diff panel. Available only for files.
- 🕘 **Show History** - Displays the history of the resource in *the History view*. Available for both files and directories.
- 🔽 **Check out** - *Checks out* the selected revision of the directory. Available only for directories.

When two nodes are selected in the revision graph of a file the right click menu of this selection contains only the **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 **Compare** action 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.

⚠ **Attention:**

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 the operation much faster.

Command Line Reference

This section specifies the equivalent Apache Subversion[TM] commands for each action available in the graphical user interface of Syncro SVN Client.

Checkout Command

Used to pull a SVN tree from the server to the local file system. The syntax of checkout command is the following.
`svn checkout` --revision rev *URL PATH*

rev	The desired revision number (optional)	
URL		Repository URL you want to check out from.
PATH		Checkout target on file system.

Update Command

Brings changes from the repository into your working copy. The syntax of update command is the following.

```
svn update --revision rev PATH
```

rev	The desired revision number (optional)	
PATH		Checkout target on file system.

Updates resources to the last revision on which they were synchronized or to the *HEAD* revision, if no repository information is available.

Commit Command

Sends changes from your working copy to the repository.
```
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		Location on the file system of the resource to commit. Can be more than one.

Diff Command

Displays the differences found between two revisions.
```
svn diff --revision rev1:rev2 PATH
```

rev1:rev2	Specifies the revisions to be compared.	
PATH		Location on the file system of the resource to be compared.

If you use the **Compare with latest from HEAD** (**Ctrl (Meta on Mac OS) + Alt + H**) 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. 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

Display commit log messages.
```
svn log --revision rev1:rev2 --limit N  --verbose PATH
```

rev1:rev2	Specifies the range of revisions for which to obtain the log.	
--limit N		Limits the number of messages brought to N.
--verbose		Gives detailed information about this command's execution.

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

Refresh

Print the status of working copy files and directories.
```
svn status --verbose PATH
```

--verbose	Specifies that the status of all files should be reported.
PATH	Location on the file system of the resource to get status for.

Synchronize

```
svn status --show-updates PATH
```

--show-updates	Gets the resource status by contacting the repository.
PATH	Location on the file system of the resource to get status for.

Import

Commits an unversioned file or tree into the repository.
```
svn import -m "log message" PATH URL
```

-m "log message"	Specifies the commit log message.	
PATH		Local path to the resource on the file system.
URL	URL on the repository where the resource will be imported.	

Export

Exports a directory tree.
```
svn export --revision rev URL PATH
```

rev	Specifies the desired revision(if necessary).	
URL		Repository URL you want to export from.
PATH		Location on the file system where to export.

Information

Displays information about a local or remote item.
```
svn info --revision rev PATH|URL
```

rev	Specifies the revision number for which the information will be requested.	
PATH		Local file system path to the resource.
URL		Repository URL for the resource.

Add

Add files, directories, or symbolic links.
```
svn add PATH...
```

PATH	Local file system path of 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	Predefined property name for ignoring resources.	
PATH		Relative path from the working copy root for the resource to be ignored.

PARENTPATH		Path to the parent of the resource to be ignored.

Delete

Deletes resources from a working copy or from an Apache Subversion[TM] repository.
`svn delete --recursive` *PATH|URL*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Local file system path of the resource to delete.
PARENTPATH		Repository URL of the resource to delete.

Copy

Copy a file or directory in a working copy or in the repository.
`svn copy`*(SRCPATH DSTPATH)|(SRCURL DSTURL)*

SRCPATH	Working copy path of the resource to be copied.	
DSTPATH		Working copy path where the resource will be copied to.
SRCURL	Repository path of the resource to be copied.	
DSTURL	Repository path where the resource will be copied to.	

Move / Rename

Move a file or directory.
`svn move`*(SRCPATH DSTPATH)|(SRCURL DSTURL)*

SRCPATH	Working copy path of the resource to be moved.	
DSTPATH		Working copy path where the resource will be moved to.
SRCURL	Repository path of the resource to be moved.	
DSTURL	Repository path where the resource will be moved to.	

Mark resolved

`svn resolved --recursive` *PATH*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Path to the resource in the local working copy.

Revert

Undo all local edits.
`svn revert [--recursive]` *PATH*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Local working copy path to revert to.

Cleanup

Recursively cleans up the working copy.

`svn cleanup` *PATH*

PATH	Local working copy path to clean up.

Show / Refresh Properties

`svn proplist` *PATH*
`svn propget` *PROPNAME PATH*

PATH	Local path of the resource.
PROPNAME	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*
`svn copy -m "log message"` *URL1@rev1 URL2*
`svn copy -m "log message"` *PATH URL*

-m "log message"	Commit message.	
URL1		Source repository URL.
rev1	Revision of the source.	
URL2	Destination repository URL.	
PATH	Source working copy path.	
URL	Destination repository URL.	

Merge

Apply the differences between two sources to a working copy path.

`svn merge [--dry-run]` *rev1:rev2 URL PATH*
`svn merge [--dry-run]` *URL1@rev1 URL2@rev2 PATH*

--dry-run	Specifies that the operation will be simulated without making any modifications.	
URL		Repository URL for the resource to merge.
URL1	Repository URL for the start branch to merge.	
rev1	Start revision for the resource to merge.	
URL2	Repository URL for the end branch to merge.	
rev2	End revision for the resource to merge.	
PATH	Destination path in the working copy for the result of the merge.	

Scan for locks

Obtains the repository status for all the resources in the path.

`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.

Lock

Lock working copy paths or URLs in the repository so that no other user can commit changes to them.
`svn lock [--force] [-m "log message"]` *PATH*

--force	Forces(steals) the lock.	
-m "log message"		Lock message.
PATH		Path to the file to be locked.

Unlock

Unlock working copy paths or URLs.
`svn unlock [--force]` *PATH*

--force	Forces(breaks) the lock.	
PATH		Path to the file from the working copy..

Mark as merged

`rename FILE FILE.TMP`
`svn update` *FILE*
`rename FILE.TMP FILE`

FILE	File to be marked as merged.	
FILE.TMP		Temporary filename.

Override and update

`svn revert` *PATH*
`svn update` *PATH*

PATH	Path of the resource to be overridden.

Override and Commit

If the resource is in conflict first you should perform a *Mark Resolved action*. If the resource has incoming changes you should perform a *Mark as Merged action* followed by a *Commit action*.

Add / Edit property

`svn propset [--recursive]` *PROPNAME PROPVALUE PATH*

--recursive	Specifies that the property should be set recursively.	
PROPNAME		Property name.
PROPVALUE	Property value.	
PATH		Resource's path.

Remove property

Removes a property from an item.

svn propdel [--recursive] *PROPNAME PATH*

--recursive	Specifies that the property should be deleted recursively.	
PROPNAME		Property name.
PATH		Resource's path.

Revert changes from this revision

svn merge rev:rev-1 *URL*

rev	Revision whose changes must be reverted.	
URL		The SVN URL corresponding to the resource.

Revert changes from these revisions

Short reference description.

svn merge rev1:rev2 *URL*

rev1	First revision number.	
rev2	Second revision number.	
URL		The SVN URL corresponding to the resource.

Technical Issues

This section contains special technical issues found during the use of Syncro SVN Client.

Authentication Certificates Not Saved

If Syncro SVN Client prompts you to enter the authentication certificate, although you already provided it in a previous session, then you should make sure that your local machine user account has the necessary rights to store certificate files in the *Subversion* configuration folder (write access to *Subversion* folder and all its subfolders). Usually, it is located in the following locations:

- Windows: user's home directory\AppData\Roaming\Subversion
- Mac OS X and Linux: user's home directory/.subversion

Updating Newly Added Resources

When you want to get from the repository a resource which is part of a newly created structure of folders, you need to also get its parent folders.
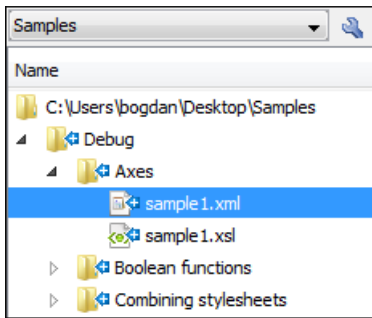
Figure 58: An incoming structure of folders from the repository

Syncro SVN Client allows you to choose how you want to deal with the entire structure from that moment onwards:

- **Update ancestor directories recursively** - This option brings the entire newly added folders structure into your working copy. In this case, the update time depends on the total number of newly incoming resources, because of the full update operation (not updating only selected resource).
- **Update selected files only (leave ancestor directories empty)** - This option brings a skeleton structure composed of the resource's parent folders only, and the selected resource at the end of the operation. All of the parent directories will have depth set to *empty* in your working copy, thus subsequent **Synchronize** operations will not report any remote modifications in those folders. If you need to update the folders to full-depth, you can use **Update to revision/depth** option from the working copy.

Cannot Access a Repository through HTTPS

If you have issues when trying to access a repository through HTTPS protocol, one of the possible causes can be the encryption protocol currently used by the application. This is happening when:

- you are running Syncro SVN Client with Java 1.6 or older;
- the repository is set to use only one of the SSLv3 or TLSv1 encryption protocols.

To solve this issue, set the *HTTPS encryption protocols* option to **SSLv3 only** or **TLSv1 only** (depending on the repository configuration).

Comparing and Merging Documents

In large teams composed either of developers or technical writers, the usage of a shared repository for the source or document files is a must. Often many authors are editing the same file at the same time.

Finding what has been modified in your files and folders can be hard. If your data is changing, you can benefit from accurate identification and processing of changes in your files and folders with Syncro SVN Client's features for comparing files and directories. These are powerful and easy to use tools that will do the job fast and thoroughly. With the new possibilities of differencing and merging, it is now easy to manage multiple changes.

Syncro SVN Client provides a simple means of performing file and folder comparisons. You can see the differences in your files and folders and merge the changes.

There are two levels on which the comparison can be done, namely comparing directories or comparing individual files.

📝 **Note:** Syncro SVN Client uses diff directories to compare archive files.

The comparison tool can also be started using command line arguments. In the installation folder there are two executable shells (`diffFiles.bat` and `diffDirs.bat` on Windows, `diffFiles.sh` and `diffDirs.sh` on Unix/Linux, `diffFilesMac.sh` and `diffDirsMac.sh` on Mac OS X). You can pass one or two command-line arguments to each of these shells. Both arguments can point to directories or archives (supported formats: *zip*, *docx*, and *xlsx*).

For example, to start the comparison between two Windows directories, use the following command:

```
diffDirs.bat "c:\Program Files" "c:\ant"
```

If there are spaces in the path names, surround the paths with quotes. If you pass only one argument, you are prompted to manually choose the second directory or archive. This is valid for the files diff utility as well.

Directories Comparison

The directories comparison result is presented as a tree of files and directories. The directories that contain different files are expanded automatically, so you can focus directly on the differences. You can merge the directories contents using the copy actions. A double click or an Enter key on a line with a pair of files starts *comparing the file content* of the two files from that line in the **Compare Files** window. Please note that the content is compared only in case of known file types, that is the files associated with the built-in editors and the file types associated with a built-in editor when the user was prompted to specify such an association (when opening for the first time a file of an unknown type).
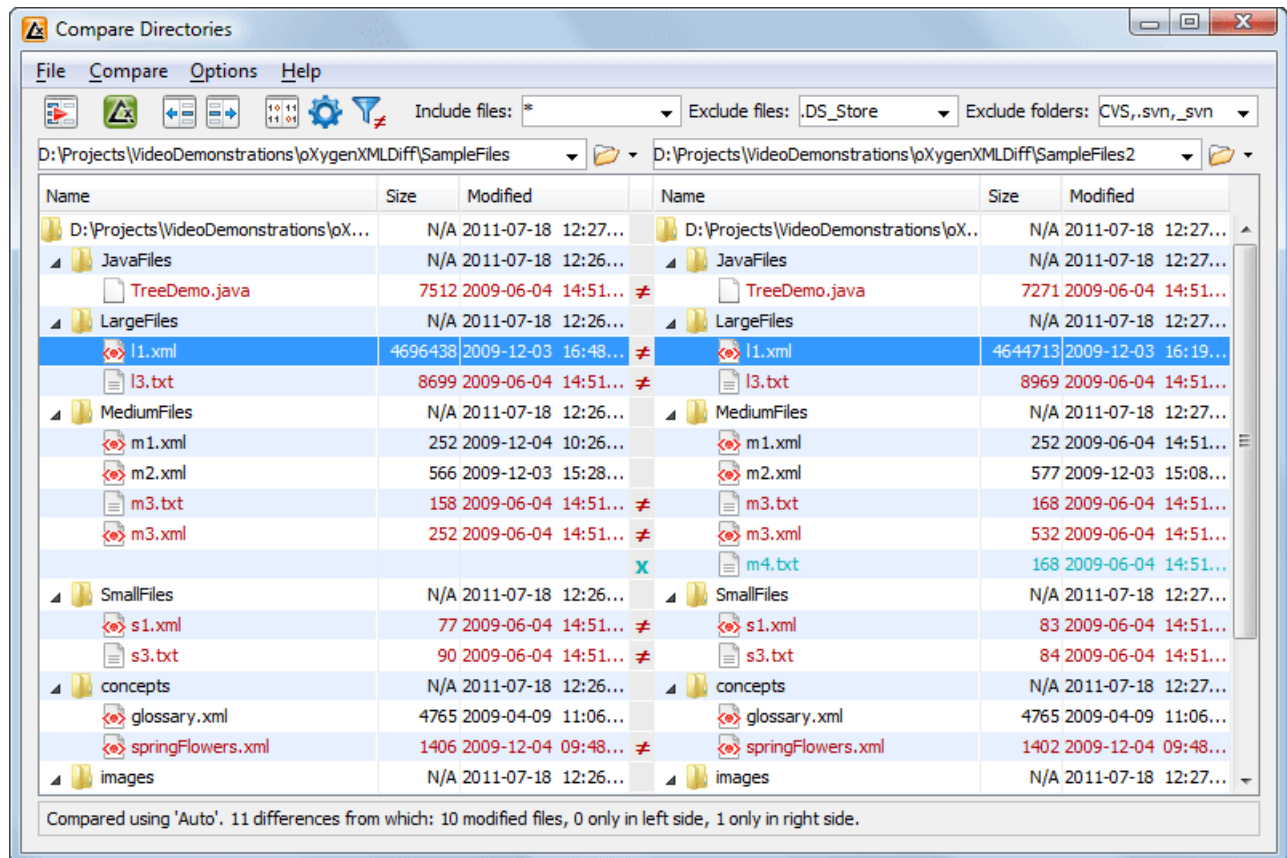
Figure 59: The Compare Directories Window

Directories Comparison User Interface

This section explains the user interface of the **Directories Comparison** window.

Compare Menu

This menu contains the following action:

- **Perform Directories Differencing** - Looks for differences between the two directories displayed in the left and right side of the application window.
- **Perform Files Differencing** - Compares the currently selected files.
- **Copy Change from Right to Left** - Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).

- ⇨ **Copy Change from Left to Right** - Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).

Compare Toolbar

The toolbar contains the following actions:

Figure 60: The Compare toolbar

- **Perform directories differencing** - Looks for differences between the two directories displayed in the left and right side of the application window.
- **Perform files differencing** - Compares the currently selected files.
- **Copy Change from Right to Left** - Copies the selected change from the right side to the left side (if there is no file/folder in the right side, the left file/folder is deleted).
- **Copy Change from Left to Right** - Copies the selected change from the left side to the right side (if there is no file/folder in the left side, the right file/folder is deleted).
- **Binary Compare** - Performs a byte-level comparison on the selected files.
- **Diff Options** - Opens the Directories Comparison preferences page.
- **Show Only Modifications** - Displays a more uncluttered file structure by hiding all identical files.
- Files and folders filters - Differences can be filtered using three filter boxes: **Include files, Exclude files, Exclude folders**. They come with predefined values and are editable to allow more custom values. All of them accept multiple comma separated values and the ***** and **?** wildcards. For example, to filter out all jpeg and gif image files, edit the **Exclude files** filter box to read ***.jpeg, *.png**. Each filter keeps a list with the latest 15 filters applied in the drop-down list of the filter box.

Directories Selector

To open the directories you want to compare, select a folder from each **Browse for local directory** button. The Diff Tool keeps track of the folders you are currently working with and those you opened in this window. You can see and select them from the two combo-boxes.

If you want to compare two archives' content you can select the archives from the **Browse for archive file** button.

Comparison Result

The directory comparison result is presented using two tree-like structures, showing files and folders names, size and modification date.

Name	Size	Modified		Name	Size	Modified
length-bad.xml	140	2010-07-12 16:43:03		length-bad.xml	140	2010-07-12 16:43:03
length-bad1.xml	140	2010-07-12 16:43:04	≠	length-bad1.xml	125	2010-09-02 11:41:14
length-bad2.xml	143	2010-07-12 16:43:04	≠	length-bad2.xml	126	2010-09-02 11:41:07
length-good.xml	141	2010-07-12 16:43:03	≠	length-good.xml	126	2010-09-02 11:41:03
length.dtd	91	2010-07-12 16:43:03	X			
length.sch	648	2010-07-12 16:43:03	X			
name-bad.xml	171	2010-07-12 16:43:03	≠	name-bad.xml	155	2010-09-02 11:40:38
name.dtd	192	2010-07-12 16:43:03	≠	name.dtd	158	2010-09-02 11:42:09
name.sch	566	2010-07-12 16:43:04	≠	name.sch	351	2010-09-02 11:40:50
present-bad.xml	224	2010-07-12 16:43:04	≠	present-bad.xml	171	2010-09-02 11:41:47
present.dtd	130	2010-07-12 16:43:03	≠	present.dtd	60	2010-09-02 11:40:28
present.sch	482	2010-07-12 16:43:03	≠	present.sch	295	2010-09-02 11:41:51
required-bad1.xml	205	2010-07-12 16:43:03	≠	required-bad1.xm	163	2010-09-02 11:41:24
required-bad2.xml	189	2010-07-12 16:43:03	≠	required-bad2.xm	155	2010-09-02 11:41:18
required-good.xml	197	2010-07-12 16:43:03		required-good.xm	197	2010-07-12 16:43:03
required.dtd	128	2010-07-12 16:43:03	≠	required.dtd	91	2010-09-02 11:41:42
required.sch	612	2010-07-12 16:43:03	≠	required.sch	363	2010-09-02 11:41:38
author	N/A	2010-09-01 11:23:14		author	N/A	2010-09-02 11:42:38
author.sch	613	2010-07-12 16:43:04	X			
source1.xml	165	2010-07-12 16:43:04	X			
source2.xml	184	2010-07-12 16:43:04	X			
paragraph	N/A	2010-09-01 11:23:14		paragraph	N/A	2010-09-02 11:39:34

Compared using 'Timestamp (last modified date/time)'. 20 differences from which: 13 modified files, 7 only in left side, 0 only in right side.

Figure 61: Comparison result

A column holding graphic symbols separate the two tree-like structures. The graphic symbols can be:

- an "X" sign, when a file or a folder exists only in one of the compared directories;
- a "not-equal" sign, if a file exists in both directories, but the content is different. The same sign appears when a collapsed folder contains modified files.

Compare Images

When double-clicking a line containing two different images a compare images dialog is displayed. The dialog presents the images in the left and right part scaled to fit the view available area. You can use the contextual menu actions to scale the images at their original size or scale them down to fit in the view area.

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

Files Comparison

To compare two files side by side, open the **Diff Files** dialog from **Tools** > **Compare Files**. Using the **Browse for local file** button open a file in the left side of the dialog, and the file you want to compare it to in the right side. To highlight the differences between the two files, click the **Perform File Differencing** button. The line numbers on each side help you to identify the locations of the differences quickly.

You can edit both the source and the target file. The differences are refreshed when you save the modified document.
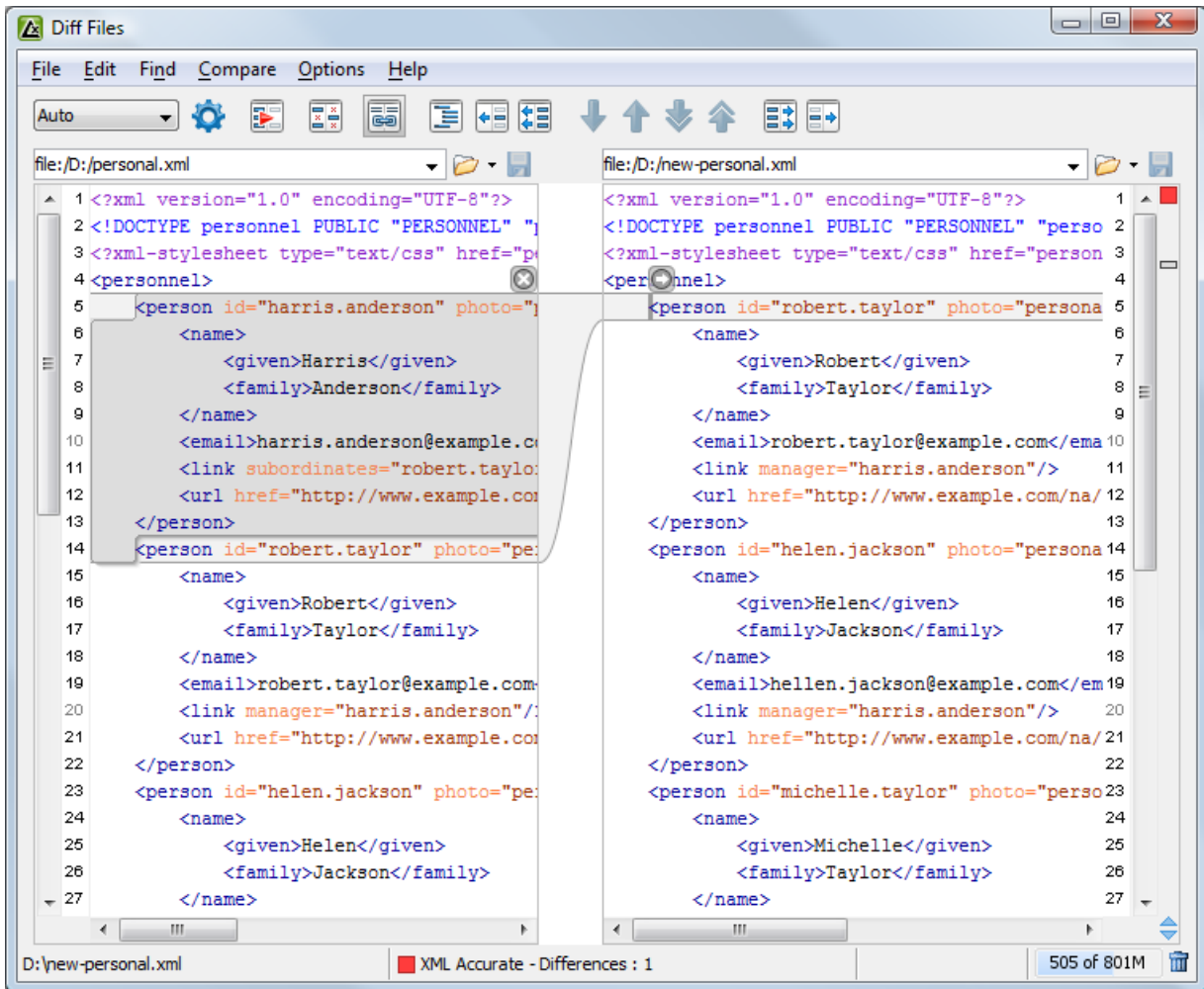
Figure 62: The Compare Files Window

Adjacent changes are grouped into blocks of changes. This layout allows an easier visual identification and focuses on a group of related changes.

A widget containing actions that can be used to copy or append changes from either of the two sides is displayed when you select a change:

- **Append left change to right** and **Append right change to left**- copy the content of the selected change from one side and appends it after the content of the corresponding change from the other side; as a result, the side towards the arrows point to, will contain the changes from both sides.
- **Copy change from left to right** and **Copy change from right to left** - replace the content of a change from one side with the content of the corresponding change from the other side.

Main Menu

This section explains the menu actions of the **Files Comparison** window.

File Menu

The following actions are available:

- **Source** - The file is displayed in the left side of the application window

    - **Source >** **Open** - Browses for a source file.
    - **Source >** **Open URL** - Opens URL to be used as a source file.

- **Source** > 📂 **Open File from Archive** - Browses an archive content for a source file.
- **Source** > 💾 **Save** - Saves the changes made in the source file.
- **Source** > **Save As...** - Displays the **Save As** dialog that allows you to save the source file with a new name.

- **Target** - The file is displayed in the right side of the application window

    - **Target** > 📂 **Open** - Browses for a target file.
    - **Target** > 📂 **Open URL** - Opens URL to be used as a target file.
    - **Target** > 📂 **Open File from Archive** - Browses an archive content for a target file.
    - **Target** > 💾 **Save** - Saves the changes made in the target file.
    - **Target** > **Save As...** - Displays the **Save As** dialog that allows you to save the target file with a new name.

- **Exit** - Quits the application.

Edit Menu

The following actions are available:

- ✂ **Cut** - Cut selection to clipboard from the local file currently opened in the focused **Compare** editor.
- 📋 **Copy** - Copy selection to clipboard from the local file currently opened in the focused **Compare** editor.
- 📋 **Paste** - Paste selection from clipboard in the local file currently opened in the focused **Compare** editor.
- ↩ **Undo** - Undo edit changes in the local file currently opened in the focused **Compare** editor.
- ↪ **Redo** - Redo edit changes in the local file currently opened in the focused **Compare** editor.

Find Menu

The find actions are the following:

- 🔍 **Find/Replace** - Perform *find/replace* operations in the file currently opened in the focused **Editor**.
- 🔍 **Find Next** - Go to the next match using the same options of the last *find* operation. The action runs in the two editor panels.
- 🔍 **Find Previous** - Go to the previous match using the same options of the last *find* operation. The action runs in the two editor panels.

Compare Menu

The following actions are available in this menu:

🔳 **Perform Files Differencing** - performs a comparison between the source and target files;

⬇ **Next Block of Changes** - jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

📄 **Note:** A change block groups one or more consecutive lines that contain at least one change.

⬆ **Previous Block of Changes** - jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

⬇ **Next Change** - jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change;

⬆ **Previous Change** - jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change;

⬇ **Last Change** - jumps to the last change from the current file;

⬆ **First Change** - jumps to the first change from the current file;

Copy All Changes from Left to Right - copies all changes from source to target file;

Copy All Changes from Right to Left - copies all changes from target to source file;

Copy Change from Left to Right - copies the selected difference from source to target file;

Copy Change from Right to Left - copies the selected difference from target to source file;

Show Word Level Details - provides a word-level comparison of the selected change;

Show Character Level Details - provides a character-level comparison of the selected change.

Options Menu

- **Preferences** - Opens the preferences pages.

- **Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page. Here you can configure all keyboard shortcuts available for menu items.

- **Reset Global Options** - Resets options to their default values.

- **Import Global Options** - Allows you to import an options set you have previously exported.

- **Export Global Options** - Allows you to export the current options set to a file.

Help Menu

The **Help** menu contains the following options:

- **Help** - opens the **Help** dialog;
- **Report problem** - opens a dialog that allows the user to write the description of a problem that was encountered while using the application;
- **Improvement Program Options** - allows you to activate or deactivate the application Product Improvement Program;
- **Support center** - opens the Syncro SVN Client Support Center web page in a browser.

Compare Toolbar

This toolbar contains the operations that can be performed on the source and target files.

Figure 63: The Compare Toolbar

The following actions are available:

- **Algorithm** - This option box allows you to select one of the 6 available compare algorithms:

  - *Characters* algorithm computes the differences at character level;

  - *Words* algorithm computes the differences at word level;

  - *Lines* algorithm computes the differences at line level, meaning that it compares two files looking for the first identical line of text. When it is found, it is considered a match. The content that precedes the match is considered to be a difference and marked accordingly. Then the algorithm continues to look for matching lines marking the content between them as differences.

  - *Syntax Aware* for known file types, like the ones listed in the **New** dialog box, for example the XML file type (which includes, among others, XSLT files, XSL-FO files, XSD files, RNG files, NVDL files), the XQUERY file type (`.xquery`, `.xq`, `.xqy`, `.xqm` extensions), the DTD file type (`.dtd`, `.ent`, `.mod` extensions), the TEXT file type (`.txt` extension), the PHP file type (`.php` extension), etc.

    This algorithm splits the files into sequences of *tokens* and computes the differences between them. A *token* can have a different meaning, depending on the type of the compared files. For example:

- when comparing XML files, a token can be one of the following:

    - the name of an XML tag;
    - the '<' character;
    - the '/>' sequence of characters;
    - the name of an attribute inside an XML tag;
    - the '=' sign;
    - the '"' character;
    - an attribute value;
    - the text string between the start tag and the end tag (that is a text node which is a child of the XML element corresponding to the XML tag that encloses the text string).

- when comparing plain text files (identified by the .txt extension), a token can be any continuous sequence of word characters (letters, digit, the '_' character) or any continuous sequence of whitespace characters including a newline character.

- *XML Fast* works on larger files but it is less precise than XML Accurate.

- *XML Accurate* works best on small XML files.

- *Auto* selects the most appropriate algorithm, based on the files content and size. By default, the **Auto** mode is selected.

- **Perform Files Differencing** - performs a comparison between the source and target files;

- **Ignore Whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before performing the comparison, the application normalizes the content and trims its leading and trailing whitespaces.

- **Synchronized scrolling** - Synchronizes scrolling of the two open files, so that a selected difference can be seen on both sides of the application window. This action enables/disables the previous described behavior.

- **Format and Indent Both Files** - Formats and indents both files before you compare them. Use this option when you compare files that contain long lines which makes it difficult to spot differences.

- **Copy Change from Right to Left** - Copies the selected difference from target to source file.

- **Copy All Changes from Right to Left** - Copies all changes from target to source file.

- **Next Block of Changes** - jumps to the next block of changes. This action is disabled when the cursor is positioned on the last change block or when there are no changes in the document;

- **Previous Block of Changes** - jumps to the previous block of changes. This action is disabled when the cursor is positioned on the first change block or when there are no changes in the document;

- **Next Change** - jumps to the next change from the current block of changes. When the last change from the current block of changes is reached, it highlights the next block of changes. This action is disabled when the cursor is positioned on the last change;

- **Previous Change** - jumps to the previous change from the current block of changes. When the first change from the current block of changes is reached, it highlights the previous block of changes. This action is disabled when the cursor is positioned on the first change;

- **Copy All Changes from Left to Right** - Copies all changes from source to target file.

- **Copy Change from Left to Right** - Copies the selected difference from source to target file.

- **First Change** - jumps to the first change from the current file;

Files Selector

To open the source and target files where you want to see the differences, use one of the **Browse for local**, **Browse for remote** or **Browse for archive** buttons. The Diff Tool keeps track of the files you are currently working with and those you opened in this window. You can see and select them from the two combo-boxes.

You can also save the changes in the source file or the target file by clicking the corresponding "Save" button.

File Contents Panel

The files are opened in two side-by-side editors. The text view is used, offering a better view of the differences.

The two editors are kept in sync, so if you scroll the text in one of them, the other one also scrolls to show the difference side-by-side. The differences are indicated using highlights connected through colored areas. To navigate between differences, do one of the following:

- press the **Go to modification** buttons;
- select a change by clicking it in the overview ruler located in the right-most part of the window. The overview ruler contains a success indicator in its upper part. It turns green in case there are no differences or red if some differences are found.
- click a colored area between the text editors.

You can edit either the source or the target file. The differences are refreshed when you save the modified document.

Both editors provide a contextual menu that contains *edit*, *merge* and *navigation* actions.

The **Find/Replace** dialog is displayed by pressing **Ctrl+F** (**Cmd+F** on Mac). **Find/Replace** options are also available:

- **F3** used for performing another search using the last search configuration;
- **Shift+F3** for performing another search in backward direction using the last search configuration.

If the compared blocks of text are too large and you want to see the differences at a finer level, you can use the comparison at **Word** or **Character** level.

Word Level Comparison

This option is only available if differences exist between the source and the target file. You can go to Word Level Comparison by clicking the **Show word level details** menu item from the **Compare** menu.
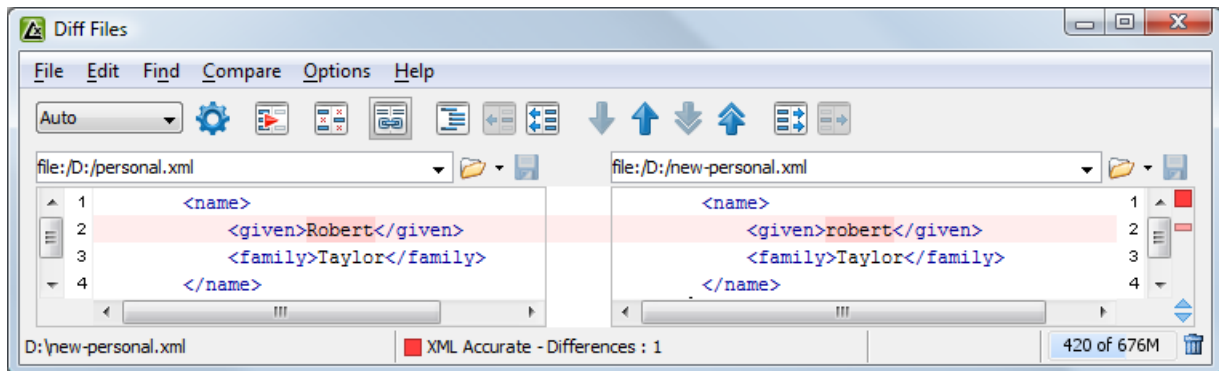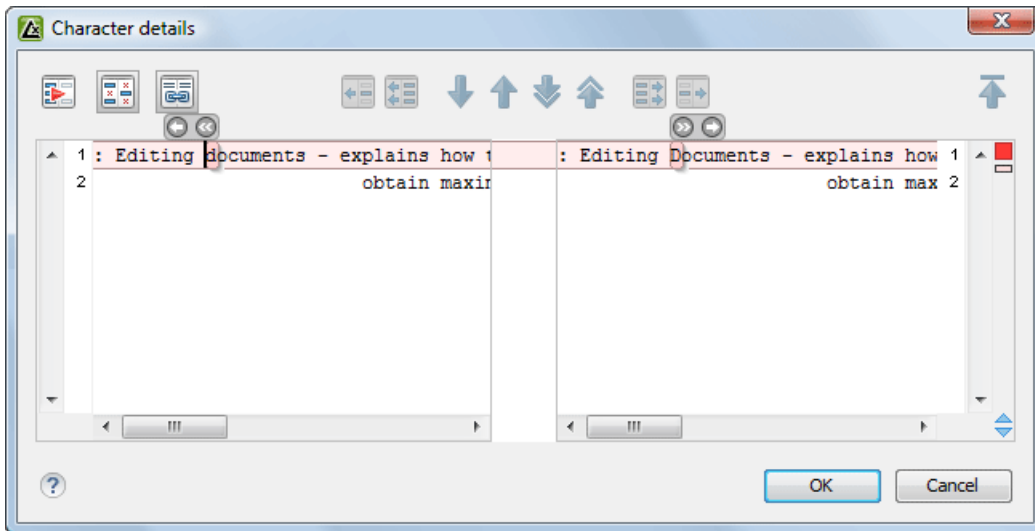

Figure 64: Word Level Comparison

Character Level Comparison

This option is only available if modifications exist between the source and the target file. You can go to Character Level Comparison by clicking the **Show Character Level details** menu item from the **Compare** menu.

Figure 65: Character Level Comparison

XML Diff API

The following interface is available for calling the XML diff processor from a custom Java application:

- `ro.sync.diff.api.DifferencePerformer` - It compares two resources of a given content type using a set of options. It has the following methods:

    - `performDiff` - Perform a diff operation between the two specified resources. It returns a list with the differences. The parameters are the following:

        1. leftContentReader - A value of type `java.io.Reader` that provides the content of the first resource.
        2. rightContentReader - A value of type `java.io.Reader` that provides the content of the second resource.
        3. leftSystemId - A string value that is the location of the first resource.
        4. rightSystemId - A string value that is the location of the second resource
        5. contentType - A constant from the `ro.sync.diff.api.DiffContentTypes` interface.
        6. diffOptions - The user options controlling algorithm strength, ignore whitespaces, ignore comments, merge adjacent differences, etc. It is a value of type `ro.sync.diff.api.DiffOptions`.
        7. diffProgressListener - An object that will be notified about the progress of the diff operation. It is a value of type `ro.sync.diff.api.DiffProgressListener`.

    - `stop` - Signal to the diff performer that it must stop.

An example of this interface can be found in the class `ro.sync.diff.api.sample.DiffXMLFilesSample` which is included in the XML Diff SDK.

Chapter

4

Configuring Syncro SVN Client

Topics:

This chapter presents all the user preferences that allow you to configure the application .

Configuring Options

A set of options controls Syncro SVN Client, allowing you to configure most of the features. To offer you the highest degree of flexibility in customizing the application to fit the end-user's role in your organization, Syncro SVN Client comes with several distinct layers of option values, arranged here according to their priority, from low to high:

• *Default Options* - should be regarded as factory defaults or built-in values.

   This predefined set of values are tuned so Syncro SVN Client behaves optimally in most working environments.

• *Customized Default Options* - designed to customize the initial option values for a group of users.

   This layer allows an administrator to deploy the application preconfigured with a standardized set of option values.

   📋 **Note:** Once this layer is set, it represents the initial state of Syncro SVN Client when an end-user presses the **Restore defaults** or **Reset Global Options** buttons.

• *Global Options* - allows individual users to personalize Syncro SVN Client according to their specific needs.

   📋 **Note:** If you set a specific option in one of the layers, but it is not applied in the application, make sure that one of the higher priority layers is not overwriting it.

Customized Default Options

You can overwrite the *Default Options* values by loading a set of options, from a specific XML structure, stored in an *options file* (a file that contains all the Syncro SVN Client options). To create an *options file*:

• set the values you want to impose as defaults in the *Preferences pages*;
• select **Options** > **Export Global Options**.

There are two ways to make Syncro SVN Client use such an options configuration file to extract the customized default options:

• set the file path of the options configuration file as value of `com.oxygenxml.default.options.`

   The file must be specified with an URL or a file path relative to the application installation folder. For example:

   ```
   -Dcom.oxygenxml.default.options=options/default.xml
   ```

• copy the options configuration file in the folder.

   📋 **Note:** Make sure that the options configuration file has an `.xml` or an `.xpr` extension, like `default-options.xml`.

Importing / Exporting Global Options

The import/export operations are located in the **Options** menu. These operations allow you to load or save global preferences as an XML file. You can use this file to reload saved options both on your computer and on others.

The following actions are available in the **Options** menu:

• **Reset Global Options** - restores the preferences set to *Customized Default Options layer*, or if this level is not defined, to *Default Options layer*;
• **Import Global Options** - allows you to import a set of *Global Options* from an *options file*;
• **Export Global Options** - allows you to export the entire set of *Global Options* to an options file.

Syncro SVN Client automatically stores your options in an `options` file. Depending on the platform you are using, this file is located in the following directories:

- [user-home-folder]\Application Data\ com.syncrosvnclient for Windows XP;
- [user-home-folder]\AppData\Roaming\ com.syncrosvnclient for Windows Vista/7;
- [user-home-folder]/Library/Preferences/ com.syncrosvnclient for Mac OS X;
- [user-home-folder]/. com.syncrosvnclient for Linux.

The name of the `options` file of Syncro SVN Client 9.1 is oxyOptionsSvn8.0.xml.

Preferences

Once the application is installed you can use the **Preferences** dialog accessed from menu **Options** > **Preferences** to customize the application.

Option pages are organized hierarchically in a tree-like structure displayed in the left side of the **Preferences** dialog. The option tree features a search box that allows you to look for a specific option name: just type-in relevant keywords and the entire options structure is trimmed down. To navigate the tree structure, use the *Up*/*Down* keys or click the tree entries to display the options pages in the right side of the **Preferences** dialog.
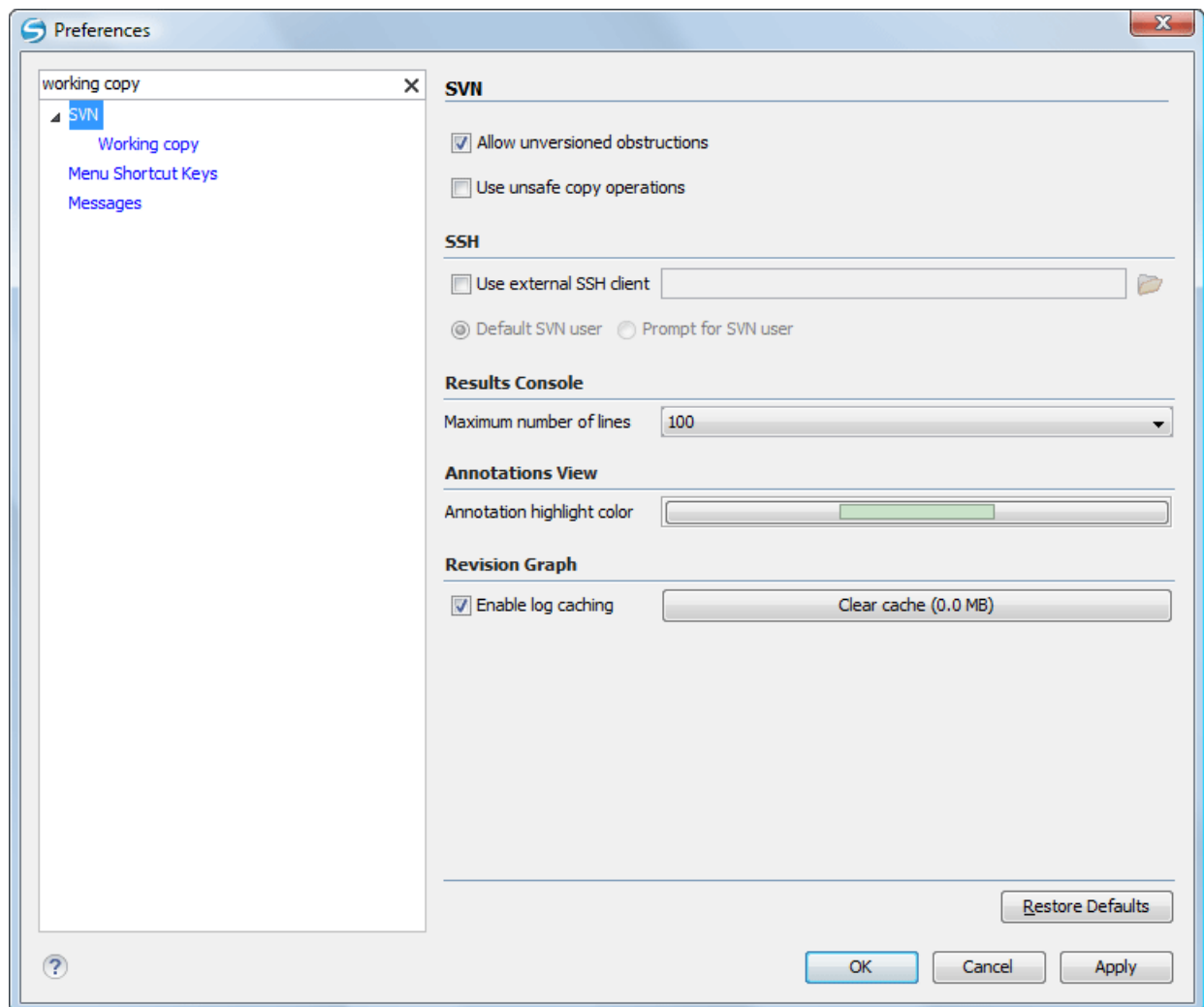

Figure 66: The Search field from the Preferences dialog

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

If you don't know how to use a specific preference that is available in any **Preferences** panel or what effect it will have you can open a help page about the current panel at any time pressing the help button ⑦ located in the left bottom corner of the dialog or pressing the F1 key.

Global Preferences

To display the **Global** preferences of Syncro SVN Client, go to **Options** > **Preferences** > **Global**.

The following options are available in the **Global** preferences page:

- **Automatic Version Checking** - checks whether a new version of Syncro SVN Client is available;
- **Language** - allows you to localize the interface and change the language to English, French, German, Dutch, Japanese, or Italian. Restart the application to apply the current selection;
- **Other language** - allows you to set the file (either *properties* or XML) that contains interface messages translated into a different language than the default one;

    📄 **Note:** After restarting the application, if some interface labels are not rendered correctly (for example Chinese or Korean characters), install the corresponding language pack from your OS installation kit (for example the East-Asian language pack).

- **Look and Feel** - use this option to change graphic style (look and feel) of the user interface;
- **Styles** - on Windows there are available the following styles:

    - Office 2003
    - Vsnet
    - Eclipse
    - Xerto
    - Default

    📄 **Note:** After changing the style, restart the application in order for the modification to take effect.

    On Linux there are available the following styles:

    - Eclipse
    - Default

    📄 **Note:** After changing the style, you have 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 for the **Office 2003** and **Default** styles. In these cases, the following themes are available:

    - normal Color
    - home Stead
    - metallic
    - default
    - gray

    On Linux and Mac OS X this option is not available.

- **Line separator** - this option defines the line separator. The **System Default** choice sets the platform-specific line separator;
- **Detect the line separator on file open** - when this option is checked, the editor detects the line separator when the edited file is loaded and it uses it when the file is saved. The new files are saved using the line separator defined by the **Line separator** option;

- **Show memory status** -enable this option to view the memory Syncro SVN Client uses. To free memory, click the

   Run Garbage Collector button located in the bottom right corner of the screen. The memory status bar has a light blue background which turns yellow or red when Syncro SVN Client uses too much memory;
- **Show Java vendor warning at startup** - Sun Microsystems/Oracle Java VM (on Windows and Linux) or Apple Computer Java VM (on Mac OS X) is recommended for running the application. If a different Java Virtual Machine is used, then a warning is displayed. This option allows the user to choose whether the warning dialog is shown or not:
- **Show hidden files and directories** - shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on Mac OS X.

Fonts Preferences

To open the **Fonts** preferences page, go to **Options** > **Preferences** > **Fonts**.

The following options are available:

- **Editor** - The family and size of the font used for displaying text information in the text editor.

   Note: On Mac OS X, the default font, Monaco, cannot render bold typeface characters.

- **Text antialiasing** - Allows you to set text anti-aliasing behavior:

  - **Default** - allows the application to use the setting of the operating system, if available;
  - **On** - sets the text anti-aliasing to pixel level;
  - **Off** - disables text anti-aliasing;
  - sub-pixel anti-aliasing modes, like GASP, LCD_HRGB, LCD_HBGR, LCD_VRGB, and LCD_VBGR.

- **Text components** - The family and size of the font used for displaying text information in text components. After changing the font, restart the application for the change in this setting to take effect.
- **GUI** - The family and size of the font used for displaying user interface labels. After changing the font, restart the application for the change in this setting to take effect.

Encoding Preferences

To open the **Encoding** preferences page, go to **Options** > **Preferences** > **Encoding**. The following encoding preferences are available:

- **Encoding for non XML files** - specifies the default encoding the Syncro SVN Client uses to open 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 declared at the beginning of the file in a special declaration element or it is assumed to be the default value, which is UTF-8;
- **Encoding errors handling** - 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:

  - **REPORT** - displays an error dialog box with the character that cannot be represented in the specified encoding. Unrecognized characters are rendered as an empty box. This is the default option;
  - **IGNORE** - the character is ignored and it is not included in the document displayed in the editor panel;

     Attention: If you edit and save the document, the characters that cannot be represented in the specified encoding are dropped.

  - **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 replacement character code is 63.

Editor Preferences

To open the **Editor** preferences page, go to **Options** > **Preferences** > **Editor**.

Use these options to configure the visual aspect of the text editor.

The following options are available:

- **Selection background color** - Background color of selected text.
- **Selection foreground color** - Text color of selected text.
- **Editor background color** - Background color for both text editor and Diff Files editors.
- **Editor caret color** - Customize the caret color.
- **Highlight current line** - Foreground color of the line numbers displayed in the editor panels.
- **Show line numbers** - Enables the line numbers stripe in the editor panels and in the **Results** view of the Debugger perspective.
- **Show TAB/NBSP/EOL/EOF marks** - Marks the *TAB*/*NBSP*/*EOL*/*EOF* characters using small icons, for a better visualization of the document. Also sets the marks color.
- **Show SPACE marks** - When checked, the *space* characters are rendered with a dot.
- **Indent with tabs** - When checked, sets the indent size to a *tab* unit. When unchecked, the application uses space characters to form an indent. The number of space characters that form a *tab* is defined by the **Indent size** option.
- **Indent size** - Sets the number of space characters or the tab size that equals a single indent. An *indent* can be a number of spaces or a tab, selectable using the **Indent With Tabs** option. For example, if set to 4, one tab equals:

  - either 4 space characters, if the **Indent With Tabs** option is unchecked;
  - or one tab that spans 4 characters, if the **Indent With Tabs** option is checked.

Open / Save Preferences

To open the **Open / Save** preferences page, go to **Options** > **Preferences** > **Editor** > **Open / Save**.

The following options are available:

- **When bidirectional (BIDI) text is detected** - allows you to choose the application behavior when you try to open a file that contains BIDI Unicode characters. You can choose between **Enable bidirectional editing mode**, **Disable bidirectional editing mode** and **Prompt for each document**.
- **Disable bidirectional text support for documents larger than (Characters)** - when you try to open a document that exceeds the specified limit, the bidirectional editing mode is turned off, even if the **When bidirectional (BIDI) text is detected** option is set to **Enable bidirectional editing mode**.
- **Show BIDI limit warning** - if enabled, a warning dialog is shown when the ope file which contains bidirectional characters is too large and bidirectional support is disabled to improve application performance.

SVN Preferences

To open the **SVN** preferences page, go to **Options** > **Preferences** > **SVN**. In this preferences page the user preferences for the embedded SVN client tool are configured. Some other preferences for the embedded SVN client tool 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.
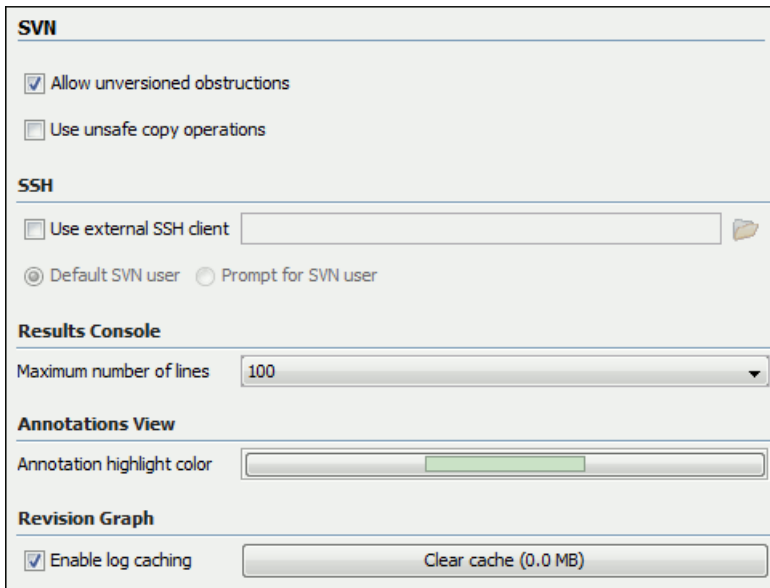
Figure 67: The SVN Preferences Panel

The SVN preferences are the following:

- **Enable symbolic link support** (*available only on Mac OS X and Linux*) - Apache Subversion^{TM} 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: Windows file systems have no symbolic links, so a Windows 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, that is a text file instead of symbolic link is created.

    Important: It is recommended 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 a path that includes a symbolic link.

- **Allow unversioned obstructions** - controls how should be handled working copy resources being ignored / unversioned when performing an update operation and from the repository are incoming files with the same name, in the same location, that intersect with those being ignored / unversioned. If the option is enabled, then the incoming items will become BASE revisions of the ones already present in the working copy, and those present will be made versioned resources and will be marked as modified. Exactly as if the user first made the update operation and after that he / she modified the files. If the option is disabled, the update operation will fail when encountering files in this situation, possibly leaving other files not updated. By default, this option is enabled.
- **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.
- **HTTPS encryption protocols** - sets a specific encryption protocol to be used when the *application accesses a repository through HTTPS protocol*. You can choose one of the following values:

    - **SSLv3, TLSv1** (default value)
    - **SSLv3 only**
    - **TLSv1 only**

- **SSH** - specifies 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 user name and / or private key / passphrase. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example on Windows the following command line uses the `plink.exe` tool as external SSH client for connecting to the SVN repository with SVN+SSH:

```
C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
host_name_or_IP_address_of_SVN_server
```

- **Results Console** - specifies the maximum number of lines displayed in the **Console** view. The default value is 1,000.
- **Annotations View** - sets 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** - enables 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. If a cache is used it will finish the action much faster.

Working Copy Preferences

To open the **Working Copy** preferences page, go to **Options** > **Preferences** > **SVN** > **Working Copy**. The option that you are able to configure in this preferences page are specific to SVN working copies:

- **Working copies location** - Allows you to define a location where you keep your working copies. This location is automatically suggested when you checkout a new working 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.
- **When loading an old format working copy** - You can instruct Syncro SVN Client to do one of the following:

  - **Automatically upgrade** - Older format working copies are upgraded to the newest known format.
  - **Never upgrade** - Older format working copies are left untouched. No attempt to upgrade the format is made.
  - **Always ask** - You are notified when such a working copy is used and you are allowed to choose what action to be taken - to upgrade or not the format of the current working copy.

- **Enable working copy caching** - If checked, the content of the working copies is cached for refresh operations.
- **Automatically refresh the working copy** - If checked, the working copy is refreshed from cache. Only the new changes (modifications with a date/time that follows the last refresh operation) are refreshed from disk. Enabled by default.
- **When synchronizing with repository** - The action that will be executed automatically after the **Synchronize** action. The possible actions are:

  - **Always switch to 'Modified' mode** - The **Synchronize** action is followed automatically by a switch to **Modified** mode of **Working Copy** view, if **All Files** mode is currently selected.
  - **Never switch to 'Modified' mode** - Keeps the currently selected view mode unchanged.
  - **Always ask** - The user is always asked if he wants to switch to **Modified** mode.

- **Application global ignores** - Allows setting file patterns that may include the `*` and `?` wildcards for unversioned files and folders that must be ignored when displaying the working copy resources in *the Working Copy view*. These patterns are case-sensitive. For example, `*.txt` matches `file.txt`, but does not match `file.TXT`.

Diff Preferences

The **Diff** preferences page allows you to set the compare options for SVN client. To open the **Diff** preferences page, go to **Options** > **Preferences** > **Diff**.
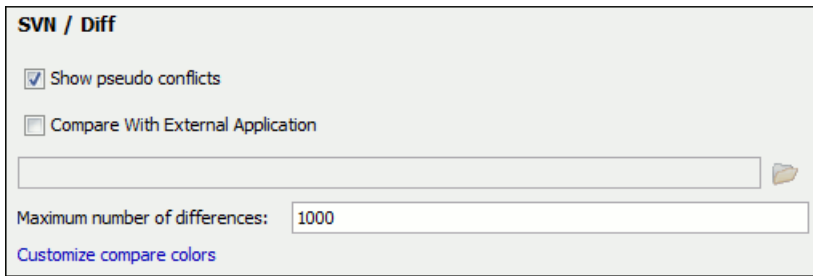
Figure 68: The SVN Diff Preferences Panel

The SVN diff preferences are the following:

- **Show pseudo conflicts** - specifies whether the *the Compare view* displays pseudo-conflicts . A pseudo-conflict occurs when two developers make the same change, for example when they both add or remove the same line of code.
- **Compare With External Application** - specifies 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. The parameter ${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison: the working copy version of a file is compared with the repository version, with the BASE revision (the latest revision read from the repository by an Update or Synchronize operation) being the common ancestor of these two compared versions.

- **Maximum number of differences** - sets the maximum number of differences allowed in the view.

Appearance Preferences

To open the **Files Comparison / Appearance** preferences page, go to **Options** > **Preferences** > **Diff** > **Files Comparison** > **Appearance**. This preferences page offers the following options:
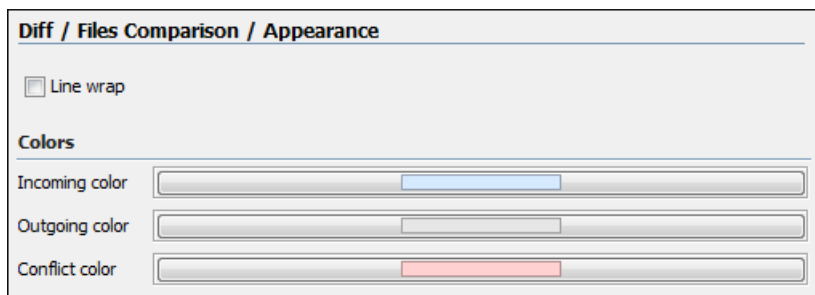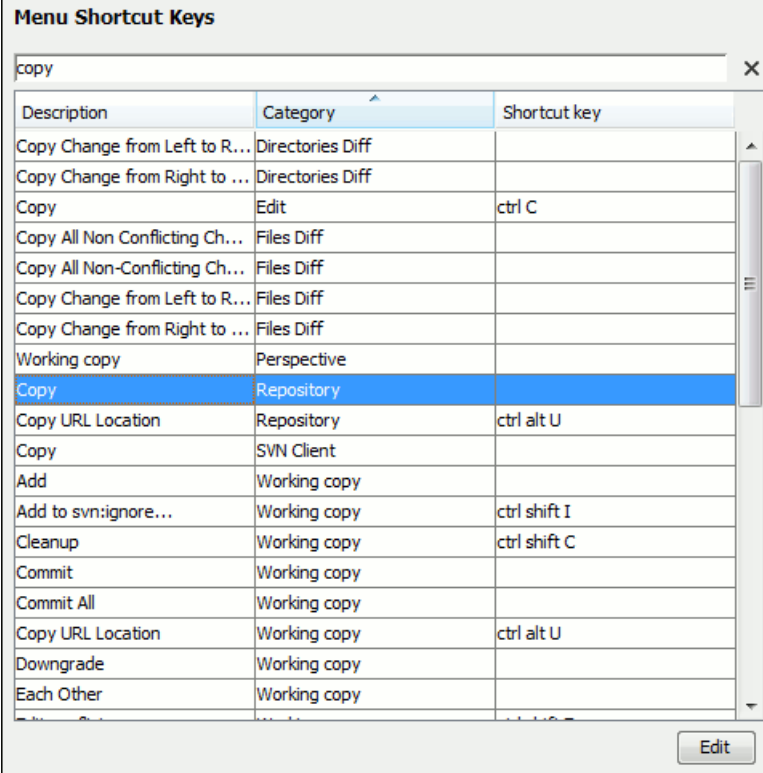
Figure 69: Files Comparison Appearance Preferences Panel

- **Line wrap** - wraps at the right margin of each panel the lines presented in the two diff panels, so no horizontal scrollbar is necessary;
- **Incoming color** - specifies the color used for incoming changes on the vertical bar, that shows the differences between the compared files;
- **Outgoing color** - specifies the color used for outgoing changes on the vertical bar, that shows the differences between the compared files;
- **Conflict color** - specifies the color used for conflicts on the vertical bar, that shows the differences between the compared files.

Menu Shortcut Keys Preferences

To open the **Menu Shortcut Keys** preferences page, go to **Options** > **Preferences** > **Menu Shortcut Keys**. You can use this page to configure shortcut keys for the actions available in Syncro SVN Client. The shortcuts assigned to menu items are displayed in the below table.

Figure 70: The Menu Shortcut Keys Preferences Panel

The **Menu shortcut Keys** preferences page also contains the shortcuts that you define at *document type* level.

> **Note:** A shortcut defined at *document type* level overwrites a default shortcut.

To find a specific action, you can use the filter to search through the **Description**, **Category**, and **Shortcut Key** columns:

- **Description** - this column provides a short description of the action;
- **Category** - this column provides a classification of the actions in categories for easier management. For example, you can distinguish the **Cut** operation available in the **Text** view from the one available in the **Tree** view by assigning them to different categories;
- **Shortcut key** - this column holds the combination of keys on your keyboard that launches the action. You can either double click a row of the **Shortcut key** column or press the **Edit** button to enter a new shortcut;

SVN File Editors Preferences

Each type of file is associated with a type of editor which opens the 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 preferences panel **SVN File Editors** which is opened from menu **Options** > **Preferences** > **SVN File Editors**.
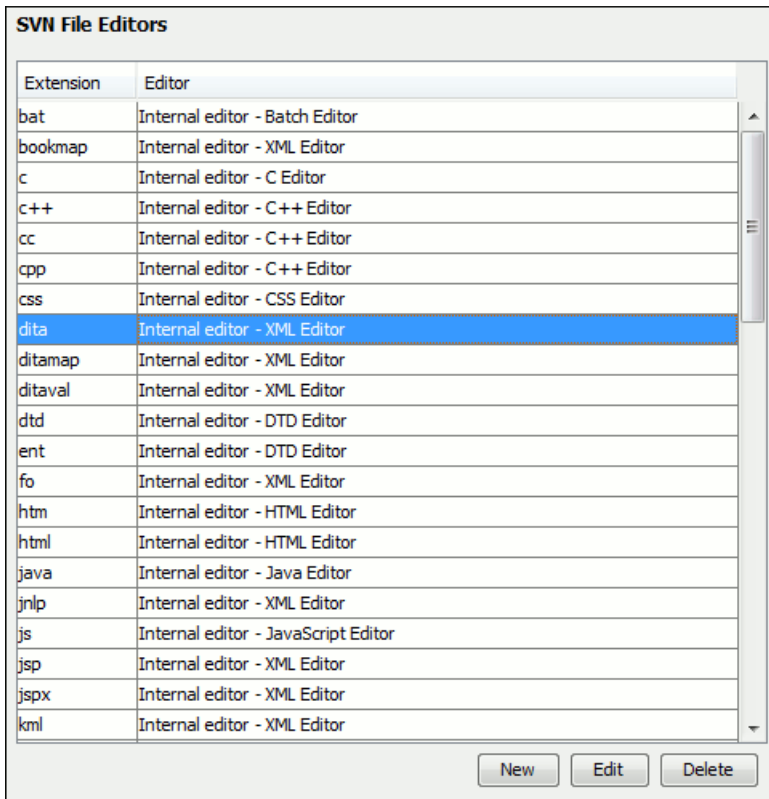
Figure 71: The SVN File Editors Preferences Panel

The **Edit** button 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 on opening a file from one of the Syncro SVN Client views.
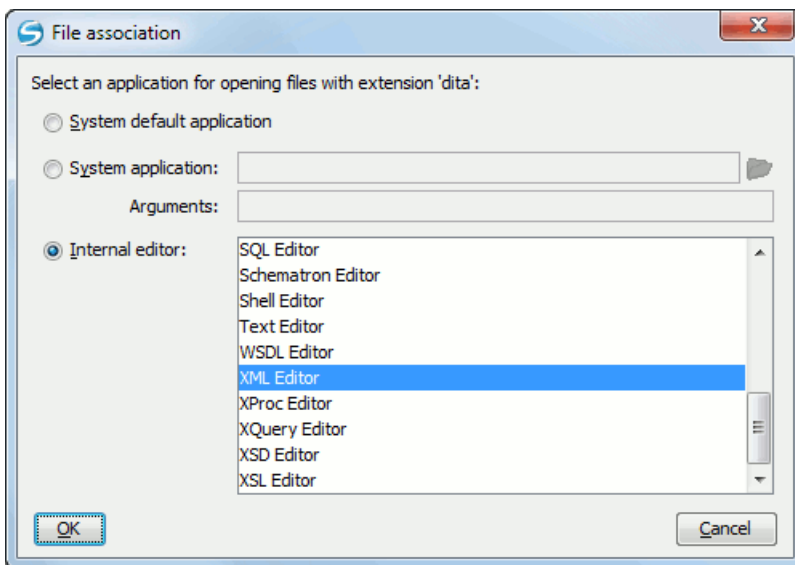
Figure 72: The Open With Dialog

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

- **System default application** - Opens the selected file using the application that is associated with that file extension by default in the 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 the command line of 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** - Allows selecting an editor type from the built-in editors that Syncro SVN Client comes with. By default, this option is disabled when selecting directories.

If a file type is associated with an internal editor other than an XML editor type 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.

The Network Connection Settings Preferences

This section presents the options available in the **Network Connection Settings** preferences pages.

Proxy Preferences

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. If you are not sure whether your computer is required to use a proxy server to connect to the Internet or you do not know the proxy parameters, consult your network administrator.

To open the **Proxy** preferences page, go to **Options** > **Preferences** > **Network Connection Settings** > **Proxy**. The following options are available:

- **Direct connection** - specifies whether the HTTP(S) connections go directly to the target host without going through a proxy server;
- **Use system settings** - specifies whether the HTTP(S) connections go through the proxy server set in the operating system. For example, on Windows the proxy settings are the ones that Internet Explorer uses;

  > ⚠ **Attention:** 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 on KDE based ones as *the Java virtual machine does not offer the necessary support yet*.

- **Manual proxy configuration** - specifies whether the HTTP(S) connections go through the proxy server specified in the **Address** and **Port** fields;
- **No proxy for** - specifies the hosts to which the connections must not go through a proxy server;
- **User** - specifies the user necessary for authentication with the proxy server;
- **Password** - specifies the password necessary for authentication with the proxy server;
- **SOCKS Proxy** - In this section you set the host and port of a SOCKS proxy through which all the connections pass. If the **Address** field is empty the connections use no SOCKS proxy.

HTTP(S) Preferences

To open the **HTTP(S)/WebDAV** preferences page, go to **Options** > **Preferences** > **Connection settings** > **HTTP(S)/WebDAV**. The following options are available:

- **Read Timeout (seconds)** - The period in seconds after which the application considers that an HTTP server is unreachable if it does not receive any response to a request sent to that server.
- **Encryption protocols (SVN Client only)** - this option is available only if you run the application with Java 1.6 or older. Sets a specific encryption protocol used when a repository is accessed through HTTPS protocol. You can choose one of the following values:

  - **SSLv3, TLSv1** (default value);
  - **SSLv3 only**;
 - **TLSv1 only**.

SSH Preferences

To open the **SSH** preferences page, go to **Options** > **Preferences** > **Connection settings** > **SSH**. The following options are available:

- **Use external SSH client** - allows you to specify the command line for an external SSH client which is 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, specify the user name and / or private key / passphrase in the command line. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example, on Windows the following command line uses the `plink.exe` tool as external SSH client for connecting to the SVN repository with SVN+SSH:

```
C:\plink-install-folder\plink.exe -l username -pw password -ssh -batch
host_name_or_IP_address_of_SVN_server
```

Messages Preferences

To open the **Messages** preferences page, go to **Options** > **Preferences** > **Messages**. This preferences page allows you to disable the following warning messages which may appear in the application:
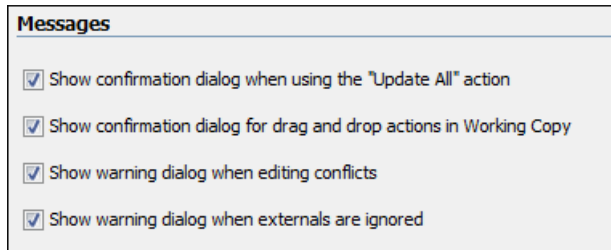
Figure 73: The Messages Preferences Panel

- **Show confirmation dialog when using the "Update All" action** - Allows you to avoid performing accidental update operations by requesting you to confirm them before execution.
- **Show confirmation dialog for drag and drop actions in Working Copy** - This option avoids doing a drag and drop when you just want to select multiple files in the Working Copy view.
- **Show warning dialog when editing conflicts** - When the **Edit Conflicts** action is executed, a warning dialog notifies you that the action overwrites the conflicted version of the file created by an update operation. The conflicted file is 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 conflicting file*.
- **Show warning dialog when "svn:externals" definitions are ignored** - A warning dialog is displayed when "svn:externals" definitions are ignored before performing any operation that updates resources of the working copy (like *Update* and *Override and Update*).

Reset Global Options

To reset all global preferences to their default values you have to go to menu **Options** > **Reset Global Options** > **Reset Global Options**.

Chapter

5

Upgrading Syncro SVN Client

Topics:

- *Upgrading Syncro SVN Client on Windows / Linux*
- *Upgrading Syncro SVN Client on Mac OS X*

This chapter presents the procedure you have to follow to upgrade Syncro SVN Client.

Upgrading Syncro SVN Client on Windows / Linux

The following steps describe the upgrading procedure of Syncro SVN Client, valid for Windows:

1. In case you added files, or modified files from the installation folder of Syncro SVN Client, save a copy of them. These files are overwritten during the upgrade.

    📄 **Note:** Custom frameworks are preserved, but we recommend you to back them up as well.

2. Run the new installation kit and install the new version in the folder where the old version was installed.
3. When you are promoted to choose whether you want to upgrade the existed installation, select **Proceed**. The old version is uninstalled automatically and the new one takes its place.
4. Copy the files you previously backed up in the installation folder.

Upgrading Syncro SVN Client on Mac OS X

The following steps describe the upgrading procedure of Syncro SVN Client, valid for Mac OS X:

1. Back up anything you need from the installation folder of Syncro SVN Client.
2. Remove the installation folder.
3. Unpack the archive with the new version.

    📄 **Note:** Do not unpack a new version of Syncro SVN Client on top of an older version. This causes problems because of used libraries that become duplicated.

4. Copy the backed-up files in the new installation folder.

Chapter

6

Common Problems

This chapter presents common problems that may appear when running the application and the solutions for these problems.

Syncro SVN Client Takes Several Minutes to Start on Mac

If Syncro SVN Client takes several minutes to start, the Java framework installed on the Mac may have a problem. One solution for this is to update Java to the latest version: go to **Apple symbol** > **Software Update**. After it finishes to check for updates, click **Show Details**, select the Java Update (if one is available) and click **Install**. If no Java updates are available, reset the Java preferences to their defaults. Start **Applications > Utilities > Java Preferences** and click **Restore Defaults**.

Special Characters Are Replaced With a Square in Editor

My file was created with other application and it contains special characters like é, ©, ®, etc. Why does Syncro SVN Client display a square for these characters when I open the file in Syncro SVN Client?

You must set a font able to render the special characters in the *Font preferences*. If it is a text file you must set also *the encoding used for non XML files.* If you want to set a font which is installed on your computer but that font is not accessible in the **Font** preferences that means the Java virtual machine is not able to load the system fonts, probably because it is not a True Type font. It is a problem of the Java virtual machine and a possible solution is to copy the font file in the `[JVM-home-folder]/lib/fonts` folder. `[JVM-home-folder]` is the value of the property *java.home* which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help** > **About**.

The Scroll Function of my Notebook's Trackpad is Not Working

I got a new notebook (Lenovo Thinkpad™ with Windows) and noticed that the scroll function of my trackpad is not working in Syncro SVN Client.

It is a problem of the Synaptics™ trackpads which can be fixed by adding the following lines to the `C:\Program Files\Synaptics\SynTP\TP4table.dat` file:

```
*,*,syncroSVNClient.exe,*,*,*,WheelStd,1,9
*,*,diffDirs.exe,*,*,*,WheelStd,1,9
*,*,diffFiles.exe,*,*,*,WheelStd,1,9
```

Grey Window on Linux With the Compiz / Beryl Window Manager

I try to run Syncro SVN Client on Linux with the Compiz / Beryl window manager but I get only a grey window which does not respond to user actions. Sometimes the Syncro SVN Client window responds to user actions but after opening and closing an Syncro SVN Client dialog or after resizing the Syncro SVN Client window or a view of the Syncro SVN Client window the content of this window becomes grey and it does not respond to user actions. What is wrong?

Sun Microsystems' Java virtual machine *does not support the Compiz window manager and the Beryl one very well*. It is expected that better support for Compiz / Beryl will be added in future versions of their Java virtual machine. You should turn off the special effects of the Compiz / Beryl window manager before starting the Syncro SVN Client application or switch to other window manager.

Set Specific JVM Version on Mac OS X

How do I configure Syncro SVN Client to run with the version X of the Apple Java virtual machine on my Mac OS X computer?

Syncro SVN Client uses the first JVM from the list of preferred JVM versions set on your Mac computer that has a version number 1.6.0 or higher. You can move your desired JVM version up in the preferred list by dragging it with the mouse on a higher position in the list of JVMs available in the **Java Preferences** panel that is opened from **Applications** > **Utilities** > **Java** > **Java Preferences**.

Segmentation Fault Error on Mac OS X

On my Mac OS X machine the application gives a *Segmentation fault* error when I double-click on the application icon. Sometimes it gives no error but it does not start. What is the problem?

Please make sure you have the latest Java update from the Apple website installed on your Mac OS X computer. If installing the latest Java update doesn't solve the problem please copy the file JavaApplicationStub from the /System/Frameworks/JavaVM.framework folder to the SyncroSVNClient.app/Contents/MacOS folder. For browsing the .app folder you have to **(CMD+click)** on the Syncro SVN Client icon and select **Show Package Contents**.

I Cannot Connect to SVN Repository From Repositories View

I cannot connect to a SVN repository from the **Repositories** view of SVN Client. How can I find more details about the error?

First check that you entered the correct URL of the repository in the **Repositories** 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 verified 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 folder 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.

Problem Report Submitted on the Technical Support Form

What details should I add to my problem report that I enter on the Technical Support online form of the product website?

For problems like server connection error, unexpected delay while editing a document, a crash of the application, etc for which the usual details requested on the Technical Support online form are not enough you should generate a log file and attach it to the problem report. In case of a crash you should also attach the crash report file generated by your operating system. For generating a logging file you need to create a text file called `log4j.properties` in the install folder 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 and close the application. The log file is called `logging.log` and is located in the install folder.

Index