Syncro SVN Client 10.1

Notice

Copyright

Contents

Chapter 1

Introduction

Topics:

Welcome to the User Manual of Syncro SVN Client 10.1!

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.

Key Features and Benefits of Syncro SVN Client

Multiplatform availability: Windows, OS X, Linux, Solaris	Multilanguage support: English, German, French, Italian and Japanese
Support for the Apache Subversion™ (SVN) versioning system version 1.8 and older.	Repository browser
Advanced working copy	Resource history
Directory change set	Revision graph
Detailed author information	Built-in text editors
Resolving conflicts	Bug tracking
Subversion properties	Check differences
OS X ready	Configurable actions key bindings

Chapter 2

Installation

Topics:

The platform requirements and installation instructions are presented in this chapter.

Installation Options for Syncro SVN Client

Choosing how Syncro SVN Client runs

You can install Syncro SVN Client to run in a number of ways:

- As a desktop application on *Windows*, *Linux*, or *Mac*.
- As a desktop application on a *Unix or Linux server* or on *Windows Terminal Server*.

Choosing an installer

You have a choice of installers;

- The native installer for your platform. On Windows and Linux, the native installer can run also in unattended mode.
- The All-platforms installer, which can be used on any supported platform.

The installation packages were checked before publication with an antivirus program to make sure they are not infected with viruses, trojan horses, or other malicious software.

Choosing a license option

You must *obtain and register a license* key to run Syncro SVN Client.

Upgrading, transferring, and uninstalling.

You can also *upgrade* Syncro SVN Client, *transfer a license*, or *uninstall* Syncro SVN Client.

Getting help with installation

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

Install Syncro SVN Client on Windows

Choosing an installer

You can install Syncro SVN Client on Windows using one of the following methods:

- Install using the *Windows installer*.
- Install using the *Windows installer in unattended mode*.
- Install using the *All Platforms installer*. Choose the all platforms installer if you have trouble installing using the Windows installer.

System Requirements

System requirements for a Windows install:

Operating systems

Windows XP, Windows Vista, Windows 7, Windows 8, Windows Server 2003, Windows Server 2008, Windows Server 2012

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 *GHz*
- Recommended - Dual Core class processor

Memory

- Minimum - 1 GB of RAM
- Recommended - 2 GB

Storage

- Minimum - 200 MB free disk space
- Recommended - 500 MB free disk space

Java

Syncro SVN Client requires Java. If you use the native Windows installer, Syncro SVN Client will be installed with its own copy of Java. If you use the all platforms installer, your system must have a compatible Java virtual machine installed.

Syncro SVN Client supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.7) from Oracle available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Syncro SVN Client may work with JVM implementations from other vendors, but there is no guarantee that those implementations will work with future Syncro SVN Client updates and releases.

Syncro SVN Client uses the following rules to determine which installed version of Java to use:

1. If you install using the native Windows installer, which installs a version of Java as part of the Syncro SVN Client installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Windows environment variable `JAVA_HOME` is set, Syncro SVN Client uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

If you run Syncro SVN Client using the batch file, `syncroSVNClient.bat`, you can edit the batch file to specify a particular version to use.

Install using the Windows installer

To install Syncro SVN Client using the Windows installer:

1. Make sure that your system meets the *system requirements*.
2. Download the Windows installer.
3. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
4. Run the installer and follow the instructions in the installation program.
5. Start Syncro SVN Client using one of the following methods:
   - Using one of the shortcuts created by the installer.
   - By running `syncroSVNClient.bat`, which is located in the install folder.
6. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Unattended Installation

You can run the installation in unattended mode by running the installer from the command line with the -q parameter. By default, running the installer in unattended mode installs Syncro SVN Client with the default options and does not overwrite existing files. You can change many options for the unattended installer using the *installer command line parameters*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`syncroSVNClient.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.

Syncro SVN Client is now installed in a new sub-folder called `syncroSVNClient`.

3. If you wish, you can move the directory where you installed Syncro SVN Client to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `syncroSVNClient`10.1.
4. Start Syncro SVN Client by running `syncroSVNClient.bat`, which is located in the install directory.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Install Syncro SVN Client on Mac OS X

Choosing an installer

You can install Syncro SVN Client on Mac OS X using one of the following methods:

- Install using the Mac OS X installation package, syncroSVNClient.zip.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Mac OS X archive installation.

System Requirements

System requirements for a Mac OS X install:

Operating system

Mac OS X version 10.5 64-bit or later

CPU

- Minimum - Intel-based Mac, 1 *GHz*
- Recommended - Dual Core class processor

Memory

- Minimum - 1 GB of RAM
- Recommended - 2 GB of RAM

Storage

- Minimum - 200 MB free disk space
- Recommended - 500 MB free disk space

Java

Syncro SVN Client requires Java to run. OS X includes Java by default or it will install it on the first attempt to run a Java application.

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 Apple). Syncro SVN Client may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Syncro SVN Client updates and releases.

Syncro SVN Client uses the following rules to determine which installed version of Java to use:

1. If you start oXygen with the application launcher (.app) file then:
   a. if you use the zip distribution for OS X Syncro SVN Client uses the Apple Java SE 6 available on your Mac computer
   b. if you use the tar.gz distribution that contains a bundled JRE then Syncro SVN Client will use that bundled JRE
2. If you start Syncro SVN Client using a startup .sh script then:
   a. if a bundled JRE is available then it will be used

  b. otherwise, if the `JAVA_HOME` environment variable is set then the Java distribution indicated by it will be used
  c. otherwise the version of Java pointed to by your `PATH` environment variable will be used

If you run Syncro SVN Client using the `syncroSVNClient.sh` script, you can change the version of Java used by editing to script file. 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:

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

OS X Installation

To install Syncro SVN Client on OS X:

1. Download the OS X installation package (syncroSVNClient.zip).

   The Safari web browser should recognize and expand the compressed file. If it is not automatically expanded, you can expand it manually by double-clicking it.
2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
3. In **Finder**, move the expanded folder to your `Applications` folder.
   Syncro SVN Client is now installed.
4. Start Syncro SVN Client, using one of the following methods:
   - Double click `Syncro SVN Client.app`.
   - Run `sh syncroSVNClientMac.sh` on the command line.

   Notice: You can start more than one instance on the same computer by running the following command for each new instance:

   ```
   open -n SyncroSVNClient.app
   ```
5. To license your copy of Syncro SVN Client, go to **Help** > **Register...** to enter your *license key*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`syncroSVNClient.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Syncro SVN Client is now installed in a new sub-folder called `syncroSVNClient`.
3. If you wish, you can move the directory where you installed Syncro SVN Client to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `syncroSVNClient`10.1.
4. Start Syncro SVN Client by running `syncroSVNClientMac.sh`, which is located in the install folder.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Install Syncro SVN Client on Linux

Choosing an installer

You can install Syncro SVN Client on Linux using any of the following methods:

- Install using the Linux installer.
- Install using the Linux installer in unattended mode.

- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Linux installer.

System Requirements

System requirements for a Linux install:

Operating system

Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.6.0 or later from Oracle

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 *GHz*
- Recommended - Dual Core class processor

Memory

- Minimum - 1 GB of RAM
- Recommended - 2 GB

Storage

- Minimum - 200 MB free disk space
- Recommended - 500 MB free disk space

Java

Syncro SVN Client requires Java. 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 available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Syncro SVN Client may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Syncro SVN Client updates and releases. Syncro SVN Client does not work with the GNU libgcj Java Virtual Machine.

Syncro SVN Client uses the following rules to determine which installed version of Java to use:

1. If you used the Linux installer, which installs a version of Java as part of the Syncro SVN Client installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Linux environment variable `JAVA_HOME` is set, Syncro SVN Client uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

You can also change the version of the Java Virtual Machine that runs Oxygen XML Author by editing the script file, `syncroSVNClient.sh`. 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:

```
/usr/bin/jre1.6.0_45/bin/java -Xmx256m ...
```

Linux Installation

Linux installation procedure.

To install Syncro SVN Client on Linux:

1. Download the Linux installer.
2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
3. Run the installer that you downloaded and follow the instructions presented in the installation program.
4. Start Syncro SVN Client using one of the following methods:
   - Use the `syncroSVNClient` shortcut created by the installer.
   - Run sh syncroSVNClient.sh from the command line. This file is located in the installation folder.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license key*.

Unattended Installation

You can run the installation in unattended mode by running the installer from the command line with the -q parameter. By default, running the installer in unattended mode installs Syncro SVN Client with the default options and does not overwrite existing files. You can change many options for the unattended installer using the *installer command line parameters*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`syncroSVNClient.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Syncro SVN Client is now installed in a new sub-folder called `syncroSVNClient`.
3. If you wish, you can move the directory where you installed Syncro SVN Client to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `syncroSVNClient`10.1.
4. Start Syncro SVN Client by running `syncroSVNClient.sh`, which is located in the install folder.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Installing Syncro SVN Client on Windows Server

Choosing an installer

You can install Syncro SVN Client on Windows using one of the following methods:

- Install using the *Windows installer*.
- Install using the *Windows installer in unattended mode*.
- Install using the *All Platforms installer*. Choose the all platforms installer if you have trouble installing using the Windows installer.

System Requirements

System requirements for a Windows Server install:

Operating systems

Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 *GHz*
- Recommended - Dual Core class processor

Memory

- Minimum values per user - 512 MB of RAM
- Recommended values per user - 512 MB of RAM

Storage

- Minimum - 200 MB free disk space
- Recommended - 500 MB free disk space

Java

Syncro SVN Client requires Java. If you use the native Windows installer, Syncro SVN Client will be installed with its own copy of Java. If you use the all platforms installer, your system must have a compatible Java virtual machine installed.

Syncro SVN Client supports only official and stable Java Virtual Machines with the version number 1.6.0 or later (the recommended version is 1.7) from Oracle available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Syncro SVN Client may work with JVM implementations from other vendors, but there is no guarantee that those implementations will work with future Syncro SVN Client updates and releases.

Syncro SVN Client uses the following rules to determine which installed version of Java to use:

1. If you install using the native Windows installer, which installs a version of Java as part of the Syncro SVN Client installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Windows environment variable `JAVA_HOME` is set, Syncro SVN Client uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

If you run Syncro SVN Client using the batch file, `syncroSVNClient.bat`, you can edit the batch file to specify a particular version to use.

Install using the Windows installer

To install Syncro SVN Client using the Windows installer:

1. Make sure that your system meets the *system requirements*.
2. Download the Windows installer.
3. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
4. Run the installer and follow the instructions in the installation program.
5. Start Syncro SVN Client using one of the following methods:
   - Using one of the shortcuts created by the installer.
   - By running `syncroSVNClient.bat`, which is located in the install folder.
6. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`syncroSVNClient.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Syncro SVN Client is now installed in a new sub-folder called `syncroSVNClient`.
3. If you wish, you can move the directory where you installed Syncro SVN Client to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `syncroSVNClient`10.1.
4. Start Syncro SVN Client by running `syncroSVNClient.bat`, which is located in the install directory.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Configuring Windows Terminal Server

Windows Terminal Server configuration procedure.

1. Install Syncro SVN Client on the server and make its shortcuts available to all users.
2. If you need to run multiple instances of Syncro SVN Client, make sure you add the *-Dcom.oxygenxml.MultipleInstances=true* parameter in the `.bat` startup script.
3. Make sure you allocate sufficient memory to Syncro SVN Client by adding the `-Xmx` parameter either in the `.bat` startup script, or in the `.vmoptions` configuration file (if you start it from an executable launcher).

Installing Syncro SVN Client on a Linux / UNIX Server

Choosing an installer

You can install Syncro SVN Client on Linux using any of the following methods:

- Install using the Linux installer.
- Install using the Linux installer in unattended mode.
- Install using the all platforms installer. Choose the all platforms installer if you have trouble installing using the Linux installer.

System Requirements

System requirements for a Linux install:

Operating system

Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.6.0 or later from Oracle

CPU

- Minimum - Intel Pentium III™/AMD Athlon™ class processor, 1 *GHz*
- Recommended - Dual Core class processor

Memory

- Minimum - 1 GB of RAM
- Recommended - 2 GB

Storage

- Minimum - 200 MB free disk space
- Recommended - 500 MB free disk space

Java

Syncro SVN Client requires Java. 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 available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*. Syncro SVN Client may work with JVM implementations from other vendors, but there is no guarantee that other implementations will work with future Syncro SVN Client updates and releases. Syncro SVN Client does not work with the GNU libgcj Java Virtual Machine.

Syncro SVN Client uses the following rules to determine which installed version of Java to use:

1. If you used the Linux installer, which installs a version of Java as part of the Syncro SVN Client installation, the version in the `jre` subdirectory of the installation directory is used.
2. Otherwise, if the Linux environment variable `JAVA_HOME` is set, Syncro SVN Client uses the Java version pointed to by this variable.
3. Otherwise the version of Java pointed to by your PATH environment variable is used.

You can also change the version of the Java Virtual Machine that runs Oxygen XML Author by editing the script file, `syncroSVNClient.sh`. 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:

```
/usr/bin/jre1.6.0_45/bin/java -Xmx256m ...
```

Linux Installation

Linux installation procedure.

To install Syncro SVN Client on Linux:

1. Download the Linux installer.
2. Validate the integrity of the downloaded file by *checking it against the MD5 sum* published on the download page.
3. Run the installer that you downloaded and follow the instructions presented in the installation program.
4. Start Syncro SVN Client using one of the following methods:
   - Use the `syncroSVNClient` shortcut created by the installer.
   - Run sh syncroSVNClient.sh from the command line. This file is located in the installation folder.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license key*.

Install using the all platforms installer

To install using the all platforms installer:

1. Download the all platforms installation package (`syncroSVNClient.tar.gz`) to a folder of your choice.
2. Extract the archive in that folder.
   Syncro SVN Client is now installed in a new sub-folder called `syncroSVNClient`.
3. If you wish, you can move the directory where you installed Syncro SVN Client to your applications directory. You can also rename it to contain the product version information. For example you can rename it as `syncroSVNClient`10.1.
4. Start Syncro SVN Client by running `syncroSVNClient.sh`, which is located in the install folder.
5. To license your copy of Syncro SVN Client go to **Help** > **Register...** and enter your *license information*.

Unix / Linux Server Configuration

To install Syncro SVN Client on a Unix / Linux server:

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 application.
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. Start 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`.

Site-wide Deployment

If you are deploying Syncro SVN Client for a group, there are a number of things you can do to customize Syncro SVN Client for your users and to make the deployment more efficient.

Creating custom default options

You can *create a custom set of default options* for Syncro SVN Client. These will become the default options for each of your users, replacing Syncro SVN Client's normal default settings. Users can still set options to suit themselves in their own copies of Syncro SVN Client, but if they choose to reset their options to defaults, the custom defaults that you set will be used.

Creating default project files

Syncro SVN Client project files are used to configure a project. You can create and deploy default project files for your projects so that your users will have a preconfigured project file to begin work with.

Shared project files

Rather than each user having their own project file, you can create and deploy shared project files so that all users share the same project configuration and settings and automatically inherit all project changes.

Using the unattended installer

You can speed up the installation process by using the *unattended installer for Windows* or *Linux installs*.

Using floating licenses

If you have a number of people using Syncro SVN Client on a part-time basis or in different time zones, you can use a *floating license* so that multiple people can share a license.

Obtaining and Registering a License Key for Syncro SVN Client

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 *http://www.syncrosvnclient.com/register.html*. 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 functionality. To continue using the software after the trial period, you must purchase a permanent license. If a trial period greater than 30 days is required, please contact support@syncrosvnclient.com.

Choosing a license type

You can use one of the following license types with Syncro SVN Client:

1. A named-user license may be used by a single named user on one or more computers. Named-user licenses are not transferable to a new named user. If you order multiple named-user licenses, you will receive a single license key good for a specified number of named users. It is your responsibility to keep track of the named users that each license is assigned to.
2. A site license may be used by all the people that belong to the organization that purchased the site license.

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

Obtaining a license

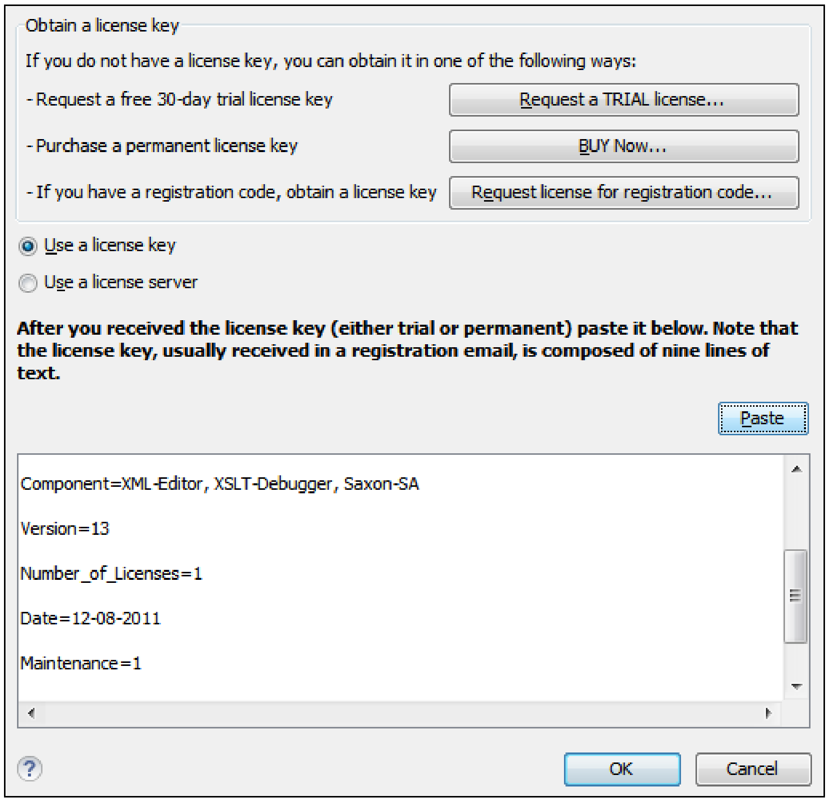
You can obtain a license for Syncro SVN Client in one of the following ways:

- You can purchase one or more licenses from the Syncro SVN Client website at *http://www.syncrosvnclient.com/buy.html*. A license key will be sent to you by email.
- If your company or organization has purchased licenses please contact your license administrator to obtain a license key.
- If you want to evaluate the product you can obtain a trial license key for 30 days from the Syncro SVN Client website at *http://www.syncrosvnclient.com/register.html*.

Register a named-user license

To register a named-user license on a machine owned by the named user:

1. Save a backup copy of the message containing the new license key.
2. Start Syncro SVN Client.
   If this is a new install of Syncro SVN Client, the registration dialog is displayed. If the registration dialog is not displayed, go to **Help** > **Register...**.


Obtain a license key
If you do not have a license key, you can obtain it in one of the following ways:
- Request a free 30-day trial license key
Request a TRIAL license...
- Purchase a permanent license key
BUY Now...
- If you have a registration code, obtain a license key
Request license for registration code...
Use a license key
Use a license server
Paste
OK
Cancel

After you received the license key (either trial or permanent) paste it below. Note that the license key, usually received in a registration email, is composed of nine lines of text.

```
Component=XML-Editor, XSLT-Debugger, Saxon-SA

Version=13

Number_of_Licenses=1

Date=12-08-2011

Maintenance=1
```

Figure 1: License Registration Dialog

3. Paste the license text into the registration dialog.
4. Press **OK**.

Register Multiple Licenses

If you are installing a named-user license on multiple machines, or you are an administrator registering named-user or floating licenses for multiple users, you can avoid having to open Syncro SVN Client on each machine by registering the license using a text file or XML file that contains the license information.

Note: If you are using floating licenses that are managed by a license server, you cannot use this method to register licenses.

To register licenses using a text file:

1. Copy the license key to a file named `licensekey.txt` and place it in the

To register licenses using an XML file:

1. Register the license on one computer using the normal *license registration procedure*.
2. Copy the `license.xml` file from the Syncro SVN Client *preferences directory* on that computer to the installation folder on each installation to be registered.

Transferring or Releasing a License Key

If you want to transfer your Syncro SVN Client license key to another computer (for example if you are disposing of your old computer or transferring it to another person), you must first unregister your license. You can then *register your license* on the new computer in the normal way.

1. Go to **Help** > **Register...**.
   The license registration dialog is displayed.

2. The license key field should be empty (this is normal). If it is not empty, delete any text in the field.
3. Make sure the option **Use a license key** is selected.
4. Click **OK**.
   A dialog is displayed asking if your want to reset your license key.
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 on 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 fix problems, and add new features.

Checking for New Versions of Syncro SVN Client

Syncro SVN Client checks for new versions automatically at start up. To disable this check, *open the **Preferences** dialog* , go to **Global**, and uncheck **Automatic Version Checking**.

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

Upgrading the Standalone Application

1. 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.
2. Download and install the new version according to the instructions for your platform and the type of installer you selected.
3. If you installed from an archive (as opposed to an executable installer) you may have to update any shortcuts you have created or modify the system PATH to point to the new installation folder.
4. Start Syncro SVN Client.
5. If you require an new license for your upgrade, install it now according to the procedure for your platform and the type of installer you selected.

Uninstalling Syncro SVN Client

Uninstalling the Syncro SVN Client Standalone

Caution: The following procedure will remove Syncro SVN Client from your system. **All data stored in the installation directory will be removed, including any customizations or any other data you have stored within that directory. Please make a back up of any data you want to keep before uninstalling Syncro SVN Client.**

1. Backup any data you want to keep from the Syncro SVN Client installation folder.
2. Remove the application.
   - On Windows use the appropriate uninstaller shortcut provided with your OS.
   - On OS X and Unix manually delete the installation folder and all its contents.
3. If you want to remove the user preferences:
   - **On Windows**, remove the directory:`%APPDATA%\com.syncrosvnClient` (usually %APPDATA% has the value `[user-home-dir]\Application Data`).

- **On Linux**, remove the directory: `.com.syncrosvnClient` from the user's home directory.
- **On Mac OS X**, remove the directory: `Library/Preferences/com.syncrosvnClient` of the user home folder .

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.

Syncro SVN Client Installer Command Line Reference

Command line options of the Syncro SVN Client installer.

The `response.varfile`

The Syncro SVN Client installers for Windows and Linux creates a file called `response.varfile`, which records the choices that the user made when running the installer interactively. You can use a `response.varfile` to set the options for an unintended install. Here is an example of a `response.varfile`:

```
#install4j response file for Oxygen XML Editor 16.0
#Fri Jul 18 21:52:15 EDT 2014
sys.adminRights$Boolean=true
sys.programGroupDisabled$Boolean=false
createDesktopLinkAction$Boolean=true
autoVersionChecking=true
sys.fileAssociation.launchers$StringArray="19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19","19"
sys.programGroupAllUsers$Boolean=true
reportProblem=true
downloadResources=true
sys.languageId=en
sys.installationDir=C\:\\Program Files (x86)\\Oxygen XML Editor 16
createQuicklaunchIconAction$Boolean=true
sys.programGroupName=Oxygen XML Editor 16.0
executeLauncherAction$Boolean=true
sys.fileAssociation.extensions$StringArray="xml","dita","ditamap","ditaval","xsl","xslt","xspec","xsd","rng","rnc","sch","dtd","mod","ent","nvdl","fo","wsdl","xquery","xq","xqy","xqm","xql","xpl","css","json","xpr"
```

The following table describes some of the settings that can be used in the `response.varfile`:

Table 1: `response.varfile` 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`.
`downloadResources`	Allows Syncro SVN Client to download resources (links to video demonstrations, webinars and upcoming events) from *http://www.syncrosvnclient.com* to populate the application welcome screen.	`true` / `false`. Default setting is `true`.

The Syncro SVN Client installation uses the install4j installer. A description of *the `response.varfile` format* can be found on the install4j site.

Command line parameters

The Syncro SVN Client installer supports the following command line parameters:

Option	**Meaning**

`-q`	Run the installer in unattended mode. The installer will not prompt the user for input during the install. Default settings will be used for all options unless a `response.varfile` is specified using the `-varfile` option or individual settings are specified using
	- on Windows:
	`syncroSVNClient.exe -q`
	- on Linux:
	`syncroSVNClient.sh -q`

`-overwrite`	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.
	- on Windows:
	`syncroSVNClient.exe -q -overwrite`
	- on Linux:
	`syncroSVNClient.sh -q -overwrite`

`-console`	To display a console for the unattended installation, add a -console parameter to the command line.
	- on Windows:
	`start /wait syncroSVNClient.exe -q -console`
	Note: The use of `start /wait` on Windows is required to make the installer run in the foreground. It you run it without `start /wait`, it will run in the background.
	- on Linux:
	`syncroSVNClient.sh -q -console`

`-varfile`	Points to the location of a `response.varfile` to be used during an unattended installation. For example:
	- on Windows:
	`syncroSVNClient.exe -q -varfile response.varfile`
	- on Linux:
	`syncroSVNClient.sh -q -varfile response.varfile`

`-V`	Is used to define a variable to be used by an unattended installation. For example:
	- on Windows:
	`syncroSVNClient.exe -q -VusageDataCollector=false`
	- on Linux:
	`syncroSVNClient.sh -q -VusageDataCollector=false`

The Syncro SVN Client installation uses the install4j installer. A full list of the *command line parameters supported by the install4j installer* can be found on the install4j site.

Chapter 3

Tools

Topics:

Syncro SVN Client provides along with the SVN support also a comparing and merging solution.

SVN Client

The Syncro SVN Client is a client application for the Apache Subversion™ version control system, compatible with Subversion 1.6, 1.7 and 1.8 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.

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™ 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 new file as a child of the selected folder from the ***Repositories** view* tree or the ***Working Copy** view* tree, depending on the view that was last used. Note that for the ***Working Copy** view*, the file is added to *version control* only if the selected folder is under *version control*.

  New Folder... (Ctrl (Command on OS X) + Shift + F)

  This operation creates a new folder as a child of the selected folder from the ***Repositories** view* tree or the ***Working Copy** view* tree, depending on the view that was last used. Note that for the ***Working Copy** view*, the file is added to *version control* only if the selected folder is under *version control*.

  New External Folder... (Ctrl (Command on OS X) + Shift + W)

  This operation allows you to add a new external definition on the selected folder. An external definition is a mapping of a local directory to *a URL of a versioned directory*, and ideally a particular revision, stored in the `svn:externals` property of the selected folder.

  Tip: You can specify a particular revision of the external item by using a *peg revision* at the end of the URL (for example, `URL@rev1234`). You can also use peg revisions to access external items that were deleted, moved, or replaced.

The URL used in the external definition format can be relative. You can specify the repository URL that the external folder points to by using one of 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 in which the `svn:externals` property is versioned.

Important: To change the target URL of an external definition, or to delete an external item, do the following:

1. Modify or delete the item definition found in the `svn:externals` property that is set on the parent folder.
2. For the change to take effect, use the **Update** operation on the parent folder of the external item.

Note: Syncro SVN Client does not support definitions of local relative external items.

Open (Ctrl (Command on OS X) + 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. This action works on any file selection from the ***Repositories*** *view*, ***Working Copy*** *view*, ***History*** *view*, or ***Directory Change Set*** *view*, depending on the view that was last used to invoke it. In the case of a folder, the action opens the selected folder with the system application for folders (for example, Windows Explorer on Windows or Finder on OS X, etc). Note that opening folders is available only for folders selected in the ***Working Copy*** *view*.

Open with... (Ctrl (Command on OS X) + Shift + O)

Displays the **Open with** dialog for specifying the editor in which the selected file is opened. If 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 the view that was last used to invoke it.

Show in Explorer/Show in Finder

Opens the parent directory of the selected working copy file and selects the file.

Save (Ctrl (Command on OS X) + S)

Saves the local file currently opened in the editor or the **Compare** view.

Save as...

Saves any file selected in the **Repositories**, **History**, or **Directory Change Set** view.

Copy URL Location (Ctrl (Command on OS X) + Alt + U)

Copies the URL location of the resource currently selected in the **Repositories** view to clipboard.

Copy to...

Copies the currently selected resource, either in **Repositories** or **Working copy** view, to a specified location.

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 (Command on OS X) + M)

Moves the currently selected resource, either in **Repositories** or **Working copy** view, to a specified location.

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 (Command on OS X) + 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. This is only active for resources under *version control*. For more details see *Scanning for locks*.
- **Lock... (Ctrl (Command on OS X) + K)** - Allows you to lock certain files that need exclusive access. You can write a comment describing the reason for the lock and you can also force (*steal*) the lock. This action is active only on files under *version control*. For more details on the use of this action see *Locking a file*.
- **Unlock... (Ctrl (Command on OS X) + 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 (Command on OS X) + P)

Opens the ***Properties** view* and displays the SVN properties for a selected resource from ***Repositories** view* or ***Working Copy** view*, depending on the view that was last used to invoke it.

Show SVN Information (Ctrl (Command on OS X) + I)

Provides additional information for a selected resource. For more details, go to *Obtain information for a resource*.

Exit (Ctrl (Command on OS X) + Q)

Closes the application.

- **Edit** menu:

Undo (Ctrl (Command on OS X) + Z)

Undo edit changes in the local file that is currently opened in the editor or the **Compare** view.

Redo (Ctrl (Command on OS X) + Y)

Redo edit changes in the local file that is currently opened in the editor or the **Compare** view.

Cut (Ctrl (Command on OS X) + X)

Cut selection from the local file that is currently opened in the editor view or the **Compare** view to clipboard.

Copy (Ctrl (Command on OS X) + C)

Copy selection from the local file that is currently opened in the editor or the **Compare** view to clipboard.

Paste (Ctrl (Command on OS X) + V)

Paste selection from clipboard into the local file that is currently opened in editor or the **Compare** view.

Find/Replace (Ctrl (Command on OS X) + F)

Perform find and replace operations in the local file that is currently opened in the editor or the **Compare** view.

Find Next (F3)

Go to the next match using the same find options of the last find operation. This 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 match using the same find options of the last find operation. This action runs in the editor panel and in any non-editable text area (for example, the **Console** view).

- **Repository** menu:

New Repository Location... (Ctrl+Alt+N (Command+Alt+N on OS X))

Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.

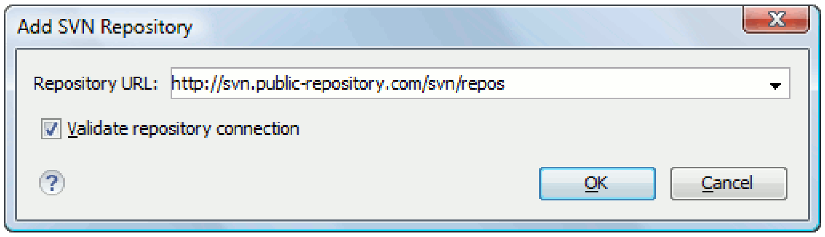

Add SVN Repository
Repository URL: http://svn.public-repository.com/svn/repos
Validate repository connection
OK
Cancel

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+Alt+E (Command+Alt+E on OS X))

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+Alt+Shift+B (Command+Alt+Shift+B on OS X))

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+Alt+Shift+R (Command+Alt+Shift+R on OS X))

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 (F5)

Refreshes the resource selected in the **Repositories** view.

Check out... (Ctrl+Alt+Shift+C (Command+Alt+Shift+C on OS X))

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...

Opens *the **Export** dialog box* that allows you to configure options for exporting a folder from the repository to the local file system.

Import:

Import folder... (Ctrl+Alt+Shift+M (Command+Alt+Shift+M on OS X))

Allows you to import the contents of a specified folder from the file system into the selected folder in a repository. To read more about this operation, see the section *Importing resources into a repository*.

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

Import Files... (Ctrl+Alt+I (Command+Alt+I on OS X))

Imports the files selected from the files system into the selected folder in the repository.

- **Working Copy** menu:

(on OS X) Working Copies Manager

Opens a dialog with a list of working copies that the Apache Subversion™ client is aware of. In this dialog you can add existing working copies or remove those that are no longer needed.

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 (Command on OS X) + 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 (Command on OS X) + 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 resources to commit. A directory will always be committed recursively. Unversioned resources will be deselected by default. In the **Commit** dialog you can also enter a comment before sending your changes to the repository.

Update all... (Ctrl (Command on OS X) + 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 (Command on OS X) + Shift + V)

Undoes all local changes for the selected resources. It does not contact the repository and the files are obtained from Apache Subversion™ pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.

Edit conflict... (Ctrl (Command on OS X) + 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 (Command on OS X) + Shift + R)

Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.

Mark as Merged (Ctrl (Command on OS X) + 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. This action is available on resources with outgoing changes, including 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. This action is available on conflicting resources. For more information see *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 item and source item), and it depends on the state of the source item.

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 item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

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 item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

Add to "svn:ignore"... (Ctrl (Command on OS X) + Alt + I)

Allows you to add files that should not participate in the *version control* operations inside your working copy . This action can only be performed on resources not under *version control*. It actually modifies the value of the `svn:ignore` property in the parent directory of the resource. Read more about this in the *Ignore Resources Not Under Version Control* section.

Add to version control... (Ctrl (Command on OS X) + Alt + V)

Allows you to add 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 (Command on OS X) + Shift + C)

Performs a maintenance cleanup operation on the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. This is useful when you already know where the problem originated and want to fix it as quickly as possible. It is only active for resources under *version control*.

Expand all (Ctrl (Command on OS X) + Alt + X)

Displays all descendants of the selected folder. The same behavior is obtained by double-clicking on a collapsed folder.

Collapse all (Ctrl (Command on OS X) + 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 file and target file.

Next Block of Changes (Ctrl+Period (Command+Period on OS X))

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.

Note: A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes (Ctrl+Comma (Command+Comma on OS X))

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.

Next Change (Ctrl+Shift+Period (Command+Shift+Period on OS X))

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 or when there are no changes.

Previous Change (Ctrl+Shift+Comma (Command+Shift+Comma on OS X))

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 or when there are no changes.

Last Change (Ctrl+E (Command+E on OS X))

Jumps to the last change.

First Change (Ctrl+B (Command+B on OS X))

Jumps to the first change.

Copy All Non-Conflicting Changes from Right to Left

This action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.

Copy Change from Right to Left

This action copies the selected change from the right editor to the left editor.

Show 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.

Format and Indent Both Files (Ctrl+Shift+P (Command+Shift+P on OS X))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

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 (Command on OS X) + 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+Shift+A (Command+Shift+A on OS X))

  Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the **Annotations** view*, along with the history of the file in the **History** view.

  Repositories

  This operation is available for any resource selected from 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 (Command on OS X) + 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 (Command on OS X) + 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 (Command on OS X) + 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 for *external* items. To read more about this action, see the *Switching the Repository Location* section.

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, if the repository itself was moved to a different server. This operation is only available for the root item of the working copy. To read more about this operation, see the *Relocate a Working Copy* section.

Create patch... (Ctrl (Command on OS X) + 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

Upgrades the format of the currently loaded working copy to the newest one known by Syncro SVN Client. This allows you to benefit of all the new features of the client.

Downgrade

Downgrades the format of the currently loaded working copy to SVN 1.7 format. 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 to SVN 1.8.

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.

  Use online help (Enabled by default)

  If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Syncro SVN Client attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

  Show Dynamic Help view

  Shows the **Dynamic Help** view.

  Check for a New Version

  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.

  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 Syncro 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.

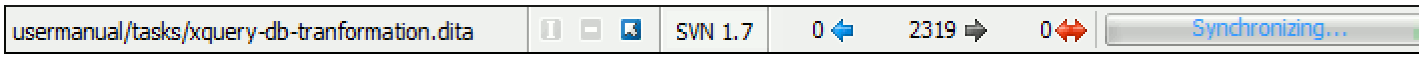


Figure 3: Status bar

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 then must share their work by committing their changes. This is done using an Apache Subversion™ repository. Syncro SVN Client supports versions 1.4, 1.5, 1.6, 1.7, and 1.8 of the SVN repository format.

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

To create a repository location, use the **New Repository Location...** action that is available in the **Repository** menu, the **Repositories** view toolbar, and in the contextual menu. This action opens the **New Repository Location** dialog box, which prompts you for the *URL of the repository* you want to connect to. You can also *use peg revisions at the end of the URLs* (for example, `URL@rev1234`) to browse only that specific revision. 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 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 to 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 the preferences, an error is displayed. If you do not want to wait until the timeout expires, you can use the **Stop** button from the toolbar of the view.

To edit a repository location, use the **Edit Repository Location...** action that is available in the **Repository** menu and in the contextual menu. This action opens the **Edit Repository Location** dialog box, which prompts you for the *URL of the repository* you want to connect to. You can also *use peg revisions at the end of the URLs* (for example, `URL@rev1234`) to browse only that specific revision.

To remove a repository location, use the **Remove Repository Location...** action that is available in the **Repository** menu and in the contextual menu. A confirmation dialog is displayed to make sure that you do not accidentally remove the wrong locations.

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

To set the reference revision number of an SVN repository use the **Change the Revision to Browse...** action that is available in the **Repository** menu and in the contextual menu. 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 in the repository are displayed as contents of the repository tree. Also, this revision number is used for all the 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 OS X - `$HOME/.subversion/auth`. Example: `/home/John/.subversion/auth`

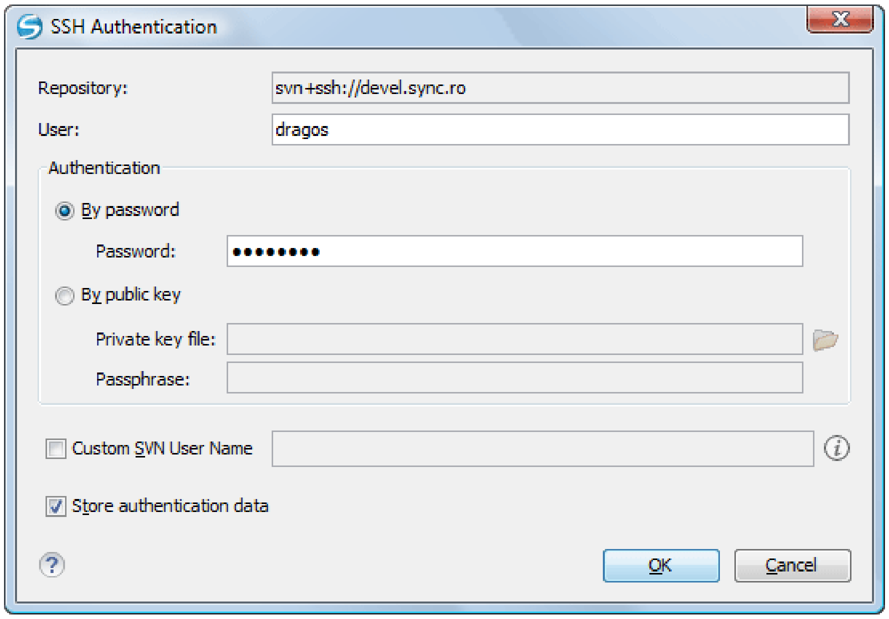
There is one file for each server that you access. If you want to make Subversion forget your credentials, you can use the **Reset authentication** command from the **Options** menu. This causes Subversion to forget all your credentials. When you reset the authentication data, restart Syncro SVN Client 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.



SSH Authentication

Repository: svn+ssh://devel.sync.ro

User: dragos

Authentication

(●) By password

Password: ●●●●●●●●●

( ) By public key

Private key file:

Passphrase:

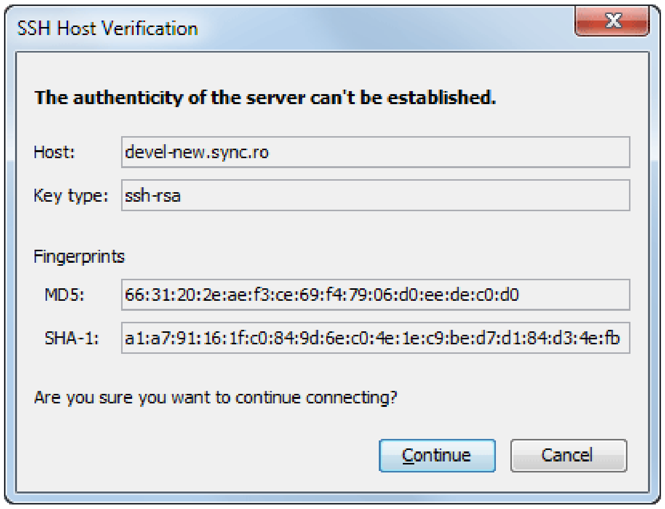
[ ] Custom SVN User Name

[✓] Store authentication data

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.


SSH Host Verification

The authenticity of the server can't be established.

Host: devel-new.sync.ro

Key type: ssh-rsa

Fingerprints

MD5: 66:31:20:2e:ae:f3:ce:69:f4:79:06:d0:ee:de:c0:d0

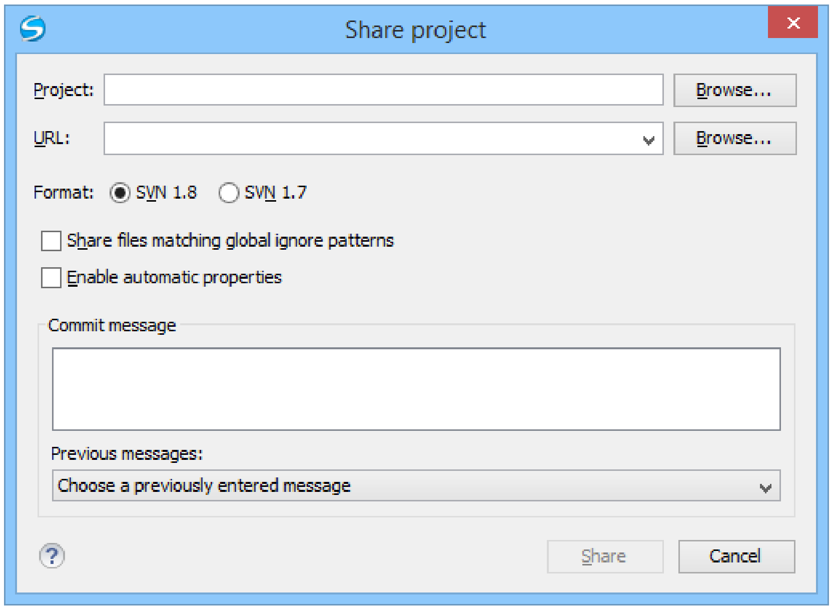
SHA-1: a1:a7:91:16:1f:c0:84:9d:6e:c0:4e:1e:c9:be:d7:d1:84:d3:4e:fb

Are you sure you want to continue connecting?

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, the Syncro SVN Client allows you to easily share it with the rest of your team. The shared project directory is automatically converted to a working copy and added under Syncro SVN Client management. The **Share project...** action is available in the **Tools** menu and the contextual menu of the **Repositories** view.


Share project

Project:

URL:

Format: [✓ SVN 1.8] SVN 1.7

☐ Share files matching global ignore patterns

☐ Enable automatic properties

Commit message

Previous messages: Choose a previously entered message

Figure 6: Share Project Dialog Box

The following options can be configured in the **Share project** dialog box:

Project

The location of the project folder on the local disk by using the text box or the **Browse** button. This folder should not be empty or already under version control.

Important: By default, the SVN system only imports the content of the specified folder, and not the root folder itself. Therefore, it is recommended to use the **Browse** button to select the project folder so that the client will automatically append the name of it to the specified URL.

URL

The new location of the project (inside the repository) that will be used to access it.

Note: *Peg revisions* have no effect for this operation since it is used to send information to the repository.

Attention: If the new location already exists, make sure that it is an empty directory to avoid mixing your project content with other files (if items exist with the same name, an error will occur and the operation will not proceed). Otherwise, if the address does not exist, it is created automatically.

Format

The SVN format of the working copy. You can choose between **SVN 1.8** or **SVN 1.7**.

Share files matching global ignore patterns

When enabled, the file names that match the patterns defined in either of the following locations are also imported into the repository:

- The `global-ignores` property in *the SVN configuration file*.
- The ***File name ignore patterns*** *option* in the ***SVN*** *>* ***Working Copy*** *preferences page*.

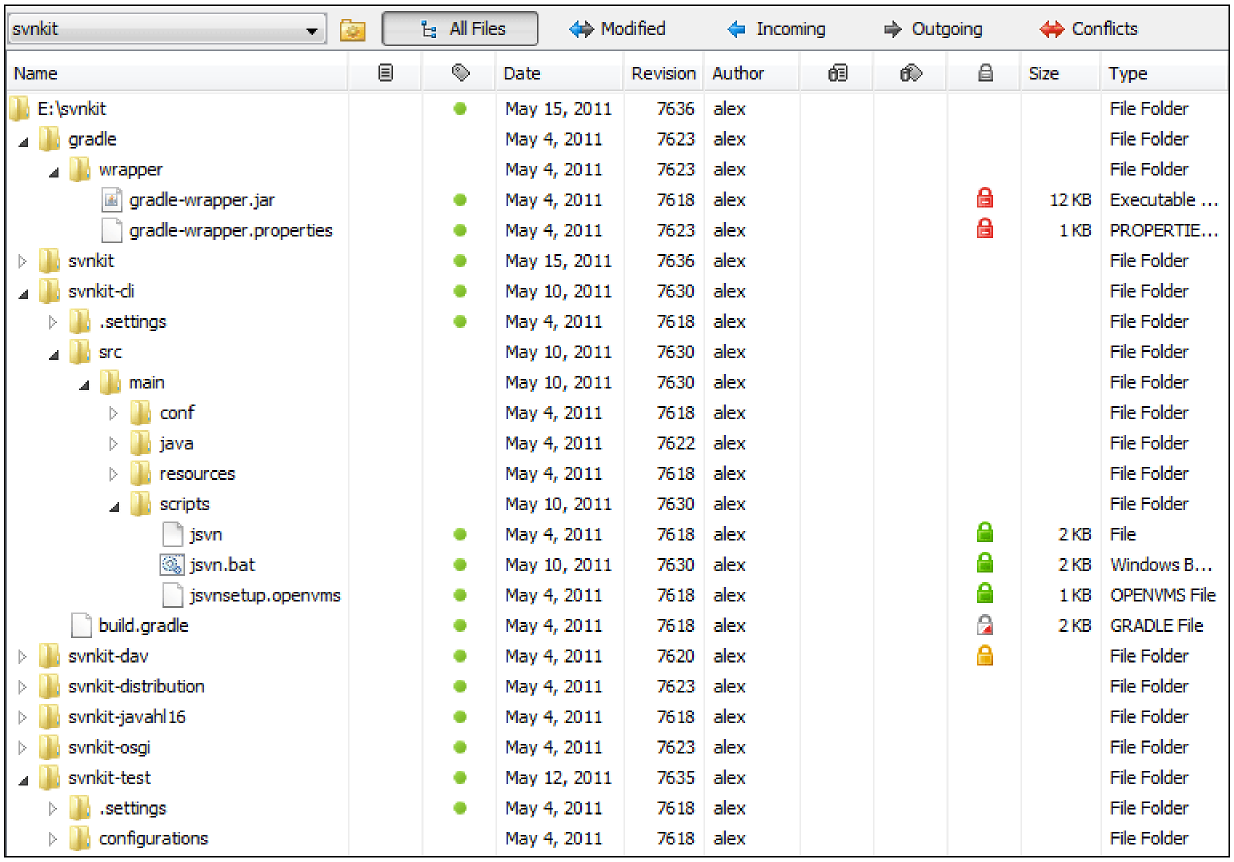
Enable automatic properties/Disable automatic properties

Enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in *the SVN configuration file*.

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

Defining a Working Copy

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



Name			Date	Revision	Author				Size	Type
E:\svnkit		●	May 15, 2011	7636	alex					File Folder
gradle			May 4, 2011	7623	alex					File Folder
wrapper			May 4, 2011	7623	alex					File Folder
gradle-wrapper.jar		●	May 4, 2011	7618	alex			🔒	12 KB	Executable ...
gradle-wrapper.properties		●	May 4, 2011	7623	alex			🔒	1 KB	PROPERTIE...
svnkit		●	May 15, 2011	7636	alex					File Folder
svnkit-cli		●	May 10, 2011	7630	alex					File Folder
.settings		●	May 4, 2011	7618	alex					File Folder
src			May 10, 2011	7630	alex					File Folder
main			May 10, 2011	7630	alex					File Folder
conf			May 4, 2011	7618	alex					File Folder
java			May 4, 2011	7622	alex					File Folder
resources			May 4, 2011	7618	alex					File Folder
scripts			May 10, 2011	7630	alex					File Folder
jsvn		●	May 4, 2011	7618	alex			🔒	2 KB	File
jsvn.bat		●	May 10, 2011	7630	alex			🔒	2 KB	Windows B...
jsvnsetup.openvms		●	May 4, 2011	7618	alex			🔒	1 KB	OPENVMS File
build.gradle		●	May 4, 2011	7618	alex			🔒	2 KB	GRADLE File
svnkit-dav		●	May 4, 2011	7620	alex			🔒		File Folder
svnkit-distribution		●	May 4, 2011	7623	alex					File Folder
svnkit-javahl16		●	May 4, 2011	7618	alex					File Folder
svnkit-osgi		●	May 4, 2011	7623	alex					File Folder
svnkit-test		●	May 12, 2011	7635	alex					File Folder
.settings		●	May 4, 2011	7618	alex					File Folder
configurations			May 4, 2011	7618	alex					File Folder

Figure 7: Working Copy View

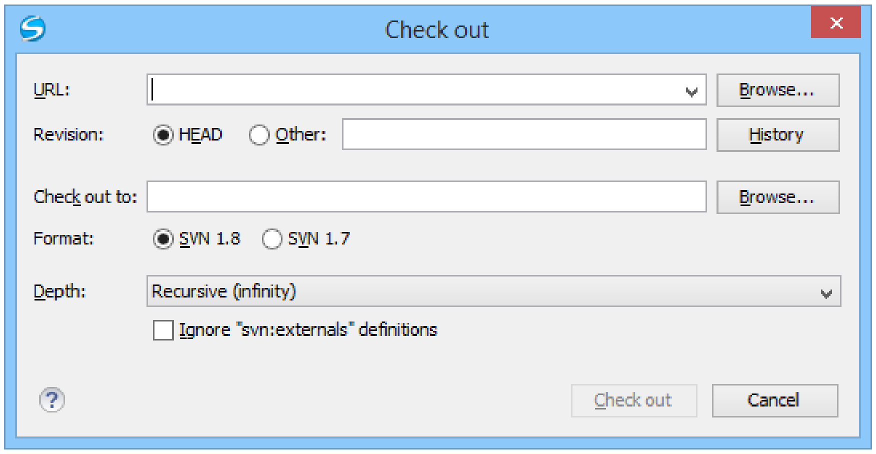
A Subversion working copy also contains some extra files, created and maintained by Subversion, to help it keep track of your files. In particular, each directory in your working copy contains a subdirectory named `.svn`, also known as the working copy *administrative directory*. This administrative 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 sub-tree of the repository.

Check Out a Working Copy

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

To check out a working copy, locate and select the desired directory in the **Repositories** view and select the **Check out...** action from the contextual menu, the toolbar, or the **Repository** menu.



Revision: [✓ HEAD] Other:
Check out to:
Format: [✓ SVN 1.8] SVN 1.7
Depth: Recursive (infinity)
☐ Ignore "svn:externals" definitions

Figure 8: Check Out Dialog

The following options can be configured in the **Check out** dialog box:

URL

The location of the repository directory to be *checked out*.

Note: To check out an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, `URL@rev1234`).

Revision

You can choose between the **HEAD** or **Other** revision. If you need to *check out* a specific revision, specify it in the **Other** text box or use the **History** button and choose a revision from *the **History** dialog box*.

Check out to

Specify *the location where you want to check out* the new working copy by typing the local path in the text box or by using the **Browse** button. If the specified local path does not point to an existing directory, it will automatically be created.

Important: By default, the SVN system only checks out the content of the directory specified by the URL, and not the directory itself. Therefore, it is recommended to use the **Browse** button to select the *check out* location so that the client will automatically append the name of the remote directory to the path of the selected directory.

Warning: The destination directory should be empty. If files exist, they are skipped (left unchanged) by the *check out* operation and *displayed as modified* after the operation has finished. Also, the destination directory must not already be under version control.

Format

The SVN format of the working copy. You can choose between **SVN 1.8** or **SVN 1.7**.

Depth

The depth is useful if you want to *check out* only a part of the selected repository directory and bring the rest of the files and subdirectories in a future update. You can find out more about the checkout depth in the *sparse checkouts* section. You can choose between the following depths:

- **Recursive (infinity)** - Checks out all the files and folders contained in the selected folder.
- **Immediate children (immediates)** - Checks out only the child files and folders without recursing subfolders.
- **File children only (files)** - Checks out only the child files.
- **This folder only (empty)** - Checks out only the selected folder (no child file or folder is included).

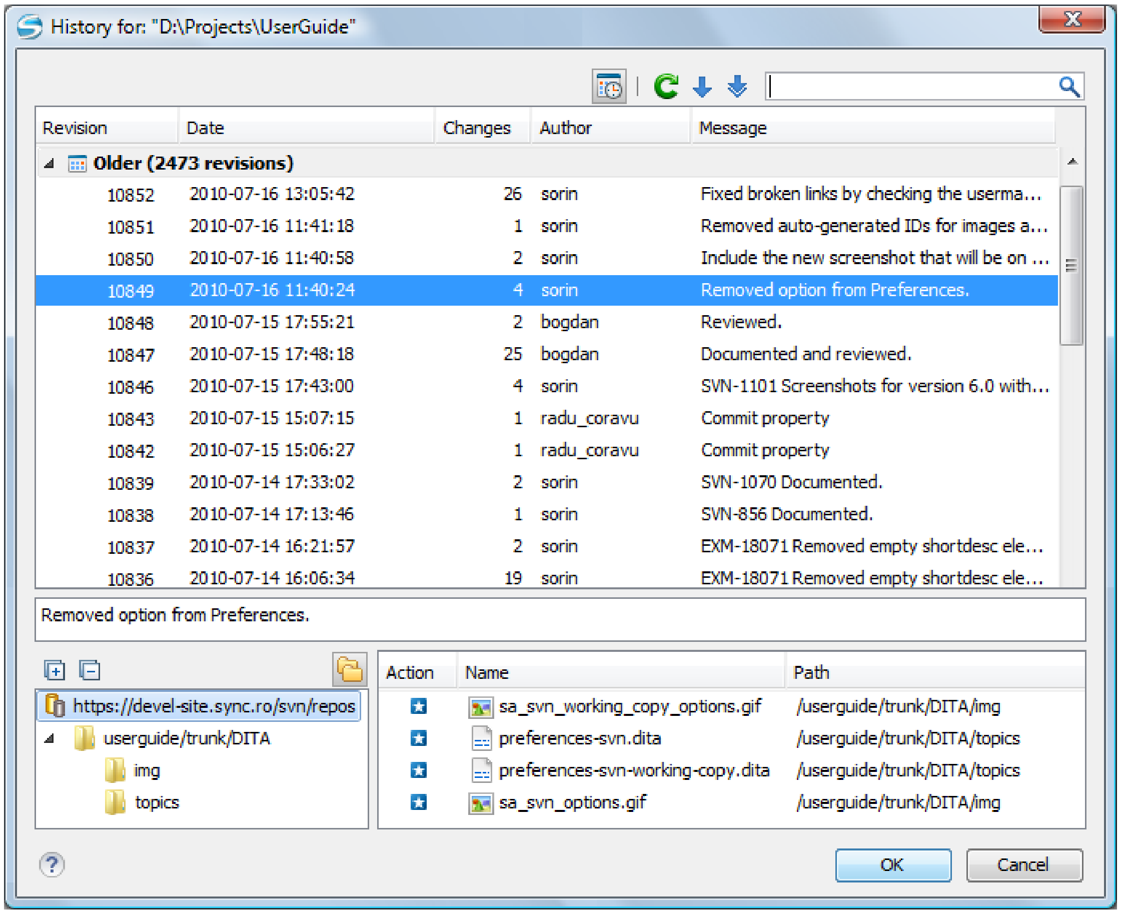
Ignore "svn:externals" definitions

When enabled, external items are ignored in the *check out* operation. This option is only available if you choose the **Recursive (infinity)** depth.

After a check out, the new working copy is 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.



Revision	Date	Changes	Author	Message
Older (2473 revisions)				
10852	2010-07-16 13:05:42	26	sorin	Fixed broken links by checking the userma...
10851	2010-07-16 11:41:18	1	sorin	Removed auto-generated IDs for images a...
10850	2010-07-16 11:40:58	2	sorin	Include the new screenshot that will be on ...
10849	2010-07-16 11:40:24	4	sorin	Removed option from Preferences.
10848	2010-07-15 17:55:21	2	bogdan	Reviewed.
10847	2010-07-15 17:48:18	25	bogdan	Documented and reviewed.
10846	2010-07-15 17:43:00	4	sorin	SVN-1101 Screenshots for version 6.0 with...
10843	2010-07-15 15:07:15	1	radu_coravu	Commit property
10842	2010-07-15 15:06:27	1	radu_coravu	Commit property
10839	2010-07-14 17:33:02	2	sorin	SVN-1070 Documented.
10838	2010-07-14 17:13:46	1	sorin	SVN-856 Documented.
10837	2010-07-14 16:21:57	2	sorin	EXM-18071 Removed empty shortdesc ele...
10836	2010-07-14 16:06:34	19	sorin	EXM-18071 Removed empty shortdesc ele...

Removed option from Preferences.

https://devel-site.sync.ro/svn/repos
userguide/trunk/DITA
img
topics

Action	Name	Path
★	sa_svn_working_copy_options.gif	/userguide/trunk/DITA/img
★	preferences-svn.dita	/userguide/trunk/DITA/topics
★	preferences-svn-working-copy.dita	/userguide/trunk/DITA/topics
★	sa_svn_options.gif	/userguide/trunk/DITA/img

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™ 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. 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.

   Note: For SVN 1.7 and newer working copies, all the internal information is kept only in the root directory. Thus, Syncro SVN Client needs to load the whole working copy.

4. Press the **OK** button.

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

Notice: You can add working copies older than SVN 1.7. However, to load any of them, Syncro SVN Client will require to upgrade the working copy to SVN 1.8 format.

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

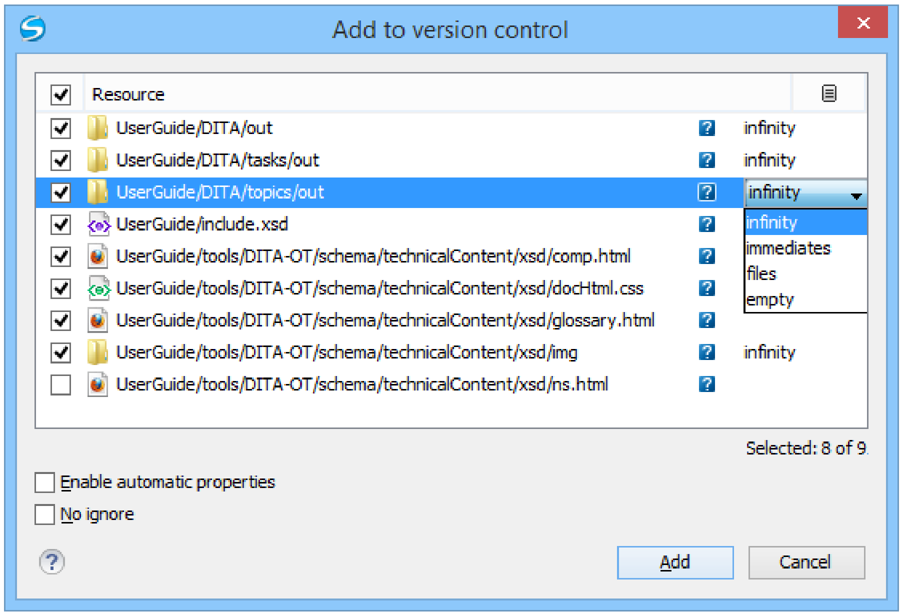
To share new files and folders (created in your working copy), add them to version control using the **Add to version control** option from the ***Working Copy** view*.

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.



	Resource		
✓	UserGuide/DITA/out	?	infinity
✓	UserGuide/DITA/tasks/out	?	infinity
✓	UserGuide/DITA/topics/out	?	infinity
✓	UserGuide/include.xsd	?	
✓	UserGuide/tools/DITA-OT/schema/technicalContent/xsd/comp.html	?	
✓	UserGuide/tools/DITA-OT/schema/technicalContent/xsd/docHtml.css	?	
✓	UserGuide/tools/DITA-OT/schema/technicalContent/xsd/glossary.html	?	
✓	UserGuide/tools/DITA-OT/schema/technicalContent/xsd/img	?	infinity
	UserGuide/tools/DITA-OT/schema/technicalContent/xsd/ns.html	?	

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 automatic properties** or **Disable automatic 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 automatic properties**, or **Disable automatic 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: *open the **Preferences** dialog* and go to **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™ 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.

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 and use the **Update** operation on the item's parent folder for the change to take effect.

Attention: For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

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: *External* items cannot be renamed using the **Rename** action because they cannot be deleted. Instead, you should edit the *svn:externals* property defining the *external* item, then use the **Update** operation on the item's parent folder for the change to take effect.

Attention: For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

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.

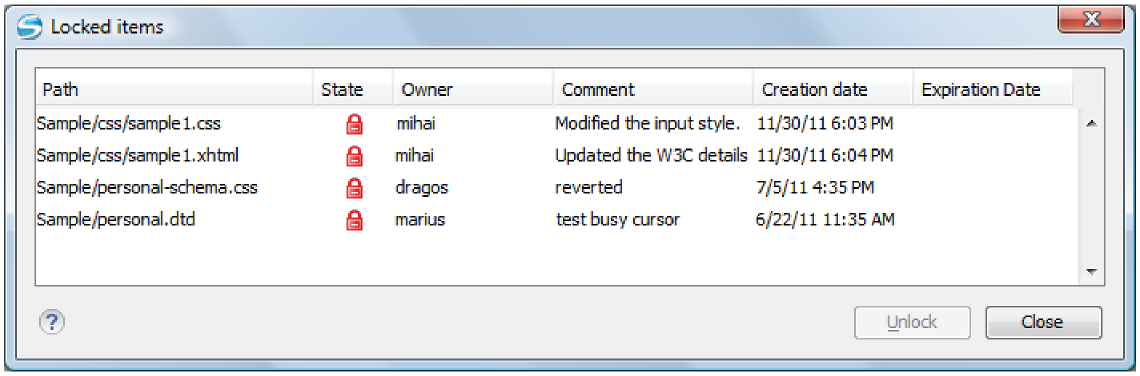
The copy-modify-merge model only works when files are contextually mergeable: this is usually the case of line-based text files (such as source code). However this is not always possible with binary formats, such as images or sounds. In these situations, the users must each have exclusive access to the 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™.

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*.

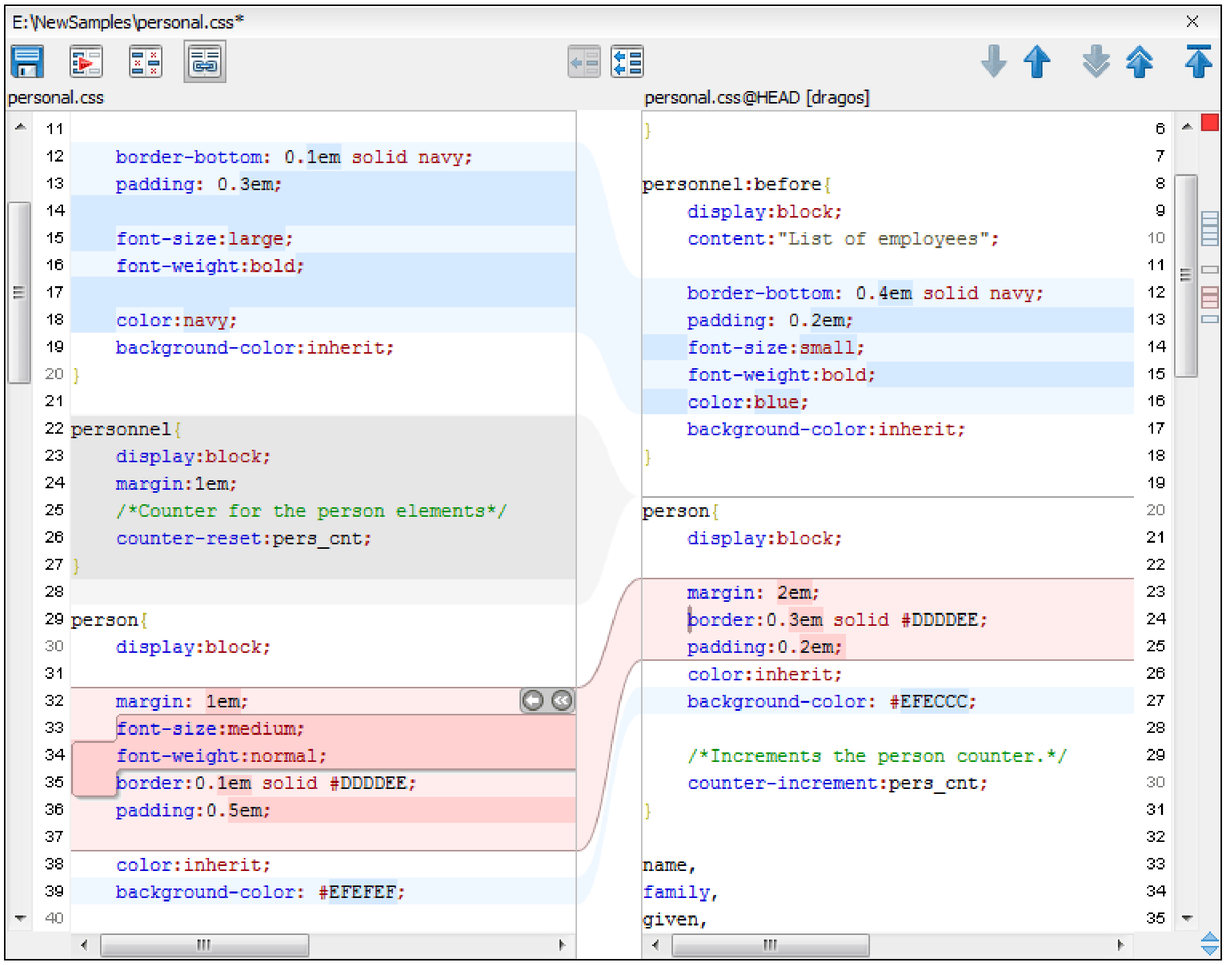
View Differences

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*.



```
    border-bottom: 0.1em solid navy;
    padding: 0.3em;

    font-size:large;
    font-weight:bold;

    color:navy;
    background-color:inherit;
}

personnel{
    display:block;
    margin:1em;
    /*Counter for the person elements*/
    counter-reset:pers_cnt;
}

person{
    display:block;

    margin: 1em;
    font-size:medium;
    font-weight:normal;
    border:0.1em solid #DDDDEE;
    padding:0.5em;

    color:inherit;
    background-color: #EFEFEF;
```

```
}

personnel:before{
    display:block;
    content:"List of employees";

    border-bottom: 0.4em solid navy;
    padding: 0.2em;
    font-size:small;
    font-weight:bold;
    color:blue;
    background-color:inherit;
}

person{
    display:block;

    margin: 2em;
    border:0.3em solid #DDDDEE;
    padding:0.2em;
    color:inherit;
    background-color: #EFECCC;

    /*Increments the person counter.*/
    counter-increment:pers_cnt;
}

name,
family,
given,
```

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™ 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 actions 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

Performs a comparison between the source file and target file.

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 so that a selected difference can be seen on both sides of the application window. This action enables/disables the previously described behavior.

Format and Indent Both Files (Ctrl+Shift+P (Command+Shift+P on OS X))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

Copy Change from Right to Left

Copies the selected difference from the target file in the right side to the source file in the left side.

Copy All Changes from Right to Left

Copies all changes from the target file in the right side to the source file in the left side.

Next Block of Changes (Ctrl+Period (Command+Period on OS X))

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.

Note: A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes (Ctrl+Comma (Command+Comma on OS X))

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.

Next Change (Ctrl+Shift+Period (Command+Shift+Period on OS X))

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 or when there are no changes.

Previous Change (Ctrl+Shift+Comma (Command+Shift+Comma on OS X))

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 or when there are no changes.

First Change (Ctrl+B (Command+B on OS X))

Jumps to the first change.

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.

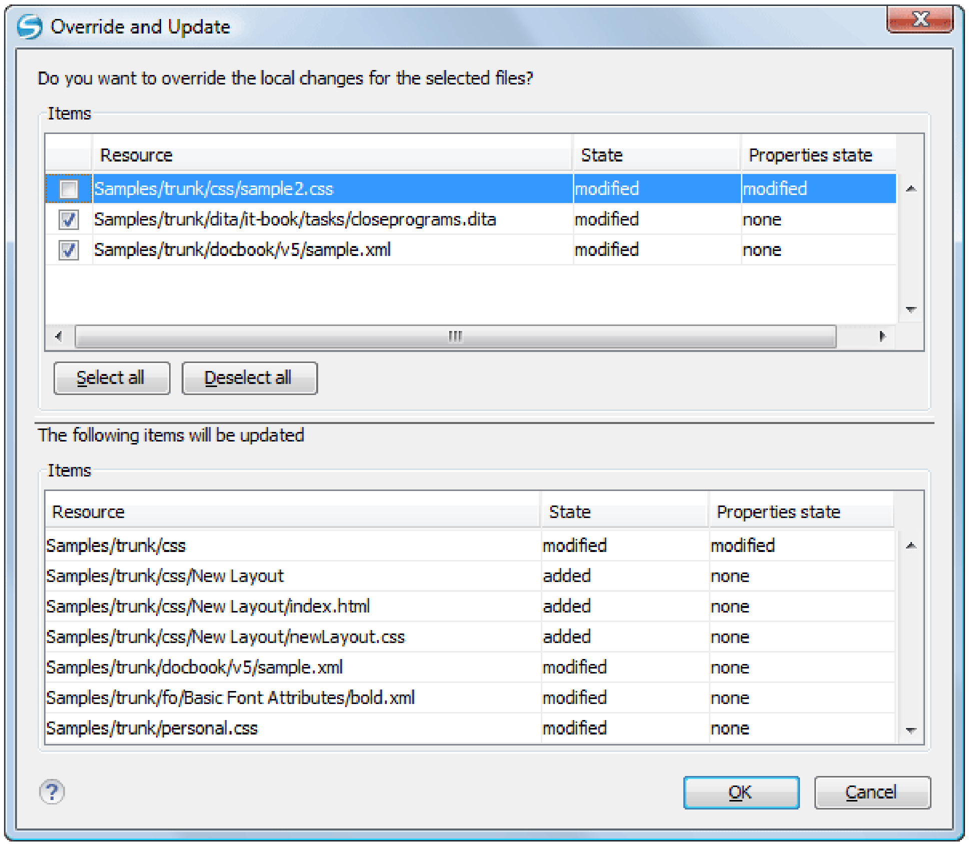
Revert Your Changes

If you want to undo changes made in your working copy, since the last update, select the items you are interested in, right click to display the contextual menu and select **Revert**. A dialog will pop up showing you the files and folders that you have changed and can be reverted. Select those you want to revert and click the **OK** button. Revert will undo only your local changes. It does not undo any changes which have already been committed. If you choose to revert a conflicting item to its pristine copy, 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.

Note: By default, a directory will be recursively reverted (including any other modified item it contains). However, if the directory has only property changes, you need to explicitly choose if the operation will include any modified items found inside it.

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, at 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.



Do you want to override the local changes for the selected files?

Items

	Resource	State	Properties state
☐	Samples/trunk/css/sample2.css	modified	modified
✓	Samples/trunk/dita/it-book/tasks/closeprograms.dita	modified	none
✓	Samples/trunk/docbook/v5/sample.xml	modified	none

The following items will be updated

Items

Resource	State	Properties state
Samples/trunk/css	modified	modified
Samples/trunk/css/New Layout	added	none
Samples/trunk/css/New Layout/index.html	added	none
Samples/trunk/css/New Layout/newLayout.css	added	none
Samples/trunk/docbook/v5/sample.xml	modified	none
Samples/trunk/fo/Basic Font Attributes/bold.xml	modified	none
Samples/trunk/personal.css	modified	none

Figure 13: Override and update dialog

In the first table of the dialog you will be able to see the resources that will be overridden. In the second table you will find the list of resources that will be updated. Only resources that have an incoming status are updated.

Tip: If you want to roll-back out of your working copy changes that have already been committed to the repository, see *Merge Revisions*.

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.

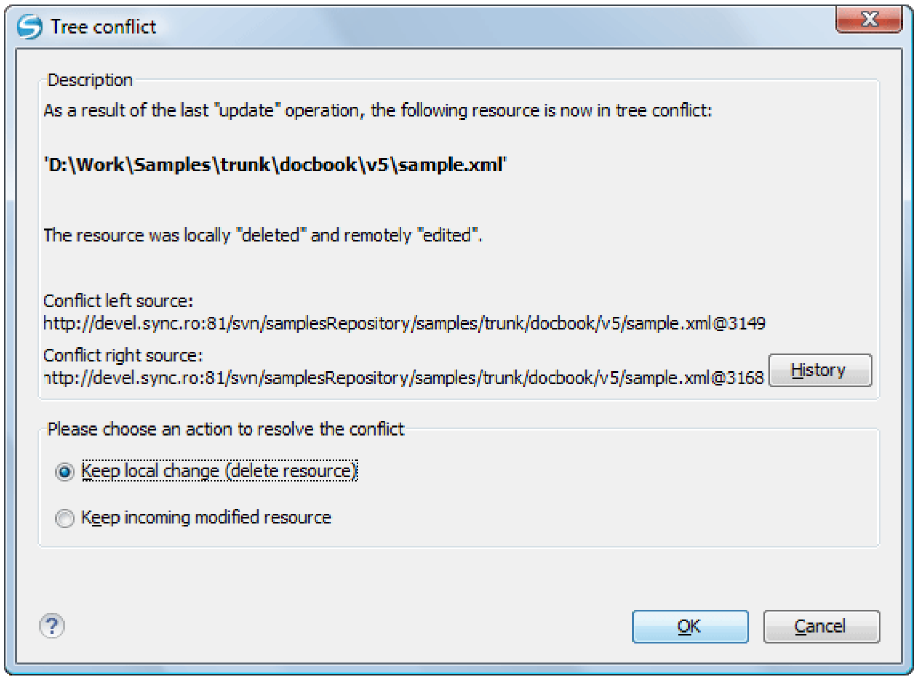
Tree Conflicts

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:


Tree conflict

Description

As a result of the last "update" operation, the following resource is now in tree conflict:

'D:\Work\Samples\trunk\docbook\v5\sample.xml'

The resource was locally "deleted" and remotely "edited".

Conflict left source:
http://devel.sync.ro:81/svn/samplesRepository/samples/trunk/docbook/v5/sample.xml@3149

Conflict right source:
http://devel.sync.ro:81/svn/samplesRepository/samples/trunk/docbook/v5/sample.xml@3168

Please choose an action to resolve the conflict

[✓ Keep local change (delete resource)]

Keep incoming modified resource

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.

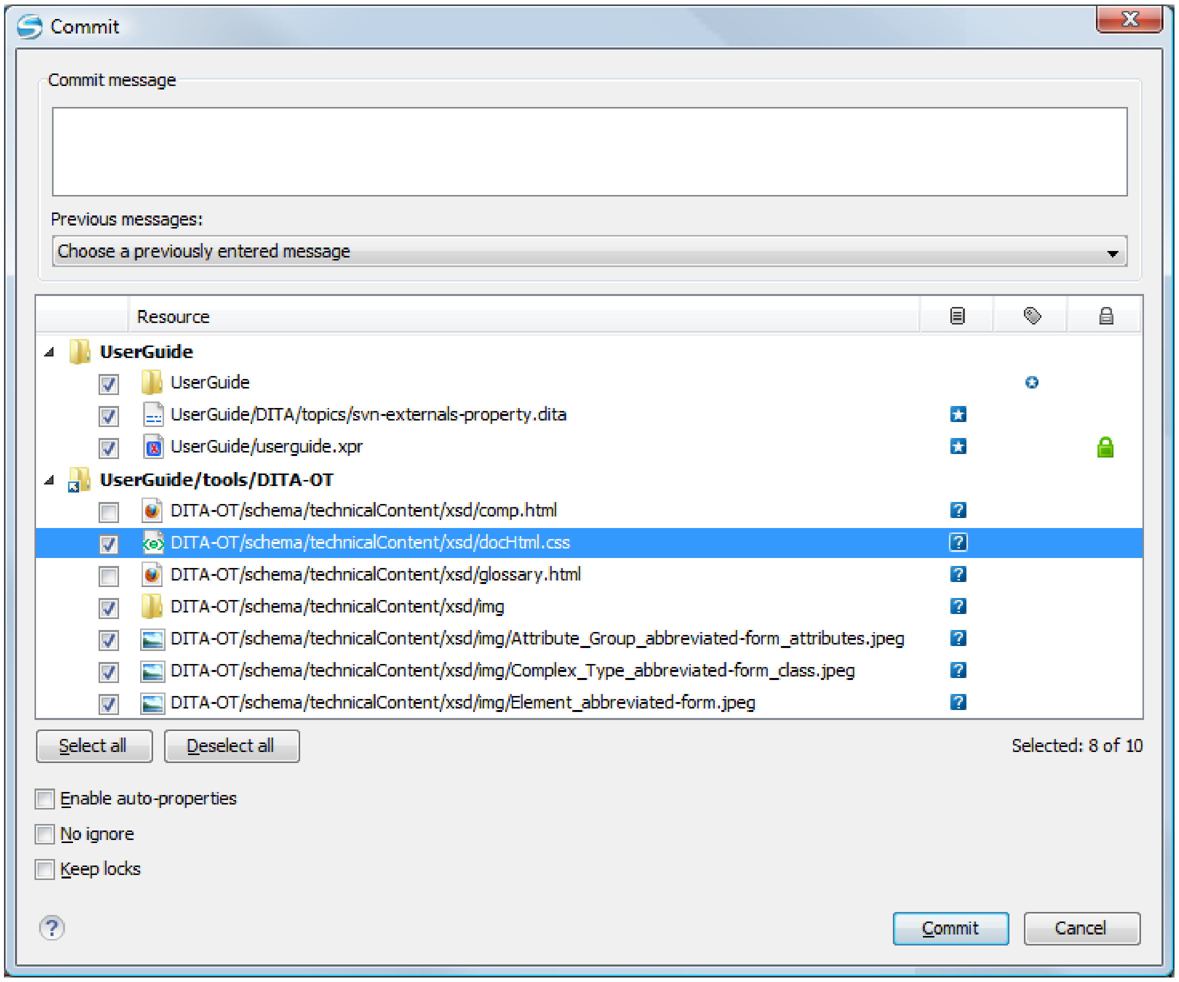
If you select multiple files and folders and then you perform an **Update** operation, all of those files and folders are updated one by one. The Subversion client makes sure that all files and folders belonging to the same repository are updated to the exact same revision, even if between those updates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



	Resource
	UserGuide
✓	UserGuide
✓	UserGuide/DITA/topics/svn-externals-property.dita
✓	UserGuide/userguide.xpr
	UserGuide/tools/DITA-OT
	DITA-OT/schema/technicalContent/xsd/comp.html
✓	DITA-OT/schema/technicalContent/xsd/docHtml.css
	DITA-OT/schema/technicalContent/xsd/glossary.html
✓	DITA-OT/schema/technicalContent/xsd/img
✓	DITA-OT/schema/technicalContent/xsd/img/Attribute_Group_abbreviated-form_attributes.jpeg
✓	DITA-OT/schema/technicalContent/xsd/img/Complex_Type_abbreviated-form_class.jpeg
✓	DITA-OT/schema/technicalContent/xsd/img/Element_abbreviated-form.jpeg

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.

Attention: For SVN 1.8 working copies: when committing items that were moved and/or renamed, make sure you select both the source and the destination, otherwise the commit operation will fail.

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 automatic properties** or **Disable automatic 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 automatic properties**, or **Disable automatic 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.

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 with the SVN Client you often need to know which files you have changed, added, removed, or renamed, or even which files got changed and committed by others. This is where the **Synchronize** action from the

Working Copy *view* comes in handy. The **Working Copy** view shows you every file that has changed 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 **Show SVN Information** action. This action is available from the **File** menu or the contextual menu of the **Working Copy**, **Repositories**, **History**, or **Directory Change Set** views, or from the **Revision Graph** dialog box. The **SVN Information** dialog box will be displayed, showing information about the selected resource. The information displayed depends on the location of the item (local or remote) and may include the following:

- Local path and repository location
- Revision number
- Last change author, revision and date
- Information about locks
- Local file status
- Local properties status
- Local directory depth
- Repository location and revision number for copied files or directories
- Path information about locally moved items
- Path information about conflict generated files
- Remote file status
- Remote properties status
- File size and other information

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

Request History for a Resource

In Apache Subversion™, both files and directories are versioned and have a history. If you want to examine the history for a selected resource and find out what happened at a certain revision you can use the **History view** that can be accessed from *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™ 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™, 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

To create a branch or tag by copying a directory, use the **Branch/Tag...** action that is available in the **Tools** menu when an item is selected in the ***Working Copy*** *view* or ***Repositories*** *view*, or from the contextual menu of the **Repositories** view.

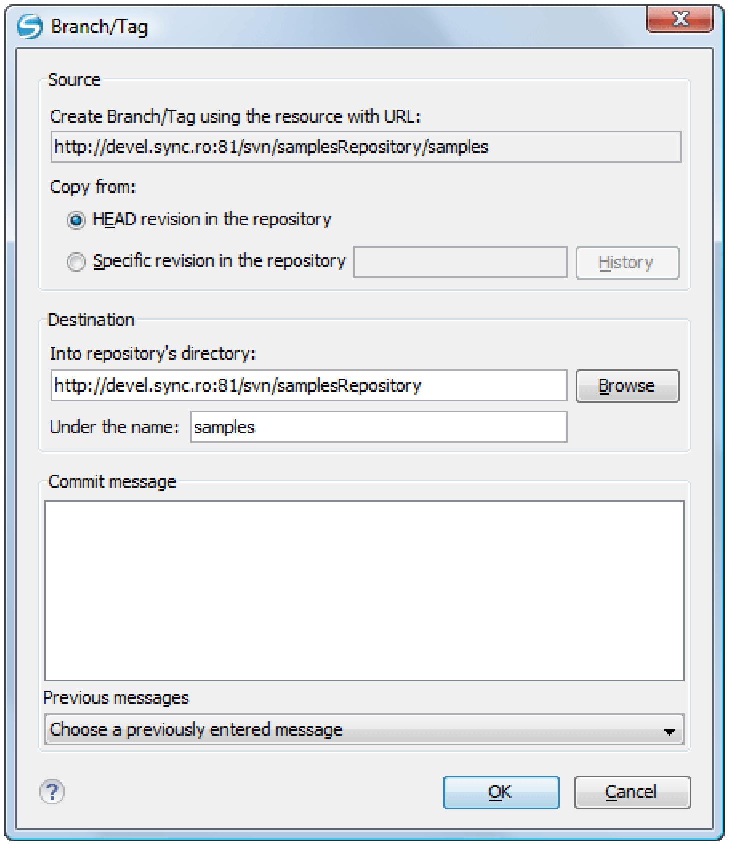

Branch/Tag
Source
Create Branch/Tag using the resource with URL:
http://devel.sync.ro:81/svn/samplesRepository/samples
Copy from:
HEAD revision in the repository
Specific revision in the repository
History
Destination
Into repository's directory:
http://devel.sync.ro:81/svn/samplesRepository
Browse
Under the name: samples
Commit message
Previous messages
Choose a previously entered message
OK
Cancel

Figure 16: The Branch/Tag Dialog Box

You can configure the following options in this dialog box:

You can specify the source revision of the copy in the **Copy from** section. You can choose between the following options:

- **HEAD revision in the repository** - The new branch or 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 into the repository, but you can specify the exact desired revision. For example, this is useful if you forgot to make a branch or tag when you released your application. If you click the **History** button you can select the revision number from *the **History** dialog box*. This type of branch will also be created very quickly.
- **Working copy** - (Available only if the item is selected from the **Working copy** view). 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, or more specifically, the locally modified files.

You can specify the location of the new branch or tag in the **Destination** section:

- **Into repository's directory** - *The URL of the parent directory* of the new branch or tag.

  Note: *Peg revisions* have no effect for this operation since it is used to send information to the repository.

- **Under the name** - You can specify another branch or tag name other than the name of the resource selected in the **Repositories** or **Working copy** view.

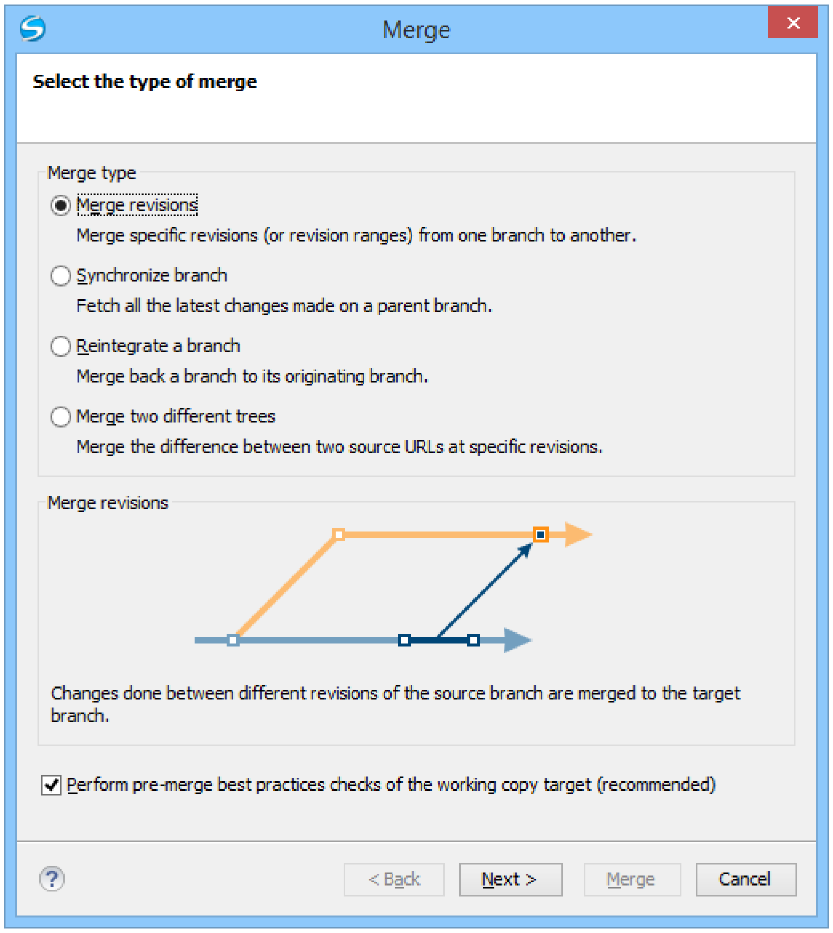
The new branch or tag will be created as a child of the specified URL of the repository directory and will have the new name.

Merging

At some stage during the development process, you will want to merge the changes made on a *branch* back into the *trunk*, or vice-versa. The *merge* is accomplished by comparing two points (branches or revisions) in the repository and applying the obtained differences to your working copy. This process is closely related to the *diff* concept.

Note: A *branch* is a line of development that exists independently of another line, yet still shares a common history if you look far enough back in time. A *branch* always begins life as *a copy of something* (such as a trunk, another branch, or tag), and moves on from there, generating its own history.

The **Merge...** action is available in the **Tools** menu. The working copy item selected when you issued the command will be the one receiving the generated changes. If there is no item selected, the *merge* operation will be performed on the entire working copy.



Select the type of merge

Merge type

(●) Merge revisions
Merge specific revisions (or revision ranges) from one branch to another.

( ) Synchronize branch
Fetch all the latest changes made on a parent branch.

( ) Reintegrate a branch
Merge back a branch to its originating branch.

( ) Merge two different trees
Merge the difference between two source URLs at specific revisions.

Merge revisions

Changes done between different revisions of the source branch are merged to the target branch.

[✓] Perform pre-merge best practices checks of the working copy target (recommended)

Figure 17: The Merge Wizard

The four types of merging are as follows:

- ***Merge revisions*** - port changes from one branch to another. Note that the *trunk* can also be considered a branch, in this context.
- ***Synchronize branch*** - fetch all the changes made on a parent branch (or the *trunk*) to a child branch.
- ***Reintegrate a branch*** - merge back a branch to its parent branch (which can also be the *trunk*).
- ***Merge two different trees*** - integrate the changes done on a branch to a different branch.

It is recommended that you enable the following pre-merge check option:

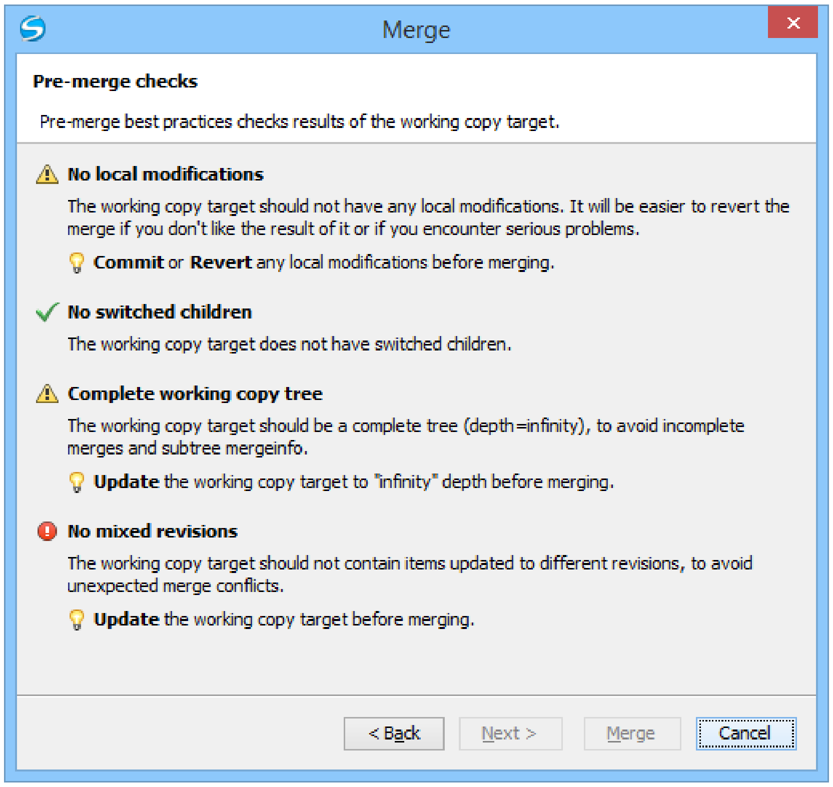
- ***Perform pre-merge best practices checks of the working copy target*** - When enabled, the SVN Client checks if the working copy target item is ready for the merge operation and displays the **pre-merge checks** wizard page.

Remember: 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.

Important: The above recommendation becomes mandatory when *reintegrating a branch*.

Pre-Merge Checks

Before performing a merge, it is recommended to make sure that the working copy target item is ready for the merge operation. The SVN Client includes a best practices step that checks various conditions of the working copy target item to ensure that the merge operation will succeed. By enabling the **Perform pre-merge best practices checks of the working copy target** option in the first page of the **Merge** wizard, the **Pre-merge checks** wizard page is displayed to give you a summary of the verified conditions.



Pre-merge checks

Pre-merge best practices checks results of the working copy target.

No local modifications

The working copy target should not have any local modifications. It will be easier to revert the merge if you don't like the result of it or if you encounter serious problems.

Commit or **Revert** any local modifications before merging.

No switched children

The working copy target does not have switched children.

Complete working copy tree

The working copy target should be a complete tree (depth=infinity), to avoid incomplete merges and subtree mergeinfo.

Update the working copy target to "infinity" depth before merging.

No mixed revisions

The working copy target should not contain items updated to different revisions, to avoid unexpected merge conflicts.

Update the working copy target before merging.

Figure 18: The Pre-Merge Checks Wizard Page

The following conditions are checked in this operation:

No local modifications

The working copy item (or any of its children) receiving the merge should not contain uncommitted changes, to make it easier to revert merge-generated changes if you encounter unexpected results.

Tip: If this condition fails, you should *commit* or *revert* the local modifications before merging.

No switched children

None of the children of the working copy item receiving the merge should be switched, to avoid incomplete merges and *subtree mergeinfo*.

Tip: If this condition fails, you should switch back all the children before merging.

Complete working copy tree

The working copy item receiving the merge should be a complete directory tree structure with an infinite depth, to avoid incomplete merges and *subtree mergeinfo*.

Tip: If this condition fails, you should change the *sticky* depth of the working copy item receiving the merge to *infinity* value.

No mixed revisions

The working copy item receiving the merge should not contain items that were updated to different revisions, to avoid unexpected merge conflicts.

Tip: If this condition fails, you should *update* the working copy before merging.

Each condition is marked with an icon that represents the state of the condition. The possible states are as follows:

- ✓ **Successful** - The condition is fulfilled successfully.
- ⚠ **Warning** - The condition is not fulfilled, but it is not mandatory.
- ❗ **Error** - The condition is not fulfilled and is mandatory, and therefore the operation cannot proceed until you solve the error.

Tip: For each condition state, a message is displayed that gives you additional information about the results and, for warning or errors, a hint that explains how you can solve them.

Important: After solving any of the warnings or errors, it is recommended that you perform the *pre-merge checks* again to make sure your new changes are valid.

Merge Revisions

This is the case when you have made one or more changes to a branch and you want to duplicate them in a different branch. For example, we know that a problem has been fixed by committing revisions `17`, `20`, and `25` on branch `B1`. These changes are also needed in branch `B2`. Thus, in order to merge them, we need a working copy of the `B2` branch.

To merge revisions from a different branch, follow these steps:

1. Go to menu **Tools** > **Merge**.
   The **Merge** wizard is opened.
2. Select the **Merge revisions** option.
3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
   a) Press the **Next** button.
   If the **Perform pre-merge best practices checks of the working copy target** option was enabled, *the* ***Pre-Merge Checks*** *wizard page* is displayed.

   Note: If errors are found you need to solve them before proceeding.

4. Press the **Next** button.
   The **Merge revisions** wizard page is displayed.
5. In the **Merge from (URL)** text box, enter *the URL of the branch or tag* that contain the changes that you want to duplicate in your working copy. In our example, it is the URL of the `B1` branch.

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

   Note: If the URL belongs to a different repository than the working copy, the **Ignore ancestry / Disable merge tracking** option, in the ***Merge Options*** *wizard page*, will be enabled automatically (and you cannot change this). This is because the *Subversion client cannot track changes between different repositories*.

   Tip: You can also specify a *peg revision* at the end of the URL (for example, `URL@rev1234`). The peg revision does not affect the merge range you select. By default, the `HEAD` revision is assumed.

6. In the **Revisions to merge** section, choose between the **all revisions** and **specific revision(s)** options.

- **all revisions** - The operation will include *all eligible revisions* that were not yet merged.
- **specific revision(s)** - You can specify one or more individual revisions and/or revision ranges. Also, you can mix *forward* ranges (for example, `1-5`), *backward* ranges (for example, `20-15`), and subtract specific revisions from a range (for example, `1-5, -3`).

Note: If using the Subversion command-line client, a revision range of the form `1-5` means all changes starting from revision 2 up to revision 5 (the changes necessary to reach revision 5, committed after revision 1). Unlike the Subversion command-line client, in Syncro SVN Client the revision ranges are inclusive, meaning that it will process all revisions, starting with revision 1, up to and including revision 5.

Attention: The `HEAD` revision is the only non-numerical revision allowed, and it can only be used when specifying revision ranges as one of the ends of the range (for example, `10-HEAD`). Be careful when using it, as it might not refer to the desired revision, if it has recently been committed by another user.

Tip: If you want to perform a *reverse merge* and roll-back your working copy changes that have already been committed to the repository, use the *negative revisions* notation (for example, `-7`) or *backward revision ranges* (for example, `20-10`).

a) If you press the **History** button, *the **History** dialog box* is displayed, which allows you to select one or more revisions to be merged.

7. Optionally, if you want to *configure the options* for your merge, press the **Next** button.
The ***Merge Options** wizard page* is displayed that allows you to configure options for the operation.

Warning: If the **Ignore ancestry / Disable merge tracking** option is enabled and you selected **all revisions** in the **Revisions to merge** section, revisions that were previously merged will also be included, which may result in conflicts.

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

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

Note: The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

Synchronize a Branch

While working on your own branch, other people on your team might continue to make important changes in the parent branch (which can be the *trunk* itself or any other branch). It is recommended to periodically duplicate those changes in your branch to make sure your changes are compatible with them. This is done by performing a *synchronize merge*, which will bring your branch up-to-date with any changes made to its ancestral parent branch since the time your branch was created or last synchronized. Subversion is aware of the history of your branch and can detect when it split away from the parent branch.

Frequently keeping your branch in sync with the parent branch helps you to prevent unexpected conflicts when the time comes for you to duplicate your changes back into the parent branch. The synchronization uses *merge tracking* to skip all those revisions that have already been merged, thus a sync merge can be repeated periodically to fetch all the latest changes of the parent branch to keep up-to-date with it.

Important: It is recommended to synchronize the whole working copy that was created from the child branch (the root of the working copy), rather than just a part of it.

After running the *synchronize merge*, your working copy from the child branch now contains new local modifications, and these edits are duplications of all of the changes that have happened on the *trunk* since you first created your branch. At this point, your private branch is now synchronized with the trunk.

To synchronize your branch with its parent branch, follow these steps:

1. Go to **Tools** > **Merge**.
   The **Merge** wizard is opened.
2. Select the **Synchronize branch** option.
3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
   a) Press the **Next** button.
      If the **Perform pre-merge best practices checks of the working copy target** option was enabled, *the **Pre-Merge Checks** wizard page* is displayed.

      Note: If errors are found you need to solve them before proceeding.
4. Press the **Next** button.
   The **Synchronize branch** wizard page is displayed.
5. In the **Parent branch (URL)** text box, enter *the URL of the branch from which you created your branch*. This means that the URL must belong to the same repository as your working copy that was created from the child branch.

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

   Tip: You can also specify a *peg revision* at the end of the URL (for example, `URL@rev1234`). The peg revision specifies both the peg revision of the URL and the latest revision that will be considered for merging. By default, the `HEAD` revision is assumed.
6. Optionally, if you want to *configure the options* for your merge, press the **Next** button.
   The ***Merge Options** wizard page* is displayed that allows you to configure options for the operation.

   Note: The **Ignore ancestry / Disable merge tracking** option is not available for this merge type, since a synchronization merge should always be recorded in the destination branch.
7. Press the **Merge** button.
   The merge operation is performed.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

Note: The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

Reintegrate a Branch

There are some conditions that apply to reintegrate a branch:

- The server must support merge tracking.
- The source branch (to be reintegrated) must be coherently synchronized with its parent branch. This means that all revisions between the branching point and the last revision merged from the parent branch to the child branch must be merged to the latter one (there must be no missing revisions in-between).
- The working copy **must not**:
  - have any local modifications.

- contain a mixture of revisions (all items must point to the same revision).
- have any sparse directories (all directories must be of depth *infinity*).
- contain any switched items.

- The revision of the working copy must be greater than or equal to the last revision of the parent branch with which the child branch was synchronized.

Tip: You can use *the pre-merge checks option* to make sure these conditions are fulfilled.

This method is useful when you have a feature branch on which the development has concluded and it should be merged back into its parent branch. Since you have kept the feature branch synchronized with its parent, the latest versions of them will be absolutely identical except for your feature branch changes. These changes can be reintegrated into the parent branch by using a working copy of it and the **Reintegrate a branch** option.

This method uses the *merge-tracking* features of Apache Subversion™ to automatically calculate the correct revision ranges and to perform additional checks that will ensure that the branch to be reintegrated has been fully updated with its parent changes. This ensures that you do not accidentally undo work that others have committed to the parent branch since the last time you synchronized the child branch with it. After the merge, all branch development will be completely merged back into the parent branch, and the child branch will be redundant and can be deleted from the repository.

Tip: Before reintegrating the child branch it is recommended to synchronize it with its parent branch one more time, to help avoid any possible conflicts.

To reintegrate a child branch into its parent branch, follow these steps:

1. Go to menu **Tools** > **Merge**.
   The **Merge** wizard is opened.
2. Select the **Reintegrate a branch** option.

   Note: This option is disabled if the selected working copy item (or if it is a directory, any of the items inside of it) has any type of modification. This is because it is mandatory for the target item to have no modifications.

3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
   a) Press the **Next** button.
      If the **Perform pre-merge best practices checks of the working copy target** option was enabled, *the **Pre-Merge Checks** wizard page* is displayed.

      Note: If errors are found you need to solve them before proceeding.

4. Press the **Next** button.
   The **Reintegrate a branch** wizard page is displayed.
5. In the **Child branch (URL)** text box, enter *the URL of the child branch to be reintegrated*. This means that the URL must belong to the same repository as your working copy that was created from the parent branch.

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

   Tip: You can also specify a *peg revision* at the end of the URL (for example, `URL@rev1234`). The peg revision specifies both the peg revision of the URL and the latest revision that will be considered for merging. By default, the `HEAD` revision is assumed.

6. The ***Merge Options** wizard page* is displayed that allows you to configure options for the operation.

   Note: Because a *reintegrate merge* is so specialized, most of the merge options are not available, except for those in the **File Comparison** category.

7. Press the **Merge** button.

The merge operation is performed.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

Note: The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

Merge Two Different Trees

This merge type is useful when you need to duplicate changes from one child branch (for example, `CB1`) to another child branch (`CB2`) from the same parent branch. The SVN client will calculate the changes necessary to get from the `HEAD` revision of the parent branch (or the *trunk*) to the `HEAD` revision of one of its child branches (`CB1`), and apply those changes to your working copy of the other branch (`CB2`). The result is that the latter child branch (`CB2`) will also include the changes made on the original child branch (`CB1`), although that branch was not reintegrated into the parent branch.

This merge type could also be used to reintegrate a child branch back into its parent when the repository does not support *merge tracking*.

Note: If the server does not support *merge-tracking*, then this is the only way to merge a branch back to its parent.

1. Go to menu **Tools** > **Merge**.
   The **Merge** wizard is opened.
2. Select the option **Merge two different trees**.
3. It is recommended that you enable the **Perform pre-merge best practices checks of the working copy target** option to make sure that the working copy target item is ready for the merge operation.
   a) Press the **Next** button.
      If the **Perform pre-merge best practices checks of the working copy target** option was enabled, *the **Pre-Merge Checks** wizard page* is displayed.

      Note: If errors are found you need to solve them before proceeding.

4. Press the **Next** button.
   The **Merge two different trees** wizard page is displayed.
5. In the **From (starting URL and revision)** section enter *the URL of the first branch*.

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

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

   Note: If the URL belongs to a different repository than the working copy, the **Ignore ancestry / Disable merge tracking** option, in the ***Merge Options** wizard page*, will be enabled automatically (and you cannot change this). This is because the *Subversion client cannot track changes between different repositories*.

   Tip: You can also specify a *peg revision* at the end of the URL (for example, `URL@rev1234`). By default, the `HEAD` revision is assumed.

6. Enter the last revision number at which the two trees were synchronized by choosing between **HEAD revision** and **other revision**.
   - **HEAD revision** - Use this option if you are sure that no one else has committed changes since the last synchronization.
   - **other revision** - Use this option to input a specific revision number and avoid losing recent commits. You can use the **History** button to see a list of all revisions.
7. In the **To (ending URL and revision)** section enter *the URL of the second branch*.

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

   Tip: If you are using this method to merge a feature branch back to its parent branch, enter the URL of the feature branch. This way, only the changes unique to this branch will be merged, since the branch should have been periodically synchronized with its parent.

   Attention: The URL must point to the same repository as the one in the **From (starting URL and revision)** field. Otherwise, the operation will not be allowed, since Subversion cannot compute changes between items from different repositories.

   Tip: You can also specify a *peg revision* at the end of the URL (for example, `URL@rev1234`). By default, the `HEAD` revision is assumed.
8. Select a revision to compute all changes committed up to that point by choosing between **HEAD revision** and **other revision**.
   - **HEAD revision** - This is the default selected revision.
   - **other revision** - Use this option if you want to enter a previous revision. You can use the **History** button to see a list of all revisions.
9. Optionally, if you want to *configure the options* for your merge, press the **Next** button.
   The ***Merge Options** wizard page* is displayed that allows you to configure options for the operation.

   Warning: If the **Ignore ancestry / Disable merge tracking** option is enabled and you selected **all revisions** in the **Revisions to merge** section, revisions that were previously merged will also be included, which may result in conflicts.
10. Press the **Merge** button.
    The merge operation is performed.

If the merge is completed successfully, all the changes corresponding to the selected revisions should be merged in your working copy.

It is recommended to look at the results of the merge, in the working copy, to review the changes and see if it meets your expectations. Since merging can sometimes be complicated, *you may need to resolve conflicts* after making major changes.

Note: The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

Merge Options

Here is the list of options that can be used when merging:

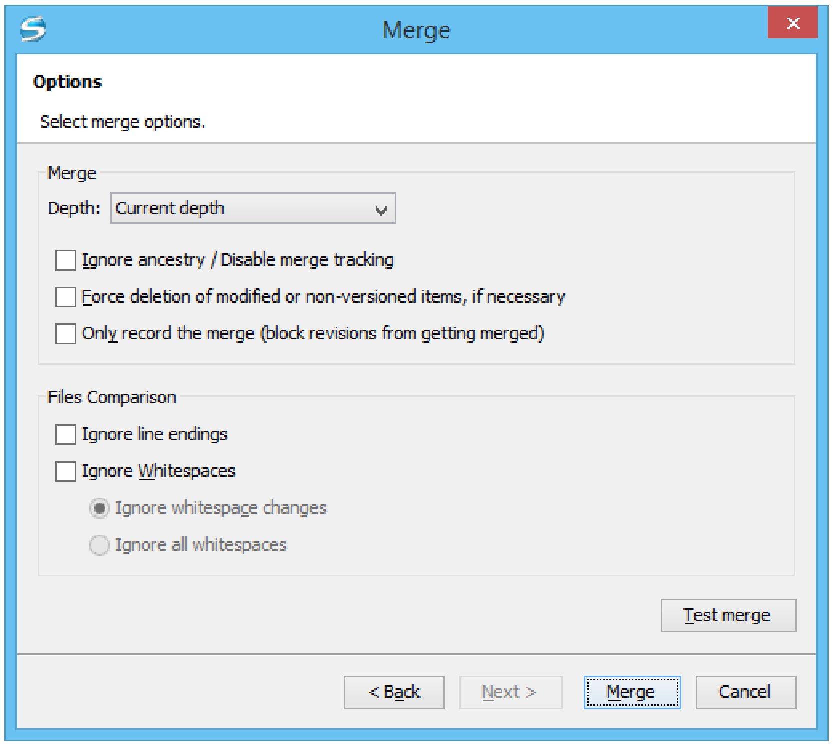

Merge
Options
Select merge options.
Merge
Depth: Current depth
Ignore ancestry / Disable merge tracking
Force deletion of modified or non-versioned items, if necessary
Only record the merge (block revisions from getting merged)
Files Comparison
Ignore line endings
Ignore Whitespaces
Ignore whitespace changes
Ignore all whitespaces
Test merge
< Back
Next >
Merge
Cancel

Figure 19: The Merge Wizard - Advanced Options

- **Depth** (This option is applicable only for directories) - sets the depth of the merge operation. You can specify how far down into your working copy the merge should go by selecting one of the following values:
  - **Current depth** - Obeys the depths registered for the directories in the working copy that are to be switched.
  - **Recursive (infinity)** - Merges all the files and folders contained in the selected folder. This is the recommended depth for most users, to avoid incomplete merges and *subtree mergeinfo*.
  - **Immediate children (immediates)** - Merges only the child files and folders without recursing subfolders.
  - **File children only (files)** - Merges only the child files.
  - **This folder only (empty)** - Merges only the selected folder (no child files or folders are included).

  Note: The *depth* term is described in the *Sparse checkouts* section. The default depth is the current depth of the working copy item receiving the merge.

- **Ignore ancestry / Disable merge tracking** - Changes the way two items are merged if they do not share a common ancestry. Most merges involve comparing items that are ancestrally related to one another. However, occasionally you may want to merge unrelated items. If this option is disabled, the first item will be replaced with the second item. In these situations, you would want the merge to do a path-based comparison only, ignoring any relations between the items. For example, if two different files have the same name and are in the same relative location, disabling the option replaces one of the files with the other one, and enabling it merges their contents.

  Note: If the URL of the merge source belongs to a different repository than the URL of the target working copy item (the one receiving the changes), this option is selected automatically (and you cannot change this). This is because the *Subversion client cannot track changes between different repositories*.

- **Force deletion of modified or non-versioned items, if necessary** - If disabled, when the merge operation involves deleting locally modified or non-versioned items, it will fail. This is done in order to prevent data loss. This option is only available if there are uncommitted changes in the working copy.
- **Only record the merge (block revisions from getting merged)** - Available when the **Ignore ancestry / Disable merge tracking** option is disabled. It enables a special mode of the merge operation that just records it in the local merge tracking information, without actually performing it (does not modify any file contents or the structure of your working copy). You might want to enable this option for two possible reasons:

- You made (or will make) the merge manually, and therefore you need to mark the revisions as being merged to make the merge tracking system aware of them. This will exclude them from future merges.
- You want to prevent one or more particular changes from being fetched in subsequent merges.

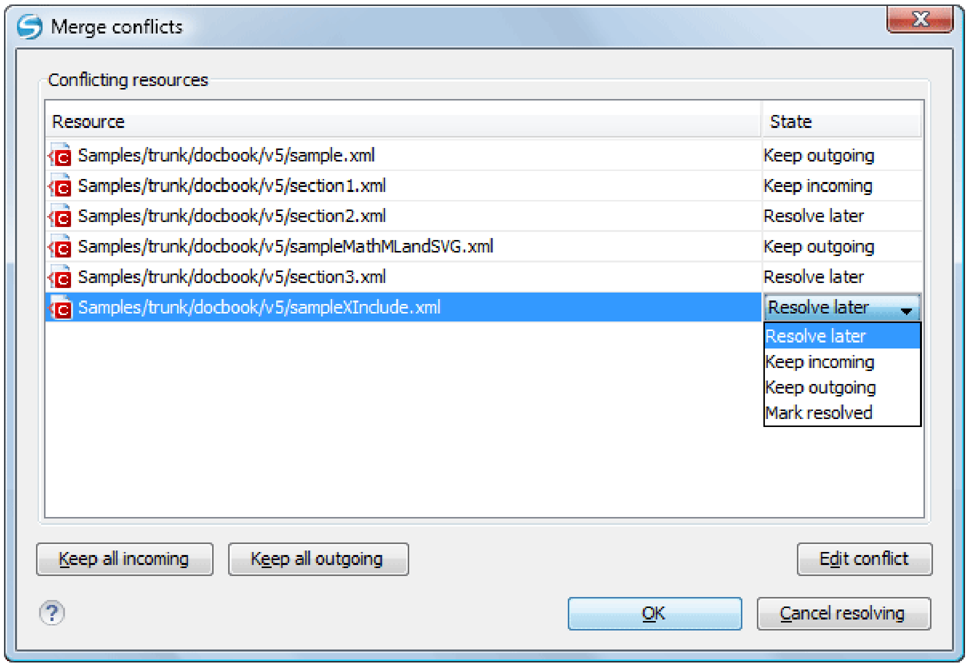
- **Ignore line endings** - Allows you to specify how the line ending changes should be handled. By default, all such changes are treated as real content changes, but you can ignore them if you select this option.
- **Ignore whitespaces** - Allows you to specify how the whitespace changes should be handled. By default, all such changes are treated as real content changes, but you can ignore them if you select this option.
  - **Ignore whitespace changes** - Ignores changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces).

    Note: Whitespaces that were added where there were none before, or that were removed, are still considered to be changes.

  - **Ignore all whitespaces** - Ignores all types of whitespace changes.
- **Test merge** - Performs a dry run of the merge operation, allowing you to *preview* it without actually performing the merge. In the **Console** view you will see a list of the working copy items that will be affected and how they will be affected. This is helpful in detecting whether or not a merge will be successful, and where conflicts may occur.

Resolving Merge Conflicts

After the merge operation is finished, it is possible to have some items in conflict. This means that some incoming modifications for an item could not be merged with the current working copy version. If there are such conflicts, the **Merge conflicts** dialog box will appear, presenting the items that are in conflict. This dialog box offers you choices for resolving the conflicts.



Resource	State
Samples/trunk/docbook/v5/sample.xml	Keep outgoing
Samples/trunk/docbook/v5/section1.xml	Keep incoming
Samples/trunk/docbook/v5/section2.xml	Resolve later
Samples/trunk/docbook/v5/sampleMathMLandSVG.xml	Keep outgoing
Samples/trunk/docbook/v5/section3.xml	Resolve later
Samples/trunk/docbook/v5/sampleXInclude.xml	Resolve later

Figure 20: Merge Conflicts Dialog

The options to resolve a conflict are as follows:

- **Resolve later** - Used for leaving the conflict as it is, to manually resolve it later.
- **Keep incoming** - This option keeps all the incoming modifications and discards all current ones from your working copy.
- **Keep outgoing** - This option keeps all current modifications from your working copy and discards all incoming ones.

- **Mark resolved** - You should choose this option after you have manually solved the conflict, to instruct the Subversion that it was resolved. To do this, use the **Edit conflict** button, which displays a dialog box that presents the contents of the conflicting items (the content of the working copy version versus the incoming version).

Additional Notes About the Merge Operation

Sub-tree Merges

It is recommended to perform a merge on the whole working copy (select its root directory when triggering the operation) to avoid *sub-tree mergeinfo*. *Sub-tree mergeinfo* is the *mergeinfo* recorded to describe a *sub-tree merge*. That is, a merge done directly to a child of a branch root that might be needed in certain situations. There is nothing special about *sub-tree merges* or *sub-tree mergeinfo* except that the complete record of merges to a branch may not be contained solely in the *mergeinfo* on the branch root and you may have to look to any *sub-tree mergeinfo* to get a full accounting. Fortunately, Subversion does this for you and rarely will you need to look for it.

Merging from Foreign Repositories

Subversion supports merging from foreign repositories. While all merge source URLs must point to the same repository, the merge target (from the working copy) may come from a different repository than the sources. However, copies made in the merge source will be transformed into plain additions in the merge target. Also, *merge-tracking* is not supported for merges from foreign repositories.

Note: When performing merges from repositories other than the one corresponding to the target item (from the working copy), the **Ignore ancestry / Disable merge tracking** option, in the ***Merge Options** wizard page*, will be enabled automatically (and you cannot change this).

General Merge Recommendations

As a recommendation, you should only merge into clean working copies that **do not** contain any of the following:

- Modifications.
- Sparse directories (all directories must be of depth *infinity*).
- Switched items.

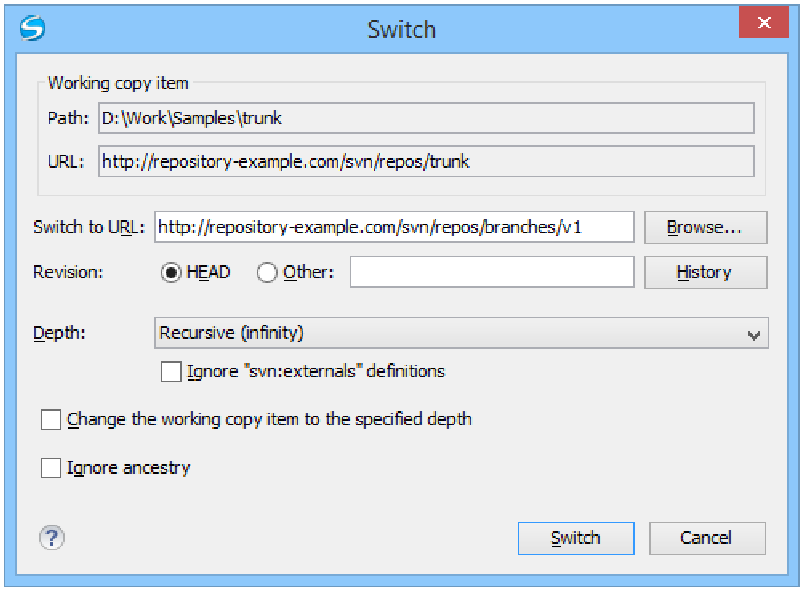
Important: This recommendation becomes mandatory when performing a *reintegrate merge* operation. Also, trying to merge to mixed-revision working copies will fail in all types of merge operations.

Remember: The merge result is only in your local working copy and needs to be committed to the repository for it to be available to others.

Switch the Repository Location

The **Switch** action is useful when the repository location of a working copy, or an already committed item in 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*.

Note: *External* items cannot be switched using this action. Instead, change the value of the `svn:externals` property set on the parent directory of the external item and update the parent directory.


Switch

Working copy item

Path: D:\Work\Samples\trunk

URL: http://repository-example.com/svn/repos/trunk

Switch to URL: http://repository-example.com/svn/repos/branches/v1

Revision: [✓ HEAD] Other:

Depth: Recursive (infinity)

☐ Ignore "svn:externals" definitions

☐ Change the working copy item to the specified depth

☐ Ignore ancestry

Figure 21: The Switch Dialog Box

The following options can be configured in the **Switch** dialog box:

Switch to URL

The new location in the same repository to which you are switching.

Tip: You can switch to items that were deleted, moved, or replaced, by specifying the original URL (before the item was removed) and use a *peg revision* at the end (for example, `URL@rev1234`).

Note: For items that are already *switched* that you want to switch back, the proposed URL is the original location of the item.

Revision

The specific version of the location to which you are switching.

Depth - (This option is applicable only for directories)

Current depth - Obeys the depths registered for the directories in the working copy that are to be switched.

Recursive (infinity) - Switches the location of the selected folder and all of its files and folders.

Immediate children (immediates) - Switches the location of the selected folder and its child files and folders without recursing subfolders.

File children only (files) - Switches the location of the selected folder and its child files.

This folder only (empty) - Switches the location of the selected folder (no child files or folders are included).

Ignore "svn:externals" definitions

When enabled, external items are ignored in the switch operation. This option is only available if you choose the **Current depth** or **Recursive (infinity)** depth.

Change the working copy item to the specified depth

Changes the *sticky* depth on the directory in the working copy.

Ignore ancestry

When disabled, the SVN system does not allow you to switch to a location that does not share a common ancestry with the current location. If enabled, the SVN does not check for a common ancestry.

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, if the repository itself is moved to a different server. In such cases, you do not have to check out 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*.

Note: *Peg revisions* have no effect for this operation.

This **Relocate** action is available in the **Tools** menu when selecting the root item of the working copy.

Note: *External* items that are defined using absolute URLs and that point to the same repository as the working copy are not affected by the **Relocate** action (the URL is not updated). To relocate these items, change the value of the `svn:externals` property for each external item, which is set on their parent directories. For example, if an external item is defined as `externalDir http://host/path/to/repo/to/dir` and the repository was moved to another server (`host2`) and its protocol changed to `https`, then the value of the property needs to be updated to `externalDir https://host2/path/to/repo/to/dir`.

Tip: If you edit external items using the method described in the previous note, on the next update the system will try to fetch the external items again. To avoid this, you can invoke the **Relocate** action on each of these external items.

Patches

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

What is a Patch

Suppose you are working with a set of XML files that you want to tag the project and distribute releases to other team members. While working on the project and correcting problems, you may need to distribute the corrections to other team members. In this case, you can distribute a patch, which is a collection of differences, that applied over the last distribution, would correct the problems. By default, the Syncro SVN Client generates patches in *the Unified Diff format*, but it can also use *the Git format*.

Creating a patch in Apache Subversion™ implies the access to either two revisions of a versioned item, or two different versioned items from the repository:

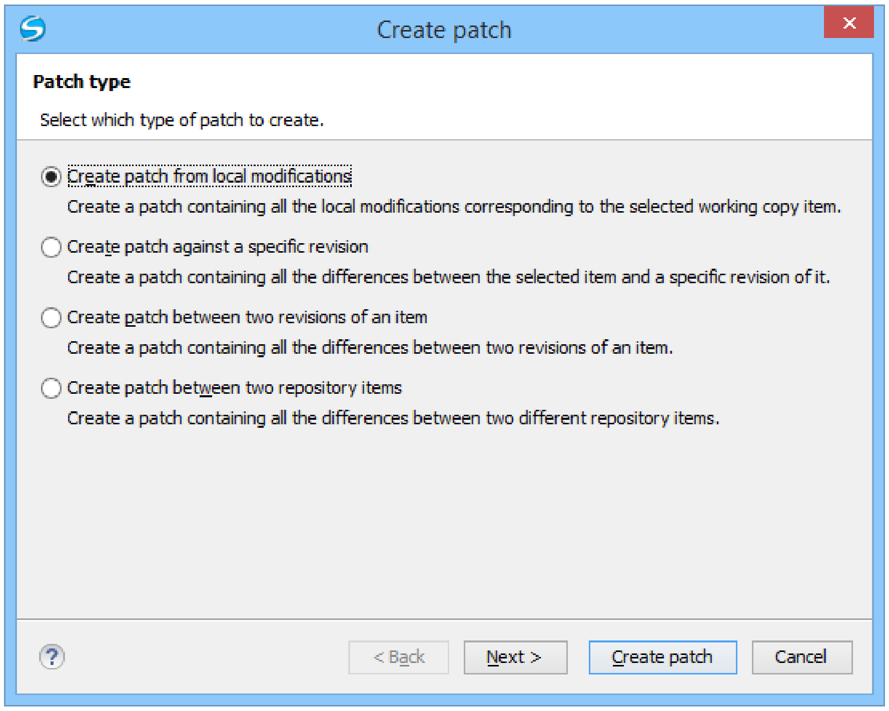
- *Two revisions of a version item* - the item can be local or remote and you can select two versions of it. This also applies when you need to generate a patch that only contains the changes in the working copy that were not yet committed.
- *Two different versioned items from the repository* - the items are remote and you need to specify a revision for both.

Warning: The resulting patch file may contain content that was written using a mix of encodings, based upon the encodings of the files that were compared. If you open the generated patch file in a text editor, it may result in unrecognizable content.

Generating a Patch - Local Items

Based on a versioned item (already committed or scheduled for addition) in the working copy, you can generate patches that contain the local changes, the differences between a specific revision of that item and the item itself, or differences between the pristine item and another item from the repository. There are four options for generating a patch based upon local items.

To open the **Create patch** wizard, use the **Create patch...** action from the **Tools** menu or from the contextual menu in the **Modified**, **Incoming**, **Outgoing**, or **Conflicts** modes.


Create patch

Patch type

Select which type of patch to create.

(●) Create patch from local modifications
Create a patch containing all the local modifications corresponding to the selected working copy item.

( ) Create patch against a specific revision
Create a patch containing all the differences between the selected item and a specific revision of it.

( ) Create patch between two revisions of an item
Create a patch containing all the differences between two revisions of an item.

( ) Create patch between two repository items
Create a patch containing all the differences between two different repository items.

Figure 22: The Create Patch Wizard - Local Items

Create Patch from Local Modifications

This is the most commonly used type of patch and contains only the local changes for the selected item.

This option is useful if you want to share changes with other team members and you are not yet ready to commit them. This option is only available for local items that contain modifications. It is not available for items in which the local file status is *unversioned or ignored, and in some cases missing or obstructed*.

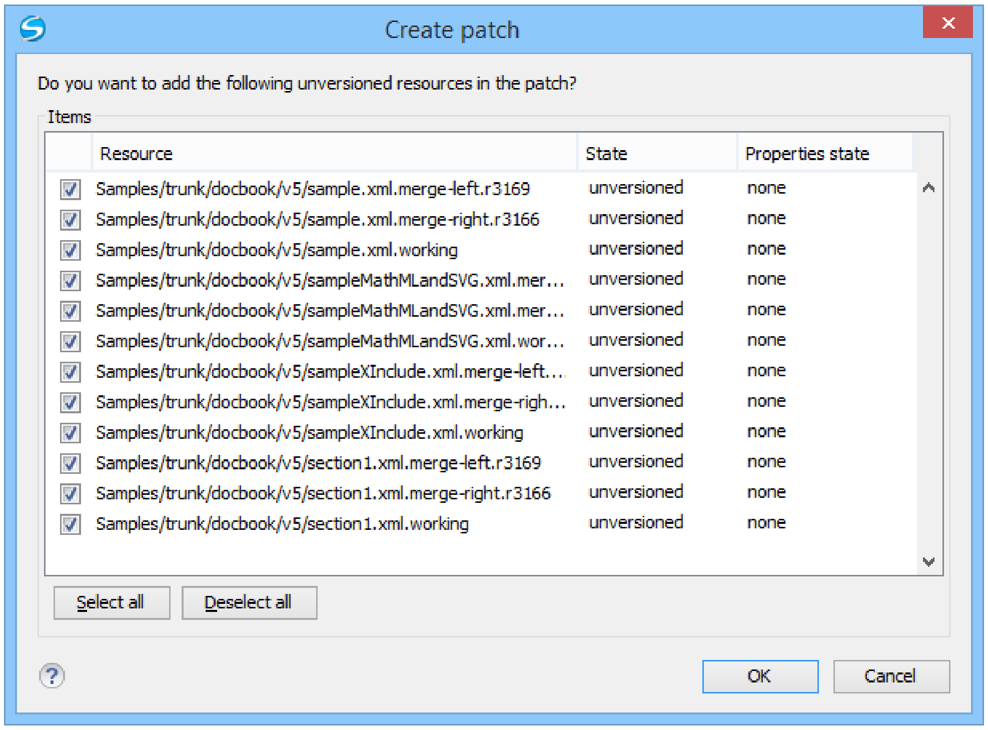
The steps are as follows:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard.
2. Select the **Create patch from local modifications** option in the dialog box.
3. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

   This options page does not remember your selections when creating future patches. It will revert to the default values.

   The **Options** wizard page is displayed.
4. Press the **Create patch** button.

   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.



Do you want to add the following unversioned resources in the patch?

Items

	Resource	State	Properties state
☑	Samples/trunk/docbook/v5/sample.xml.merge-left.r3169	unversioned	none
☑	Samples/trunk/docbook/v5/sample.xml.merge-right.r3166	unversioned	none
☑	Samples/trunk/docbook/v5/sample.xml.working	unversioned	none
☑	Samples/trunk/docbook/v5/sampleMathMLandSVG.xml.mer...	unversioned	none
☑	Samples/trunk/docbook/v5/sampleMathMLandSVG.xml.mer...	unversioned	none
☑	Samples/trunk/docbook/v5/sampleMathMLandSVG.xml.wor...	unversioned	none
☑	Samples/trunk/docbook/v5/sampleXInclude.xml.merge-left...	unversioned	none
☑	Samples/trunk/docbook/v5/sampleXInclude.xml.merge-righ...	unversioned	none
☑	Samples/trunk/docbook/v5/sampleXInclude.xml.working	unversioned	none
☑	Samples/trunk/docbook/v5/section1.xml.merge-left.r3169	unversioned	none
☑	Samples/trunk/docbook/v5/section1.xml.merge-right.r3166	unversioned	none
☑	Samples/trunk/docbook/v5/section1.xml.working	unversioned	none

Figure 23: The Create Patch Dialog Box - Add Unversioned Resources

The patch is created and stored in the path specified in *the **Output** section of the **Options** page* or in the default location.

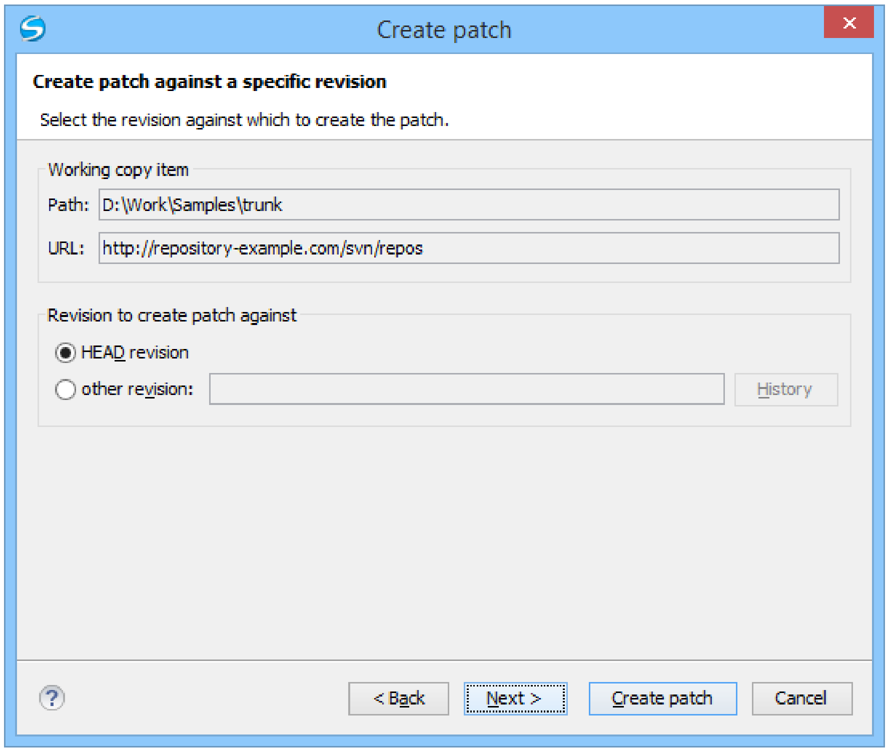
Create Patch Against a Specific Revision

This type of patch contains changes between an old revision and the current content from the selected item within the working copy.

This option is useful if you want to obtain differences between an older revision and the current state of the working copy (for instance, to test how current changes apply to an older version).

The steps are as follows:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard.
2. Select the **Create patch against a specific revision** option in the dialog box.
3. Press the **Next** button.
   The second step of the wizard is opened:


Create patch

Create patch against a specific revision

Select the revision against which to create the patch.

Working copy item

Path: D:\Work\Samples\trunk

URL: http://repository-example.com/svn/repos

Revision to create patch against

[✓ HEAD revision]

other revision:

Figure 24: The Create Patch Wizard - Step 2

4. Select the **revision to create patch against**.

   You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the **History** button* to display a list of the item revisions.

   Note: If the **revision to create patch against** is older than the revision to which the working copy item was updated, the patch will include changes that were made **after** the selected revision.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

   This options page does not remember your selections when creating future patches. It will revert to the default values.

   The **Options** wizard page is displayed.
6. Press the **Create patch** button.
   The patch is created and stored in the path specified in *the **Output** section of the **Options** page* or in the default location.

Create Patch Between Two Revisions of an Item

This type of patch contains historical changes between two revisions of a selected item.

This option is useful if you want to share changes between two revisions with other team members.

Tip: If you need to generate a patch between two revisions of a previously *deleted*, *moved*, or *replaced* item, you should use *the **Create patch between two repository items** option* instead.

The steps are as follows:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard.
2. Select the **Create patch between two revisions of an item** option in the dialog box.
3. Press the **Next** button.
   The second step of the wizard is opened:

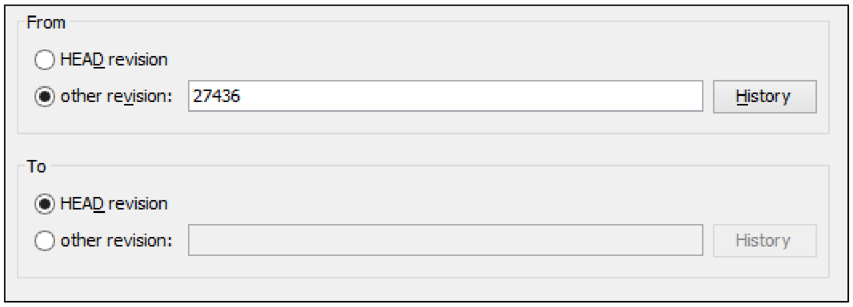


Figure 25: The Create Patch Wizard - Step 2

4. Select the starting and ending revisions in the **From** and **To** sections.

   You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the **History** button* to display a list of the item revisions.

   Note: The patch will only include changes between the two specified revisions, starting with the changes that were made **after** the older revision.

   Tip: If you want to reverse changes done between two revisions by using a patch file, you can specify the newer revision in the **From** section and the older version in the **To** section.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

   This options page does not remember your selections when creating future patches. It will revert to the default values.

   The **Options** wizard page is displayed.
6. Press the **Create patch** button.
   The patch is created and stored in the path specified in *the **Output** section of the **Options** page* or in the default location.

Create Patch Between Two Repository Items

This type of patch contains changes between one version of an item and a specific version of another item.

This option is useful for generating a patch that contains changes between existing, or even previously deleted, moved, or replaced items from different branches. This is the default option when you do not have a working copy loaded, when no repository items are selected, or when exactly two repository items of the same kind are selected.

Tip: To access an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, `URL@rev1234`).

The steps are as follows:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard.
2. Select the **Create patch between two repository items** option in the dialog box.
3. Press the **Next** button.
   The second step of the wizard is opened:

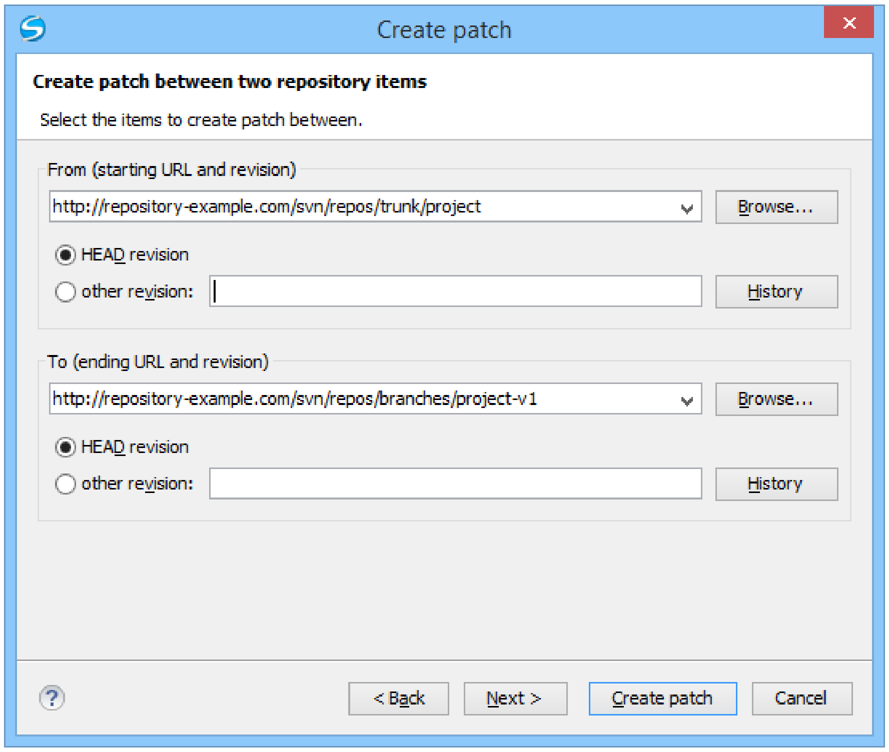

Create patch
Create patch between two repository items
Select the items to create patch between.
From (starting URL and revision)
http://repository-example.com/svn/repos/trunk/project
Browse...
HEAD revision
other revision:
History
To (ending URL and revision)
http://repository-example.com/svn/repos/branches/project-v1
Browse...
HEAD revision
other revision:
History
< Back
Next >
Create patch
Cancel

Figure 26: The Create Patch Wizard - Step 2

4. Select *the starting and ending URLs* and revisions in the **From** and **To** sections.

   You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the **History** button* to display a list of the item revisions.

   Important: Both URLs must point to items from the same repository.

   Note: If you use a *peg* revision in the URL field, anything specified in the **other revision** field is ignored.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

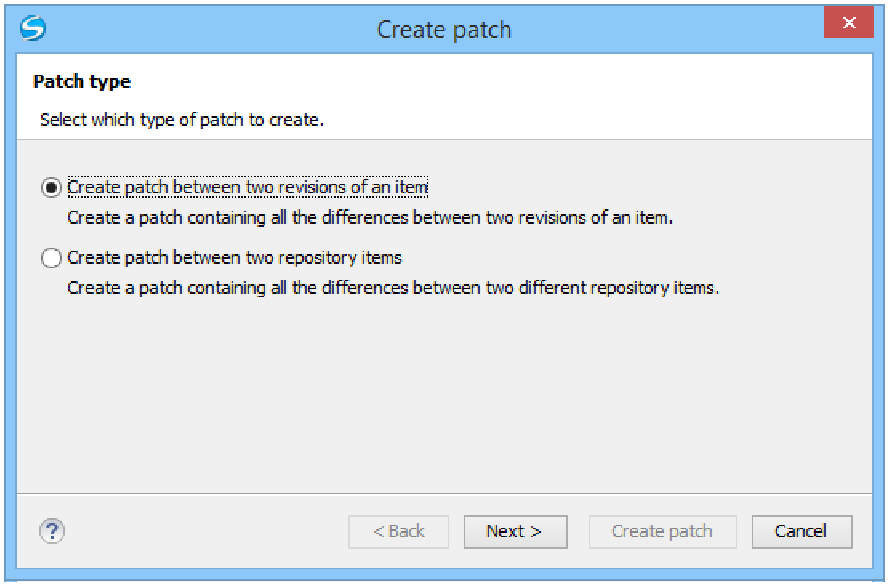
   This options page does not remember your selections when creating future patches. It will revert to the default values.

   The **Options** wizard page is displayed.
6. Press the **Create patch** button.
   The patch is created and stored in the path specified in *the **Output** section of the **Options** page* or in the default location.

Generating a Patch - Remote Items

Based on a repository item, you can generate patches that contain the differences between two specific revisions of that item, or between a revision of that same item and another revision of another item from the repository. There are two options for generating a patch based upon remote items.

To open the **Create patch** wizard, use the **Create patch...** action from the **Tools** menu.



(●) Create patch between two revisions of an item
Create a patch containing all the differences between two revisions of an item.

( ) Create patch between two repository items
Create a patch containing all the differences between two different repository items.

Figure 27: The Create Patch Wizard - Remote Items

Create Patch Between Two Revisions of an Item

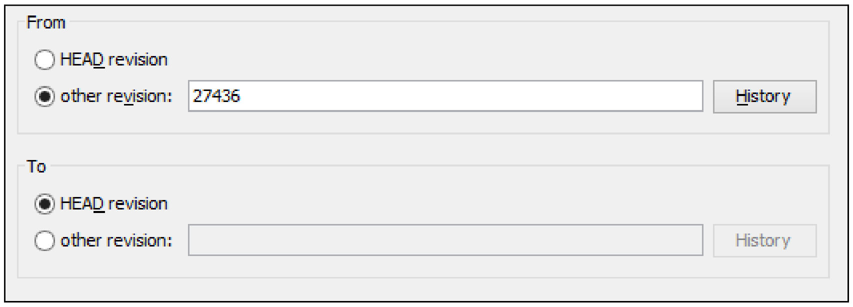
This type of patch contains historical changes between two revisions of a selected item.

This option is useful if you want to share changes between two revisions with other team members.

Tip: If you need to generate a patch between two revisions of a previously *deleted*, *moved*, or *replaced* item, you should use *the **Create patch between two repository items** option* instead.

The steps are as follows:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard.
2. Select the **Create patch between two revisions of an item** option in the dialog box.
3. Press the **Next** button.
   The second step of the wizard is opened:



From
( ) HEAD revision
(●) other revision: 27436

To
(●) HEAD revision
( ) other revision:

Figure 28: The Create Patch Wizard - Step 2

4. Select the starting and ending revisions in the **From** and **To** sections.

   You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the **History** button* to display a list of the item revisions.

   Note: The patch will only include changes between the two specified revisions, starting with the changes that were made **after** the older revision.

   Tip: If you want to reverse changes done between two revisions by using a patch file, you can specify the newer revision in the **From** section and the older version in the **To** section.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

   This options page does not remember your selections when creating future patches. It will revert to the default values.

   The **Options** wizard page is displayed.
6. Press the **Create patch** button.
   The patch is created and stored in the path specified in *the **Output** section of the **Options** page* or in the default location.

Create Patch Between Two Repository Items

This type of patch contains changes between one version of an item and a specific version of another item.

This option is useful for generating a patch that contains changes between existing, or even previously deleted, moved, or replaced items from different branches. This is the default option when you do not have a working copy loaded, when no repository items are selected, or when exactly two repository items of the same kind are selected.

Tip: To access an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, `URL@rev1234`).

The steps are as follows:

1. Go to menu **Tools** > **Create patch**.
   This opens the **Create patch** wizard.
2. Select the **Create patch between two repository items** option in the dialog box.
3. Press the **Next** button.
   The second step of the wizard is opened:

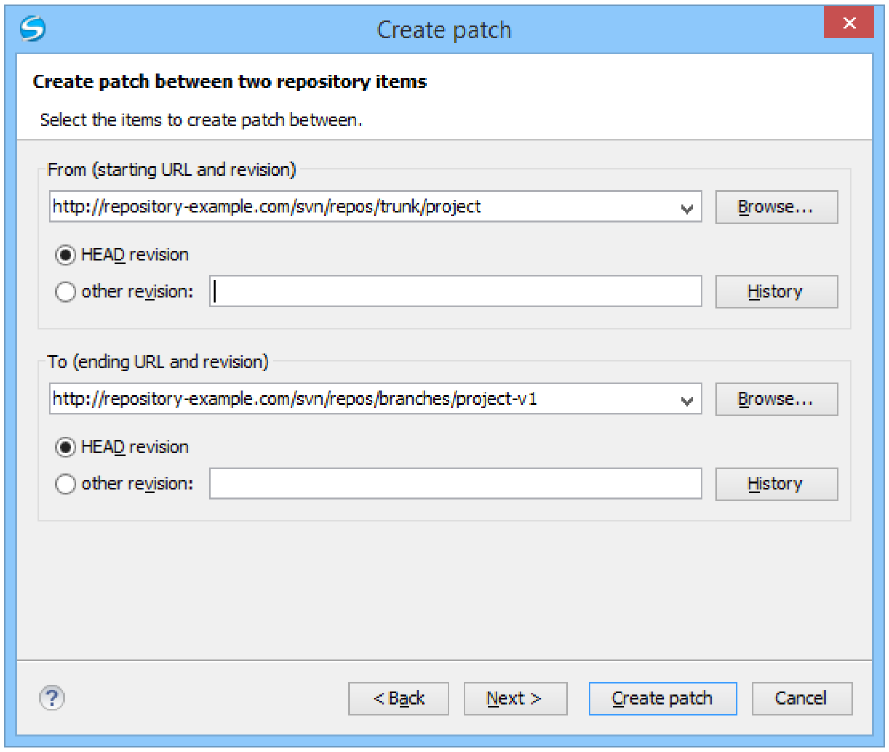

Create patch
Create patch between two repository items
Select the items to create patch between.
From (starting URL and revision)
http://repository-example.com/svn/repos/trunk/project
Browse...
HEAD revision
other revision:
History
To (ending URL and revision)
http://repository-example.com/svn/repos/branches/project-v1
Browse...
HEAD revision
other revision:
History
< Back
Next >
Create patch
Cancel

Figure 29: The Create Patch Wizard - Step 2

4. Select *the starting and ending URLs* and revisions in the **From** and **To** sections.

   You can select between the **HEAD revision** and a specific revision number. For the **other revision** option, you can press *the **History** button* to display a list of the item revisions.

   Important: Both URLs must point to items from the same repository.

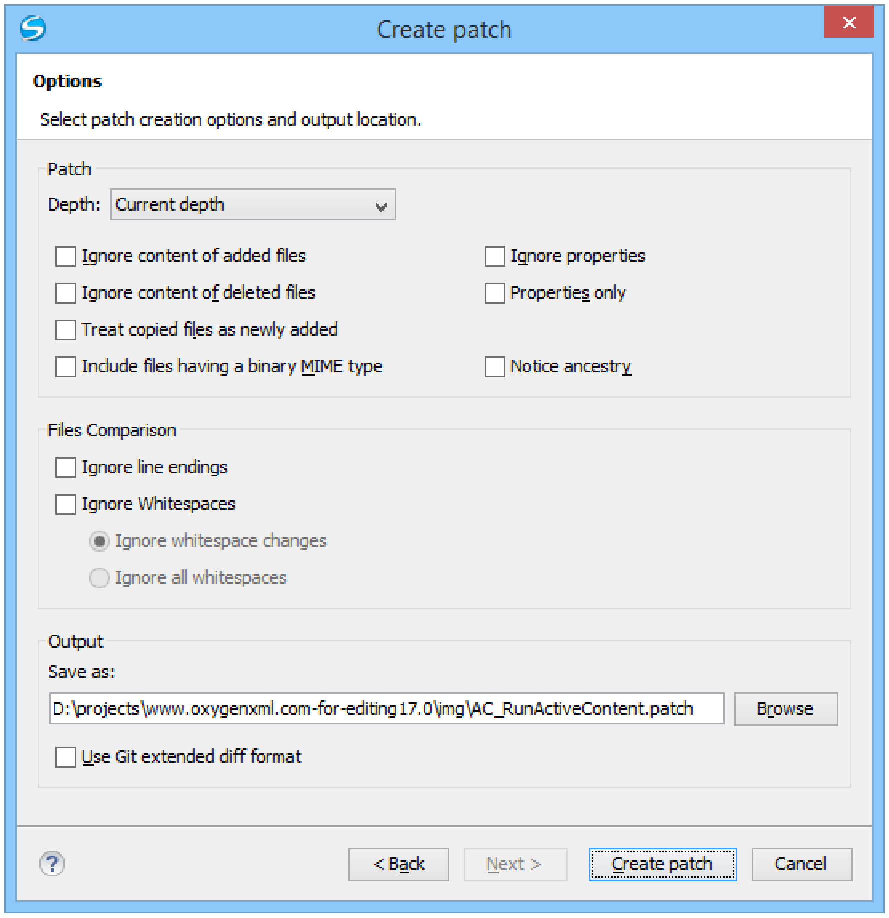
Note: If you use a *peg* revision in the URL field, anything specified in the **other revision** field is ignored.

5. Optionally, if you want *to configure the options* for your patch, press the **Next** button.

   This options page does not remember your selections when creating future patches. It will revert to the default values.

   The **Options** wizard page is displayed.
6. Press the **Create patch** button.
   The patch is created and stored in the path specified in *the **Output** section of the **Options** page* or in the default location.

Patch Options


Create patch

Options

Select patch creation options and output location.

Patch

Depth: Current depth

- [ ] Ignore content of added files
- [ ] Ignore content of deleted files
- [ ] Treat copied files as newly added
- [ ] Include files having a binary MIME type
- [ ] Ignore properties
- [ ] Properties only
- [ ] Notice ancestry

Files Comparison

- [ ] Ignore line endings
- [ ] Ignore Whitespaces
  - (●) Ignore whitespace changes
  - ( ) Ignore all whitespaces

Output

Save as: D:\projects\www.oxygenxml.com-for-editing17.0\img\AC_RunActiveContent.patch

- [ ] Use Git extended diff format

Figure 30: The Create Patch Wizard - Options

Patch Section

Depth - (This option is applicable only for directories)

Current depth - The depth of recursing the folder for creating the patch is the same as the depth of that same folder in the working copy (available only when generating patches that contain changes from 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 recursing 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 (no child file or folder is included in the patch).

Ignore content of added files

When enabled, the patch file does not include the content of the *added* items. This option corresponds to the `--no-diff-added` option of the `svn diff` command.

Ignore content of delete files

When enabled, the patch file does not include the content of the *deleted* items. This option corresponds to the `--no-diff-deleted` option of the `svn diff` command.

Treat copied files as newly added

When enabled, copied items are treated as new, rather than comparing the items with their sources. This option corresponds to the `--show-copies-as-adds` option of the `svn diff` command.

Include files having a binary MIME type

When enabled, the application is forced to compare items that are considered binary file types. This option corresponds to the `--force` option of the `svn diff` command.

Ignore properties

When enabled, differences in the properties of items are ignored. This option corresponds to the `--ignore-properties` option of the `svn diff` command.

Properties only

When enabled, only differences in the properties of the items are included in the patch file (file content is ignored). This option corresponds to the `--properties-only` option of the `svn diff` command.

Note: The **Ignore properties** and **Properties only** options are mutually exclusive.

Notice ancestry

If enabled, the SVN common ancestry is taken into consideration when calculating the differences. This option corresponds to the `--notice-ancestry` option of the `svn diff` command.

Files Comparison Section

Ignore line endings

If enabled, the differences in line endings are ignored when the patch is generated. This option corresponds to the `--ignore-eol-style` option of the `svn diff` command.

Ignore whitespaces

If enabled, it allows you to specify how the whitespace changes should be handled. When enabled, you can then choose between two options:

- **Ignore whitespace changes** (`--ignore-space-change`) - Ignores changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces).

  Note: Whitespaces that were added where there were none before, or that were removed, are still considered to be changes.

- **Ignore all whitespaces** (`--ignore-all-space`) - Ignores all types of whitespace changes.

Output Section

Save as

The patch will be created and saved in the specified file.

Use Git extended diff format

When enabled, the patch is generated using the *Git* format. This option corresponds to the `--git` option of the `svn diff` command.

Working with Repositories

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

Importing Resources Into a Repository

Importing resources into a repository is the process of copying local files and directories into a repository so that they can be managed by an Apache Subversion™ 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*.

The **Import folder...** and **Import Files...** actions are available from the **Import** submenu of the **Repository** menu or of the contextual menu in the **Repositories** view. These actions open a dialog box that allow you to select the directories or files that will be imported into the selected repository location.

The **Import folder...** action opens the **Import folder** dialog box.

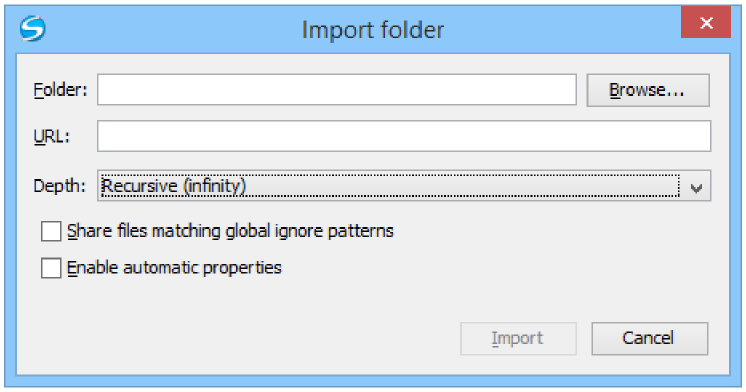


Figure 31: The Import Folder Dialog Box

You can configure the following options:

Folder

Specify *the local folder* by using the text box or the **Browse** button. This folder should not be empty or already under version control.

Important: By default, the SVN system only imports the content of the specified folder, and not the folder itself. Therefore, it is recommended to use the **Browse** button to select the local folder so that the client will automatically append the name of it to the specified URL.

URL

Specify *the repository location* that will be used to access the folder to be imported.

Note: *Peg revisions* have no effect for this operation since it is used to send information to the repository.

Attention: If the new location already exists, make sure that it is an empty directory to avoid mixing your project content with other files (if items exist with the same name, an error will occur and the operation will not proceed). Otherwise, if the address does not exist, it is created automatically.

Depth

Recursive (infinity) - Imports all the files and folders contained in the selected folder.

Immediate children (immediates) - Imports only the child files and folders without recursing subfolders.

File children only (files) - Import only the child files.

This folder only (empty) - Imports only the selected folder (no child file or folder is included).

Share files matching global ignore patterns

When enabled, the file names that match the patterns defined in either of the following locations are also imported into the repository:

- The `global-ignores` property in *the SVN configuration file*.
- The ***File name ignore patterns*** *option* in the ***SVN*** *>* ***Working Copy*** *preferences page*.

Enable automatic properties/Disable automatic properties

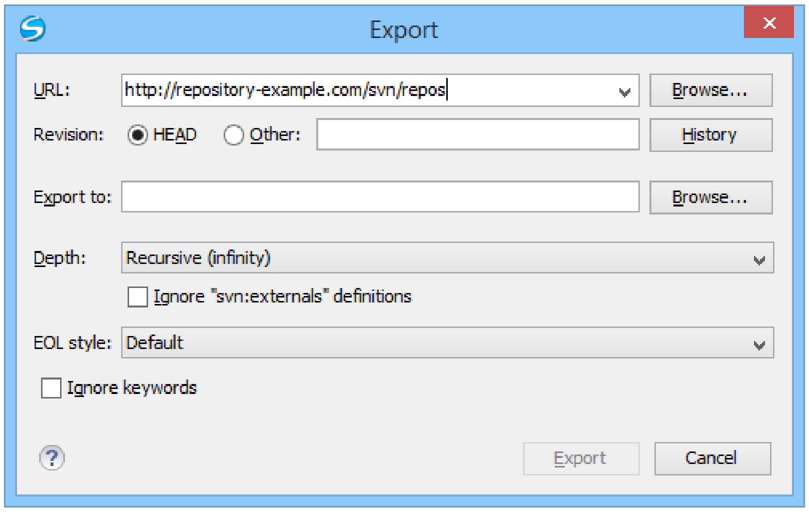
Enables or disables automatic property assignment (per runtime configuration rules), overriding the `enable-auto-props` runtime configuration directive, defined in *the SVN configuration file*.

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

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:



Export

URL: http://repository-example.com/svn/repos

Revision: ● HEAD ○ Other:

Export to:

Depth: Recursive (infinity)

☐ Ignore "svn:externals" definitions

EOL style: Default

☐ Ignore keywords

Figure 32: Export from Repository

You can configure the following options:

URL

Specify *the source directory from the repository* by using the text box or the **Browse** button.

Tip: To export an item that was deleted, moved, or replaced, you need to specify the original URL (before the item was removed) and use a *peg revision* at the end (for example, `URL@rev1234`).

Note: The content of the selected directory from the repository and not the directory itself will be exported to the file system.

Revision

You can choose between the **HEAD** or **Other** revision. If you need to export a specific revision, specify it in the **Other** text box or use the **History** button and choose a revision from the **History** dialog box.

Export to

Specify *the location where you want to export* the repository directory by typing the local path in the text box or by using the **Browse** button. If the specified local path does not point to an existing directory, it will automatically be created.

Important: By default, the SVN system only exports the content of the directory specified by the URL, and not the directory itself. Therefore, it is recommended to use the **Browse** button to select the *export* location so that the client will automatically append the name of the remote directory to the path of the selected directory.

Warning: The destination directory should be empty. If files exist, they will be overwritten by exported files with matching names.

Depth

Recursive (infinity) - Exports all the files and folders contained in the selected folder.

Immediate children (immediates) - Exports only the child files and folders without recursing subfolders.

File children only (files) - Export only the child files.

This folder only (empty) - Exports only the selected folder (no child file or folder is included).

Ignore "svn:externals" definitions

When enabled, external items are ignored in the export operation. This option is only available if you choose the **Recursive (infinity)** depth.

EOL style

Defines the *end-of-line (EOL)* marker that should be used when exporting files that have the value or the `svn:eol-style` property set to `native`. You can choose between the following styles:

- **Default** - It uses the system-specific *end-of-line* marker.
- **CRLF** - The **Windows**-specific *end-of-line* marker (*carriage return - line feed*).
- **LF** - The **Unix / OS X**-specific *end-of-line* marker (*line feed*).
- **CR** - The **Mac OS 9 (or older)**-specific *end-of-line* marker (*carriage return*).

Ignore keywords

When enabled, the export operation does not expand the *SVN keywords* found inside the files.

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.

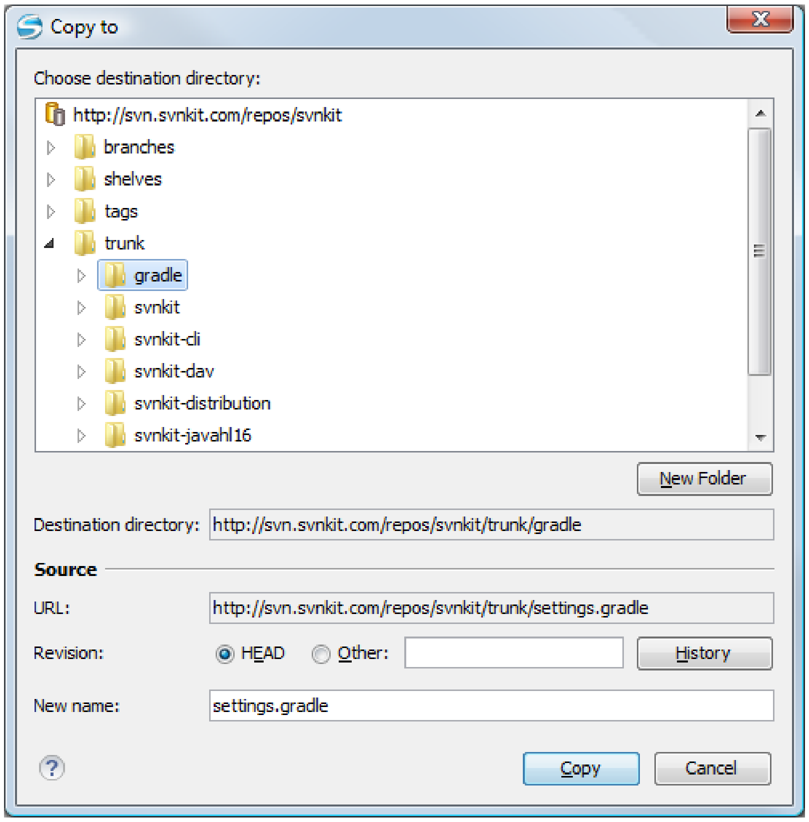

Copy to
Choose destination directory:
http://svn.svnkit.com/repos/svnkit
branches
shelves
tags
trunk
gradle
svnkit
svnkit-cli
svnkit-dav
svnkit-distribution
svnkit-javahl16
New Folder
Destination directory: http://svn.svnkit.com/repos/svnkit/trunk/gradle
Source
URL: http://svn.svnkit.com/repos/svnkit/trunk/settings.gradle
Revision: HEAD Other:
History
New name: settings.gradle
Copy
Cancel

Figure 33: 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.

Syncro SVN Client Views

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

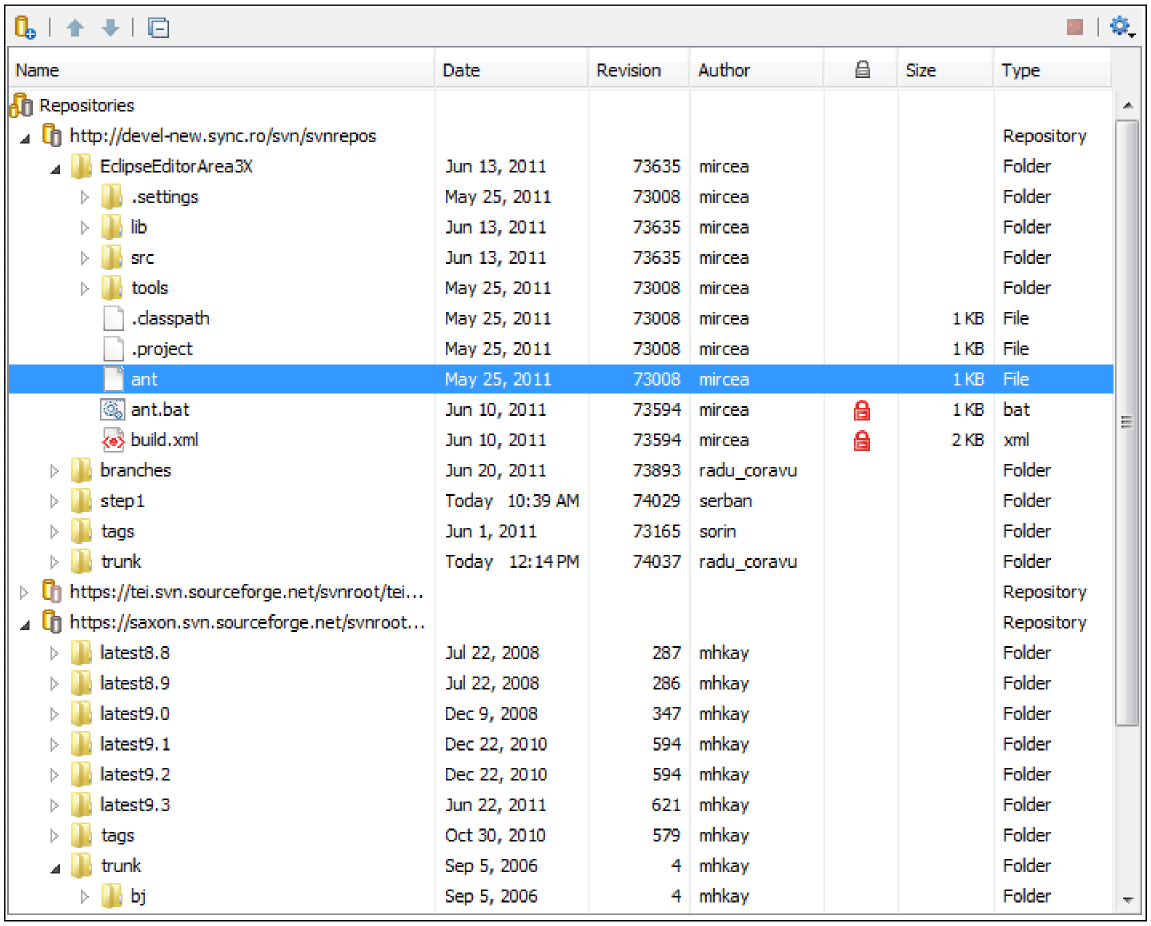
- *Repositories View*
- *Working Copy View*
- *History View*
- *Console View*

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™ repository locations and browse repositories. If no connections to your repository are available, you can *add a new repository location*. 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 34: 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+Alt+N (Command+Alt+N on OS X))

Displays the **Add SVN Repository** dialog. This dialog allows you to define a new repository location.

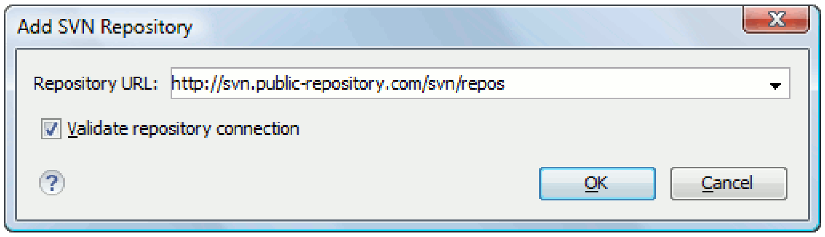

Add SVN Repository
Repository URL:
http://svn.public-repository.com/svn/repos
Validate repository connection
OK
Cancel

Figure 35: 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+Alt+E (Command+Alt+E on OS X))

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+Alt+Shift+B (Command+Alt+Shift+B on OS X))

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+Alt+Shift+R (Command+Alt+Shift+R on OS X))

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 (F5)

Refreshes the resource selected in the **Repositories** view.

Check out... (Ctrl+Alt+Shift+C (Command+Alt+Shift+C on OS X))

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:

Import folder... (Ctrl+Alt+Shift+M (Command+Alt+Shift+M on OS X))

Allows you to import the contents of a specified folder from the file system into the selected folder in a repository. To read more about this operation, see the section *Importing resources into a repository*.

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

Import Files... (Ctrl+Alt+I (Command+Alt+I on OS X))

Imports the files selected from the files system into the selected folder in the repository.

Export...

Opens *the **Export** dialog box* that allows you to configure options for exporting a folder from the repository to the local file system.

Show History... (Ctrl+H (Command+H on OS X))

Displays the history of the selected resource. At the start of the operation, you can set filtering options.

Show Annotation... (Ctrl+Shift+A (Command+Shift+A on OS X))

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the **Annotations** view*, along with the history of the file in the **History** view.

Revision Graph (Ctrl+Shift+G (Command+Shift+G on OS X))

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+Alt+U (Command+Alt+U on OS X))

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+M (Command+M on OS X))

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... (Ctrl+Shift+F (Command+Shift+F on OS X))

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

Locking

(available only for files):

Lock... (Ctrl+K (Command+K on OS X))

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+Shift+K (Command+Shift+K on OS X))

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+Shift+P (Command+Shift+P on OS X))

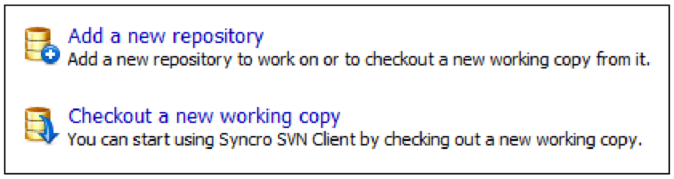
Brings up the *Properties view* displaying the SVN properties for the selected resource. This view does not allow adding, editing, or removing SVN properties of a repository resource. These operations are allowed only for working copy resources.

Show SVN Information (Ctrl+I (Command+I on OS X))

Provides additional information for the selected resource. For more details, go to *Obtain information for a resource*.

Assistant Actions

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



Add a new repository
Add a new repository to work on or to checkout a new working copy from it.

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

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

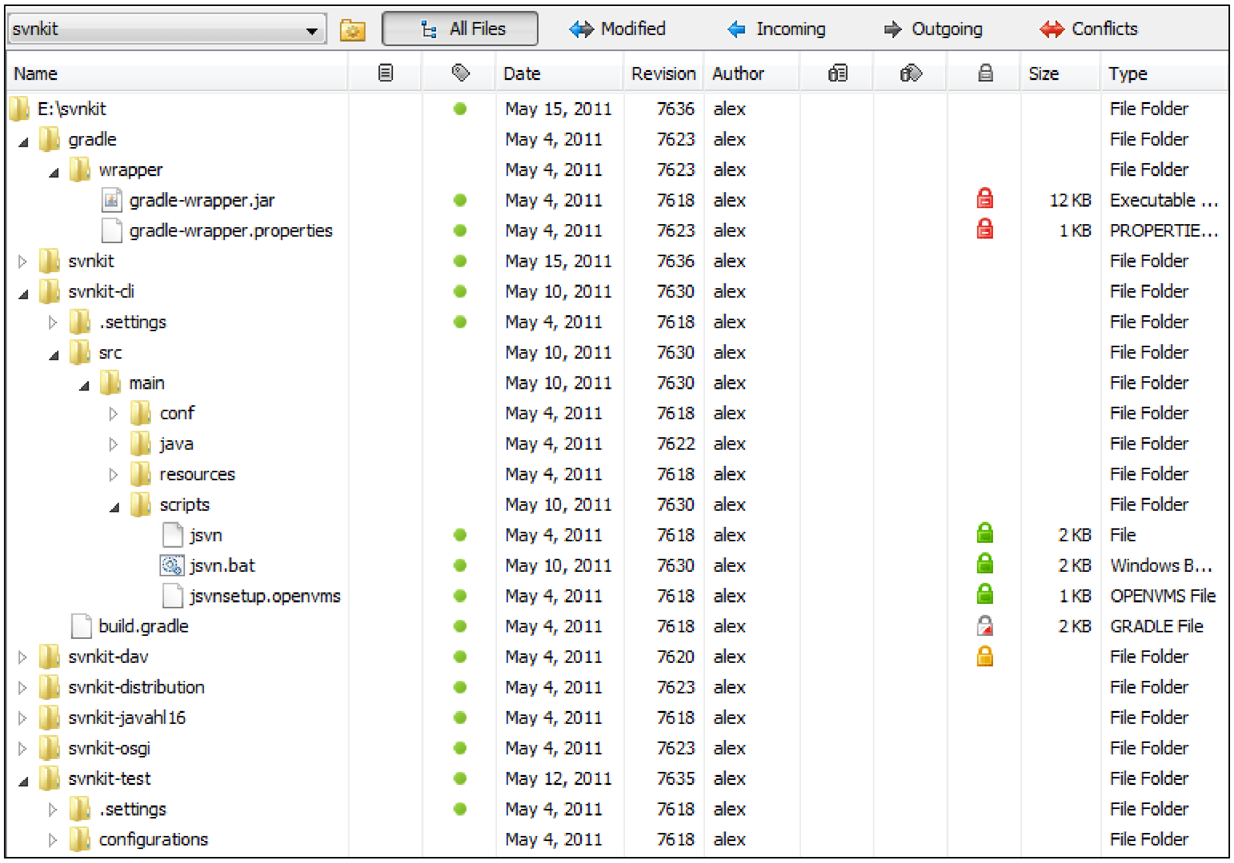
The **Working Copy** view allows you to manage the content of an SVN working copy.

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:

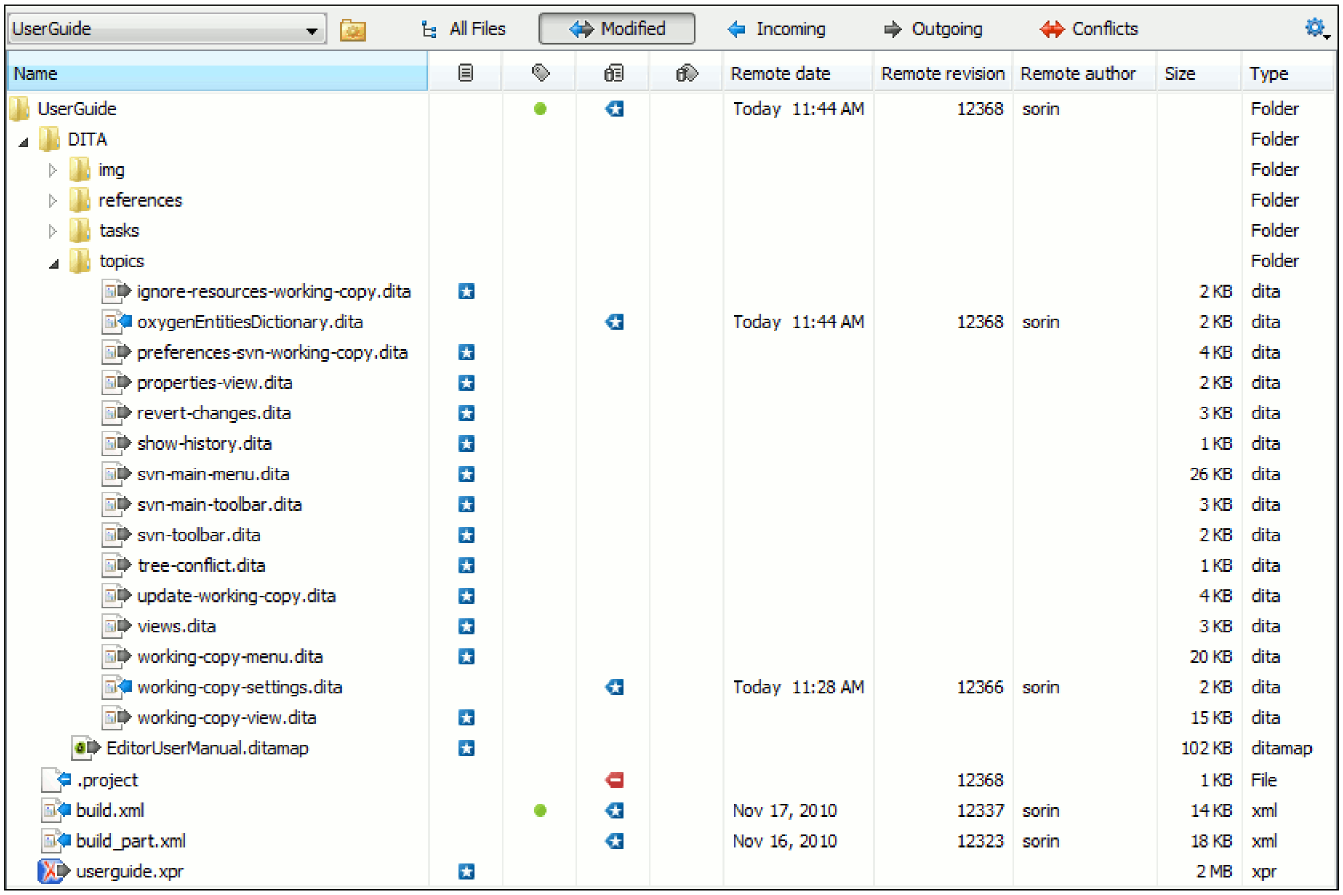
- **All Files** - Resources (files and folders) are presented in a hierarchical structure with the root of the tree representing the location of the working copy on the file system. Each resource has an icon representation which describes the type of resource and also depicts the state of that resource with a small overlay icon.



Name			Date	Revision	Author				Size	Type
E:\svnkit		●	May 15, 2011	7636	alex					File Folder
gradle			May 4, 2011	7623	alex					File Folder
wrapper			May 4, 2011	7623	alex					File Folder
gradle-wrapper.jar		●	May 4, 2011	7618	alex			🔒	12 KB	Executable ...
gradle-wrapper.properties		●	May 4, 2011	7623	alex			🔒	1 KB	PROPERTIE...
svnkit		●	May 15, 2011	7636	alex					File Folder
svnkit-cli		●	May 10, 2011	7630	alex					File Folder
.settings		●	May 4, 2011	7618	alex					File Folder
src			May 10, 2011	7630	alex					File Folder
main			May 10, 2011	7630	alex					File Folder
conf			May 4, 2011	7618	alex					File Folder
java			May 4, 2011	7622	alex					File Folder
resources			May 4, 2011	7618	alex					File Folder
scripts			May 10, 2011	7630	alex					File Folder
jsvn		●	May 4, 2011	7618	alex			🔒	2 KB	File
jsvn.bat		●	May 10, 2011	7630	alex			🔒	2 KB	Windows B...
jsvnsetup.openvms		●	May 4, 2011	7618	alex			🔒	1 KB	OPENVMS File
build.gradle		●	May 4, 2011	7618	alex			🔒	2 KB	GRADLE File
svnkit-dav		●	May 4, 2011	7620	alex			🔒		File Folder
svnkit-distribution		●	May 4, 2011	7623	alex					File Folder
svnkit-javahl16		●	May 4, 2011	7618	alex					File Folder
svnkit-osgi		●	May 4, 2011	7623	alex					File Folder
svnkit-test		●	May 12, 2011	7635	alex					File Folder
.settings		●	May 4, 2011	7618	alex					File Folder
configurations			May 4, 2011	7618	alex					File Folder

Figure 36: 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).



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 37: 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 and pseudo-conflicts).

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*.

      Attention: If you don't explicitly set the `global-ignores` runtime configuration option - either to your preferred set of patterns or to an empty string - Subversion uses the default value.

    - ***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).

- **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;

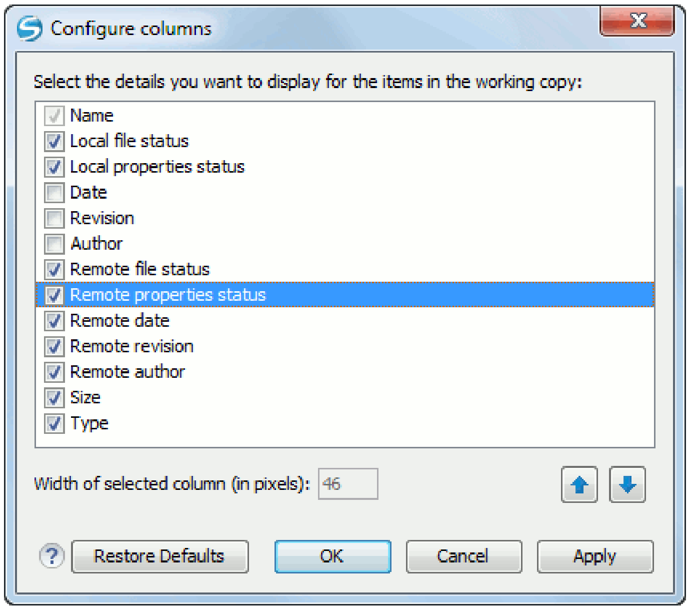
  > **Note:** In case this option is disabled, it will be ignored for items that, after a synchronize, are reported as incoming from the repository. This applies for all working copy modes, except **All Files**.

- **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:


Configure columns

Select the details you want to display for the items in the working copy:

- [✓] Name
- [✓] Local file status
- [✓] Local properties status
- [ ] Date
- [ ] Revision
- [ ] Author
- [✓] Remote file status
- [✓] Remote properties status
- [✓] Remote date
- [✓] Remote revision
- [✓] Remote author
- [✓] Size
- [✓] Type

Width of selected column (in pixels): 46

Figure 38: 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, you are prompted to upgrade it to SVN 1.8 in order to load it;
- if the format is 1.7, Syncro SVN Client takes into account the state of the ***When loading an old format working copy*** option.

To change how working copy formats are handled, *open the **Preferences** dialog* and go to **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 1.7 and SVN 1.8 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 (Command on OS X) + 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 (Command on OS X) + Alt + C)

Displays changes made in the currently selected file.

Open (Ctrl (Command on OS X) + 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 OS X, etc).

Open with... (Ctrl (Command on OS X) + 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 (Command on OS X) + Alt + X)

Displays all descendants of the selected folder. The same behavior is obtained 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 (Command on OS X) + 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 (Command on OS X)+ 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 resources to commit. A directory will always be committed recursively. Unversioned resources will be deselected by default. In the **Commit** dialog you can also enter a comment before sending your changes to the repository.

Revert... (Ctrl (Command on OS X) + Shift + V)

Undoes all local changes for the selected resources. It does not contact the repository and the files are obtained from Apache Subversion™ 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. This action is available on resources with outgoing changes, including 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. This action is available on conflicting resources. For more information see *Drop incoming modifications*.

Mark Resolved (Ctrl (Command on OS X) + Shift + R)

Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.

Mark as Merged (Ctrl (Command on OS X) + 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 (Command on OS X) + 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 (Command on OS X) + 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 (Command on OS X) + Alt + C)** - Compares the working copy file with the BASE revision file (the so-called *pristine copy*).
- **Revision (Ctrl (Command on OS X) + Alt + R)** - Shows the **History view** containing the log history of that resource.
- **Branch/Tag** - Opens the **Compare with Branch/Tag** dialog box that allows you to specify *another file from the repository* (**To URL** field) to compare with the working copy file. You can specify the revision of the repository file by choosing between **HEAD revision** or specific **Other revision**.

  Tip: To compare with a file that was deleted, moved, or replaced, you need to specify the original URL (before the file was removed) and use a *peg revision* at the end (for example, `URL@rev1234`).

- **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 currently selected resources with their versions from the *BASE/HEAD* revision:

- For the **Replace with BASE revision** action, the resources being unversioned or added have no *BASE* revision, and thus 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), since you cannot recover a tree of folders
- For the **Replace with latest from HEAD** action, you must be aware that there are cases when resources will be completely deleted or reverted to the BASE revision and then updated to a HEAD revision 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 that is locally added on the parent folder and not yet committed to the repository

Show History... (Ctrl (Command on OS X) + 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+Shift+A (Command+Shift+A on OS X))

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the* ***Annotations*** *view*, along with the history of the file in the **History** view.

Revision Graph (Ctrl (Command on OS X) + G)

This action allows you to see the graphical representation history of a resource. For more details about the revision graph of resources, see *Revision Graph*.

Copy URL Location (Ctrl (Command on OS X) + 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 item and source item), and it depends on the state of the source item.

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 item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

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 item and the source item that is usually reported as *missing*), and it depends on the state of the source item.

Copy to...

Copies the currently selected resource to a specified location.

Move to... Ctrl+M (Command+M on OS X)

Moves the currently selected resource to a specified location.

Rename... (F2)

As with the move command, a copy of the original resource will be made with the new name and the original will be marked as deleted. Note that you can only rename one resource at a time.

Delete (Delete)

Schedules selected items for deletion upon the next commit and removes 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 (Command on OS X)+ 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 (Command on OS X) + Shift + W)

This operation allows you to add a new external definition on the selected folder. An external definition is a mapping of a local directory to *a URL of a versioned directory*, and ideally a particular revision, stored in the `svn:externals` property of the selected folder.

Tip: You can specify a particular revision of the external item by using a *peg revision* at the end of the URL (for example, `URL@rev1234`). You can also use peg revisions to access external items that were deleted, moved, or replaced.

The URL used in the external definition format can be relative. You can specify the repository URL that the external folder points to by using one of 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 in which the `svn:externals` property is versioned.

Important: To change the target URL of an external definition, or to delete an external item, do the following:

1. Modify or delete the item definition found in the `svn:externals` property that is set on the parent folder.
2. For the change to take effect, use the **Update** operation on the parent folder of the external item.

Note: Syncro SVN Client does not support definitions of local relative external items.

Add to "svn:ignore"... (Ctrl (Command on OS X) + Alt + I)

Allows you to add files that should not participate in the *version control* operations inside your working copy . This action can only be performed on resources not under *version control*. It actually modifies the value of the `svn:ignore` property in the parent directory of the resource. Read more about this in the *Ignore Resources Not Under Version Control* section.

Add to version control... (Ctrl (Command on OS X) + Alt + V)

Allows you to add 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 (Command on OS X) + Shift + C)

Performs a maintenance cleanup operation on the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. This is useful when you already know where the problem originated and want to fix it as quickly as possible. It is only active for resources under *version control*.

Locking:

- **Scan for locks... (Ctrl (Command on OS X) + 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. This is only active for resources under *version control*. For more details see *Scanning for locks*.
- **Lock... (Ctrl (Command on OS X) + K)** - Allows you to lock certain files that need exclusive access. You can write a comment describing the reason for the lock and you can also force (*steal*) the lock. This action is active only on files under *version control*. For more details on the use of this action see *Locking a file*.
- **Unlock... (Ctrl (Command on OS X) + 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+P (Command+P on OS X))

Brings up the *Properties view* and displays the SVN properties for the selected resource.

Show SVN Information (Ctrl+I (Command+I on OS X))

Provides additional information for the selected resource from the working copy. For more details, go to *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.

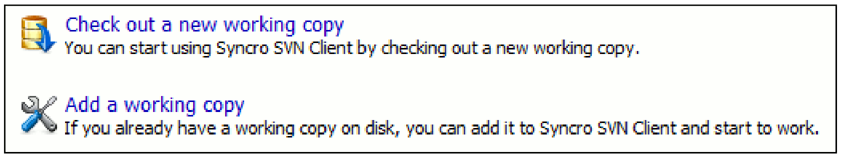
Also, files and folders can be added to the file tree of the view as *unversioned* resources by drag and drop operations from other applications (for example from Windows Explorer or Mac OS X Finder). In this case, the items from the file system are only copied, without removing them from their original 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 39: 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:

- 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™, 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:

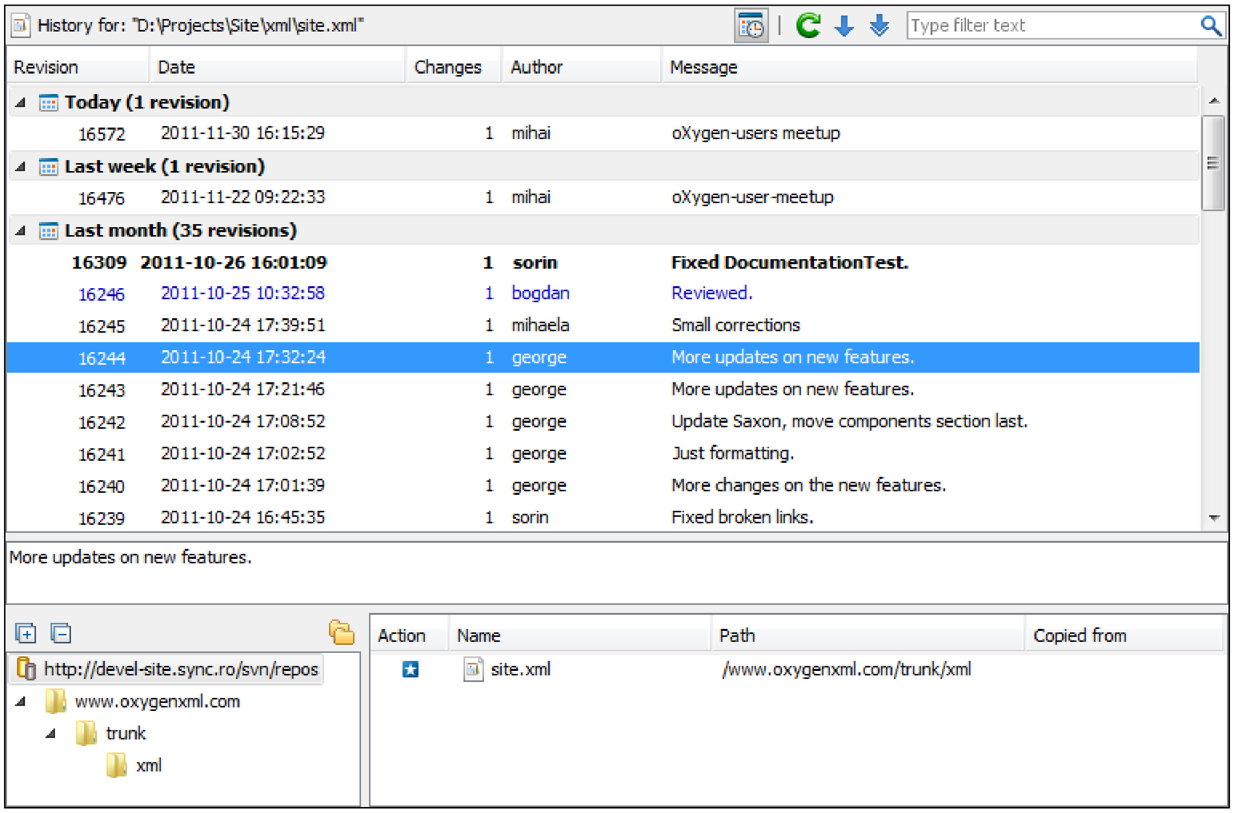
- The table showing details about each revision, like: revision number, commit date and time, number of changes (more details available in the tooltip), author's name, and a fragment of the commit message.

  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*.
  - - Resource was *replaced* in the repository.
  - - Resource was deleted from the repository.


History for: "D:\Projects\Site\xml\site.xml"

Revision	Date	Changes	Author	Message
Today (1 revision)				
16572	2011-11-30 16:15:29	1	mihai	oXygen-users meetup
Last week (1 revision)				
16476	2011-11-22 09:22:33	1	mihai	oXygen-user-meetup
Last month (35 revisions)				
16309	**2011-10-26 16:01:09**	**1**	**sorin**	**Fixed DocumentationTest.**
16246	2011-10-25 10:32:58	1	bogdan	Reviewed.
16245	2011-10-24 17:39:51	1	mihaela	Small corrections
16244	2011-10-24 17:32:24	1	george	More updates on new features.
16243	2011-10-24 17:21:46	1	george	More updates on new features.
16242	2011-10-24 17:08:52	1	george	Update Saxon, move components section last.
16241	2011-10-24 17:02:52	1	george	Just formatting.
16240	2011-10-24 17:01:39	1	george	More changes on the new features.
16239	2011-10-24 16:45:35	1	sorin	Fixed broken links.

More updates on new features.

```
http://devel-site.sync.ro/svn/repos
  www.oxygenxml.com
    trunk
      xml
```

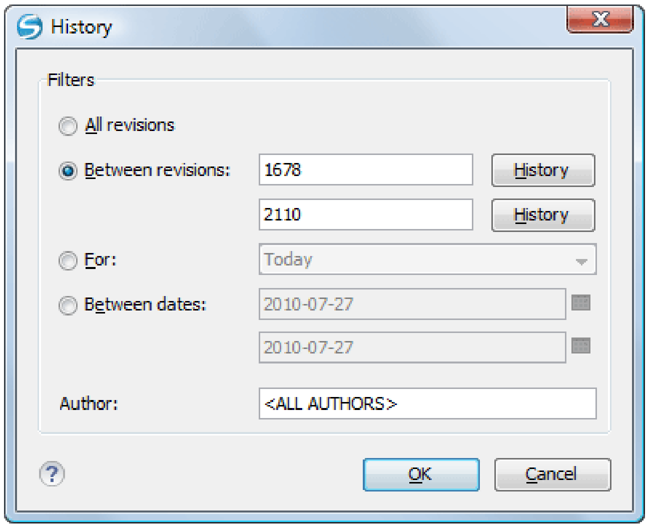
Action	Name	Path	Copied from
	site.xml	/www.oxygenxml.com/trunk/xml	

Figure 40: 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.



Between revisions: 1678 – 2110
For: Today
Between dates: 2010-07-27 – 2010-07-27
Author: <ALL AUTHORS>

Figure 41: 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.

Contextual Menu Actions

The **History** view contains the following contextual menu 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 that were made in the selected revisions. The changes are reverted only in your working copy and does not affect the repository items. It does not replace your working copy items with those from the selected revisions. This action is enabled when the resource history was launched for a local working copy resource.

Note: For items displayed in the **Affected Paths** section that were *added*, *deleted*, or *replaced*, this action has no effect because such changes are considered to be changes to the parent directory. To revert these type of changes, follow these steps:

1. Request the history for the parent directory.
2. Identify the revision that contains the changes you want to revert.
3. Invoke the action on that revision.

Warning: There are instances where the SVN Client is not able to identify the corresponding working copy item for the selected item in the **Affected Paths** section. In this case, the action does not proceed and an error message is displayed. For example, the selected item in the **Affected Paths** section is from a different repository location than the working copy item for which the history is displayed.

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.

Export...

Opens *the **Export** dialog box* that allows you to configure options for exporting a folder from the repository to the local file system.

Show Annotation... (Ctrl+Shift+A (Command+Shift+A on OS X))

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the **Annotations** view*, along with the history of the file in the **History** view.

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.

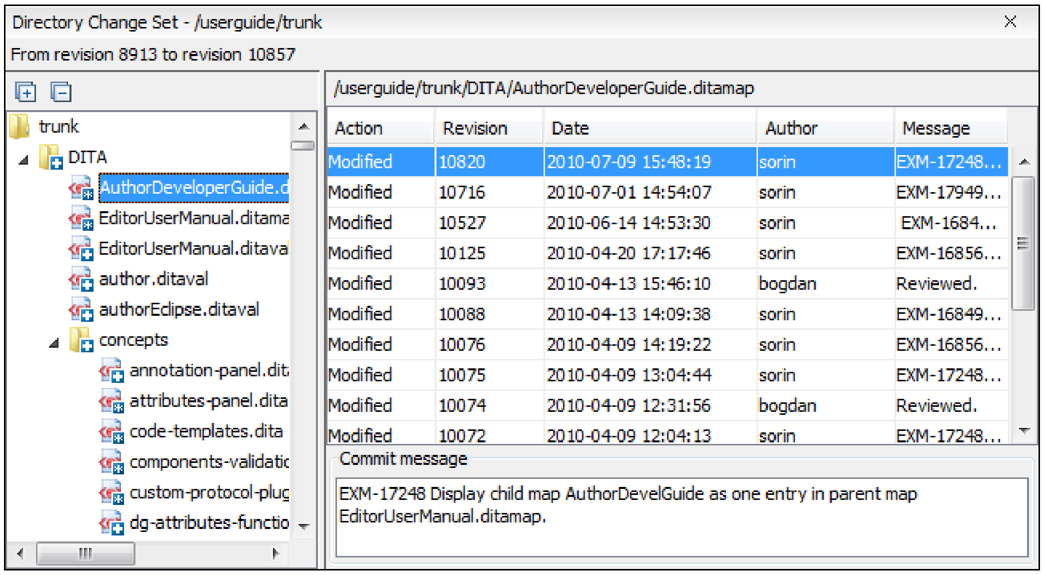
Revert changes from these revisions

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.



Directory Change Set - /userguide/trunk

From revision 8913 to revision 10857

- trunk
  - DITA
    - AuthorDeveloperGuide.d
    - EditorUserManual.ditama
    - EditorUserManual.ditava
    - author.ditaval
    - authorEclipse.ditaval
    - concepts
      - annotation-panel.dit
      - attributes-panel.dita
      - code-templates.dita
      - components-validatic
      - custom-protocol-plug
      - dg-attributes-functio

/userguide/trunk/DITA/AuthorDeveloperGuide.ditamap

Action	Revision	Date	Author	Message
Modified	10820	2010-07-09 15:48:19	sorin	EXM-17248...
Modified	10716	2010-07-01 14:54:07	sorin	EXM-17949...
Modified	10527	2010-06-14 14:53:30	sorin	EXM-1684...
Modified	10125	2010-04-20 17:17:46	sorin	EXM-16856...
Modified	10093	2010-04-13 15:46:10	bogdan	Reviewed.
Modified	10088	2010-04-13 14:09:38	sorin	EXM-16849...
Modified	10076	2010-04-09 14:19:22	sorin	EXM-16856...
Modified	10075	2010-04-09 13:04:44	sorin	EXM-17248...
Modified	10074	2010-04-09 12:31:56	bogdan	Reviewed.
Modified	10072	2010-04-09 12:04:13	sorin	EXM-17248...

Commit message

EXM-17248 Display child map AuthorDevelGuide as one entry in parent map EditorUserManual.ditamap.

Figure 42: 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 contextual 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.

Export...

Opens *the **Export** dialog box* that allows you to configure options for exporting a folder from the repository to the local file system.

Show Annotation... (Ctrl+Shift+A (Command+Shift+A on OS X))

Opens the **Show Annotation** dialog box that computes *the annotations for a file and displays them in the **Annotations** view*, along with the history of the file in the **History** view.

Show SVN Information (Ctrl (Command on OS X) + I)

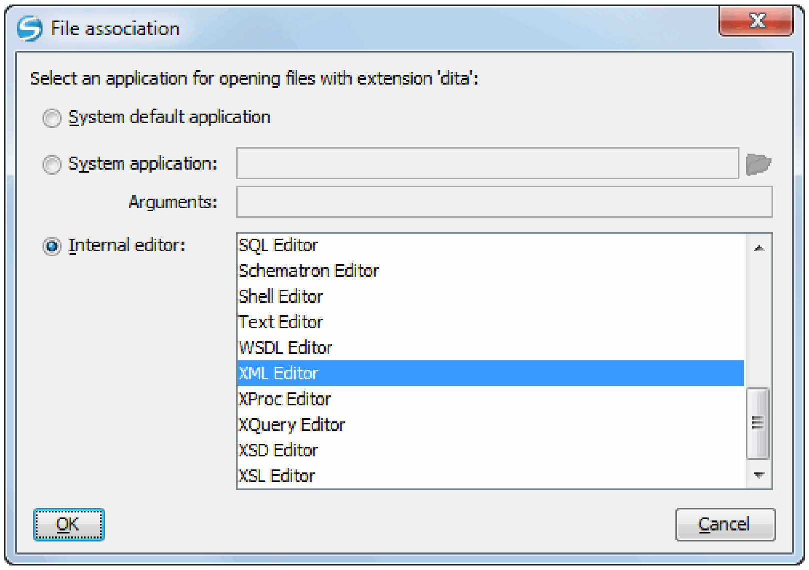
Provides additional information for a selected resource. For more details, go to *Obtain information for a resource*.

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:



Select an application for opening files with extension 'dita':

System default application

System application:

Arguments:

[✓ Internal editor:]

SQL Editor
Schematron Editor
Shell Editor
Text Editor
WSDL Editor
XML Editor
XProc Editor
XQuery Editor
XSD Editor
XSL Editor

Figure 43: 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.

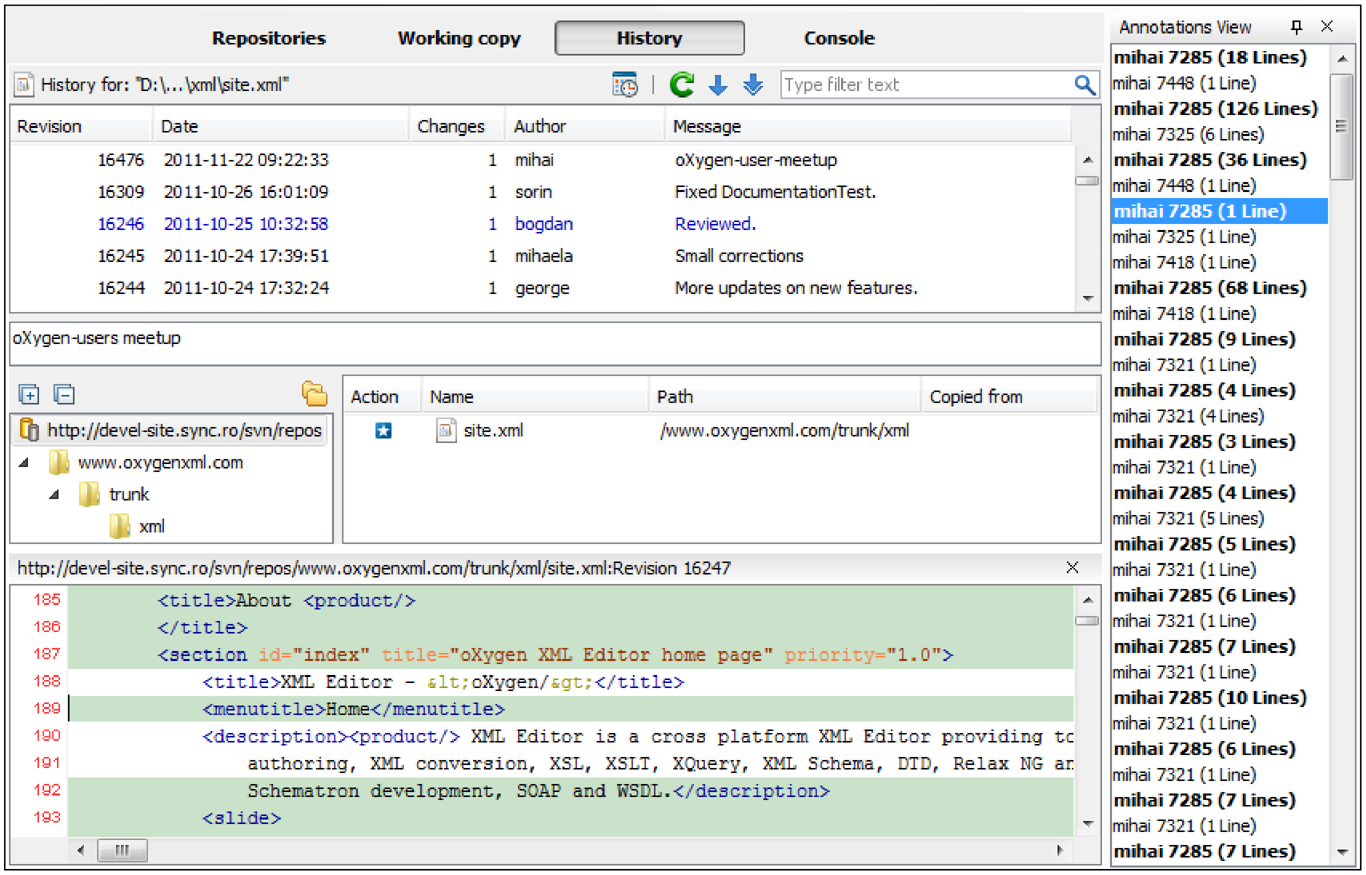
Only one file at a time can be edited in an internal editor. If you try to open another file it will be opened in the same editor window. The editor provides syntax highlighting for known file types. This means that a different color will be used for each recognized token type found in the file. If the 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+S (Command+S on OS X)** 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 revision and the author 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 in which the line was last modified. The same revision is highlighted in the **History view** and you can also see all the lines that were changed in the same revision highlighted in the editor panel. Also, the entries of the **Annotations view** corresponding to that revision are highlighted. Therefore, the **Annotations view**, **History view**, and annotations editor panel are all synchronized. Clicking on a line in one of them highlights the corresponding lines in the other two.



History for: "D:\...\xml\site.xml"

Revision	Date	Changes	Author	Message
16476	2011-11-22 09:22:33	1	mihai	oXygen-user-meetup
16309	2011-10-26 16:01:09	1	sorin	Fixed DocumentationTest.
16246	2011-10-25 10:32:58	1	bogdan	Reviewed.
16245	2011-10-24 17:39:51	1	mihaela	Small corrections
16244	2011-10-24 17:32:24	1	george	More updates on new features.

oXygen-users meetup

http://devel-site.sync.ro/svn/repos
www.oxygenxml.com
trunk
xml

Action	Name	Path	Copied from
	site.xml	/www.oxygenxml.com/trunk/xml	

http://devel-site.sync.ro/svn/repos/www.oxygenxml.com/trunk/xml/site.xml:Revision 16247

```
        <title>About <product/>
        </title>
        <section id="index" title="oXygen XML Editor home page" priority="1.0">
            <title>XML Editor - &lt;oXygen/&gt;</title>
            <menutitle>Home</menutitle>
            <description><product/> XML Editor is a cross platform XML Editor providing tc
                authoring, XML conversion, XSL, XSLT, XQuery, XML Schema, DTD, Relax NG ar
                Schematron development, SOAP and WSDL.</description>
            <slide>
```

mihai 7285 (18 Lines)
mihai 7448 (1 Line)
mihai 7285 (126 Lines)
mihai 7325 (6 Lines)
mihai 7285 (36 Lines)
mihai 7448 (1 Line)
mihai 7285 (1 Line)
mihai 7325 (1 Line)
mihai 7418 (1 Line)
mihai 7285 (68 Lines)
mihai 7418 (1 Line)
mihai 7285 (9 Lines)
mihai 7321 (1 Line)
mihai 7285 (4 Lines)
mihai 7321 (4 Lines)
mihai 7285 (3 Lines)
mihai 7321 (1 Line)
mihai 7285 (4 Lines)
mihai 7321 (5 Lines)
mihai 7285 (5 Lines)
mihai 7321 (1 Line)
mihai 7285 (6 Lines)
mihai 7321 (1 Line)
mihai 7285 (7 Lines)
mihai 7321 (1 Line)
mihai 7285 (10 Lines)
mihai 7321 (1 Line)
mihai 7285 (6 Lines)
mihai 7321 (1 Line)
mihai 7285 (7 Lines)
mihai 7321 (1 Line)
mihai 7285 (7 Lines)

Figure 44: The Annotations View

The annotations of a file are computed with the **Show Annotation...** action, which is available in the **History** menu, and from the contextual menu of the following views: *the **Repositories** view*, ***Working copy** view*, ***History** view*, and ***Directory Change Set** view*.

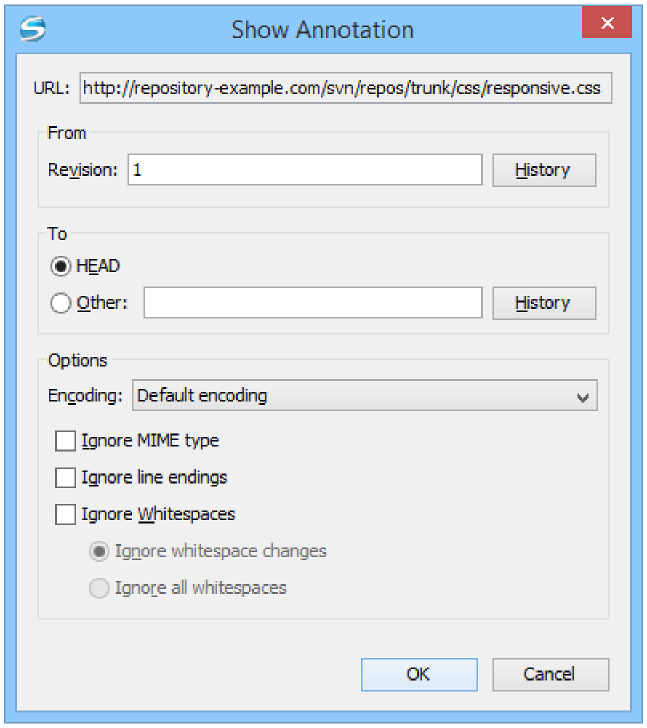


Figure 45: The Show Annotation Options Dialog Box

The following options can be configured in the **Show Annotation** dialog box:

From Revision Section

Select the revision from which to start computing the annotation. If you press the **History** button, *the **History** dialog box* is displayed, which allows you to select a revision.

To Revision Section

Select the ending revision by choosing between the **HEAD** revision or specify it in the **Other** text box . If you press the **History** button, *the **History** dialog box* is displayed, which allows you to select a revision.

Encoding

Select the encoding to be used when the annotation is computed. For each line of text, the SVN Client looks through the history of the file to be annotated see when it was last modified, and by whom. It is required that it is in the form of a text file. Therefore, encoding is needed to properly decode and read the file content. By default, the encoding of the operating system is used.

Ignore MIME type

If enabled, the file is treated as a text file and ignores what the SVN system infers from the `svn:mime-type` property.

Ignore line endings

If enabled, the differences in line endings are ignored when the annotation is computed.

Ignore whitespaces

If enabled, it allows you to specify how the whitespace changes should be handled. When enabled, you can then choose between two options:

- **Ignore whitespace changes** - Ignores changes in the amount of whitespaces or to their type (for example, when changing the indentation or changing tabs to spaces).

  Note: Whitespaces that were added where there were none before, or that were removed, are still considered to be changes.

- **Ignore all whitespaces** - Ignores all types of whitespace changes.

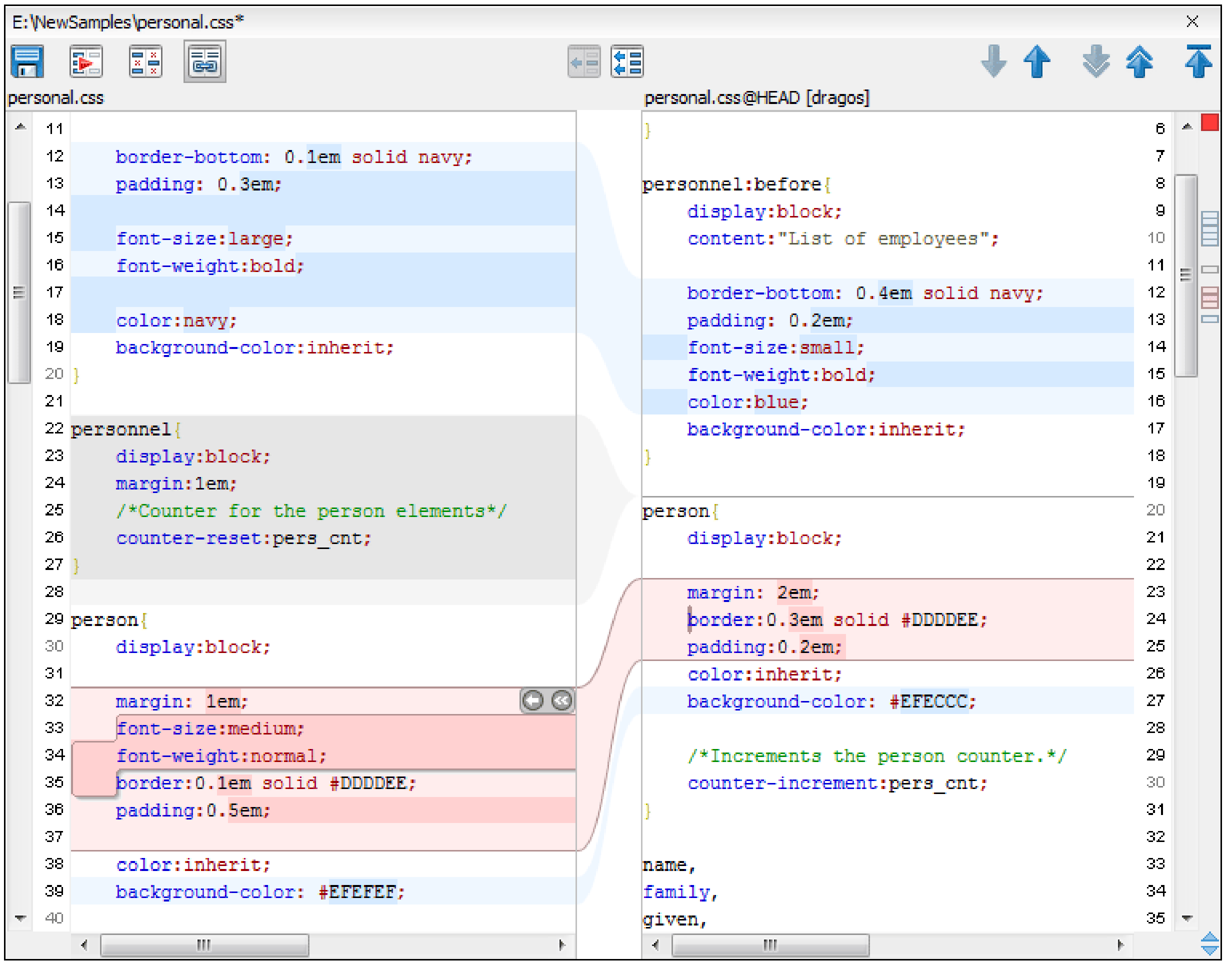
Tip: Enabling any of these *ignore* options can help you better determine the last time a meaningful change was made to a given line of text.

After you configure the options and press **OK**, the annotations will be computed and the **Annotations** view is displayed, where all the users that modified the selected resource will be presented, along with the specific lines and revision numbers modified by each user.

Note: If the file has a very long history, the computation of the annotation data can take a long time to process.

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.



```
    border-bottom: 0.1em solid navy;
    padding: 0.3em;

    font-size:large;
    font-weight:bold;

    color:navy;
    background-color:inherit;
}

personnel{
    display:block;
    margin:1em;
    /*Counter for the person elements*/
    counter-reset:pers_cnt;
}

person{
    display:block;

    margin: 1em;
    font-size:medium;
    font-weight:normal;
    border:0.1em solid #DDDDEE;
    padding:0.5em;

    color:inherit;
    background-color: #EFEFEF;
```

```
}

personnel:before{
    display:block;
    content:"List of employees";

    border-bottom: 0.4em solid navy;
    padding: 0.2em;
    font-size:small;
    font-weight:bold;
    color:blue;
    background-color:inherit;
}

person{
    display:block;

    margin: 2em;
    border:0.3em solid #DDDDEE;
    padding:0.2em;
    color:inherit;
    background-color: #EFECCC;

    /*Increments the person counter.*/
    counter-increment:pers_cnt;
}

name,
family,
given,
```

Figure 46: 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*.

Compare View Toolbar

The list of actions available in the **Compare** view toolbar include:

Save action

Saves the content of the left editor when it can be edited.

Perform Files Differencing

Performs a comparison between the source file and target file.

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 so that a selected difference can be seen on both sides of the application window. This action enables/disables the previously described behavior.

Format and Indent Both Files (Ctrl+Shift+P (Command+Shift+P on OS X))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

Next Block of Changes (Ctrl+Period (Command+Period on OS X))

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.

Note: A change block groups one or more consecutive lines that contain at least one change.

Copy Change from Right to Left

Copies the selected difference from the target file in the right side to the source file in the left side.

Copy All Changes from Right to Left

Copies all changes from the target file in the right side to the source file in the left side.

Previous Block of Changes (Ctrl+Comma (Command+Comma on OS X))

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.

Next Change (Ctrl+Shift+Period (Command+Shift+Period on OS X))

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 or when there are no changes.

Previous Change (Ctrl+Shift+Comma (Command+Shift+Comma on OS X))

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 or when there are no changes.

First Change (Ctrl+B (Command+B on OS X))

Jumps to the first change.

Most of these actions are also available 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).

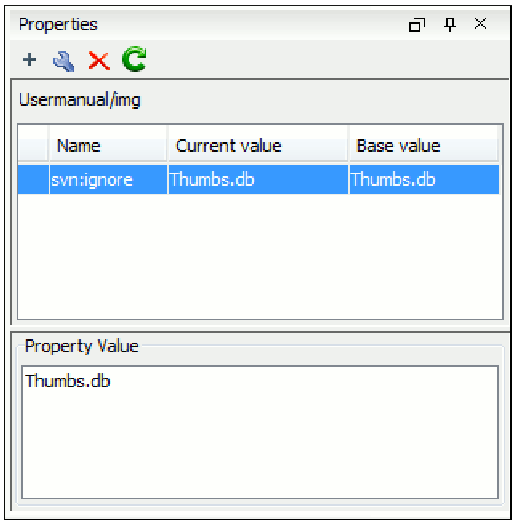
Compare Images View

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™ properties for the currently selected resource from either the **Working Copy** view or the **Repositories** view.



Usermanual/img

	Name	Current value	Base value
	svn:ignore	Thumbs.db	Thumbs.db

Property Value

Thumbs.db

Figure 47: 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.

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™ 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 that is specific 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*.

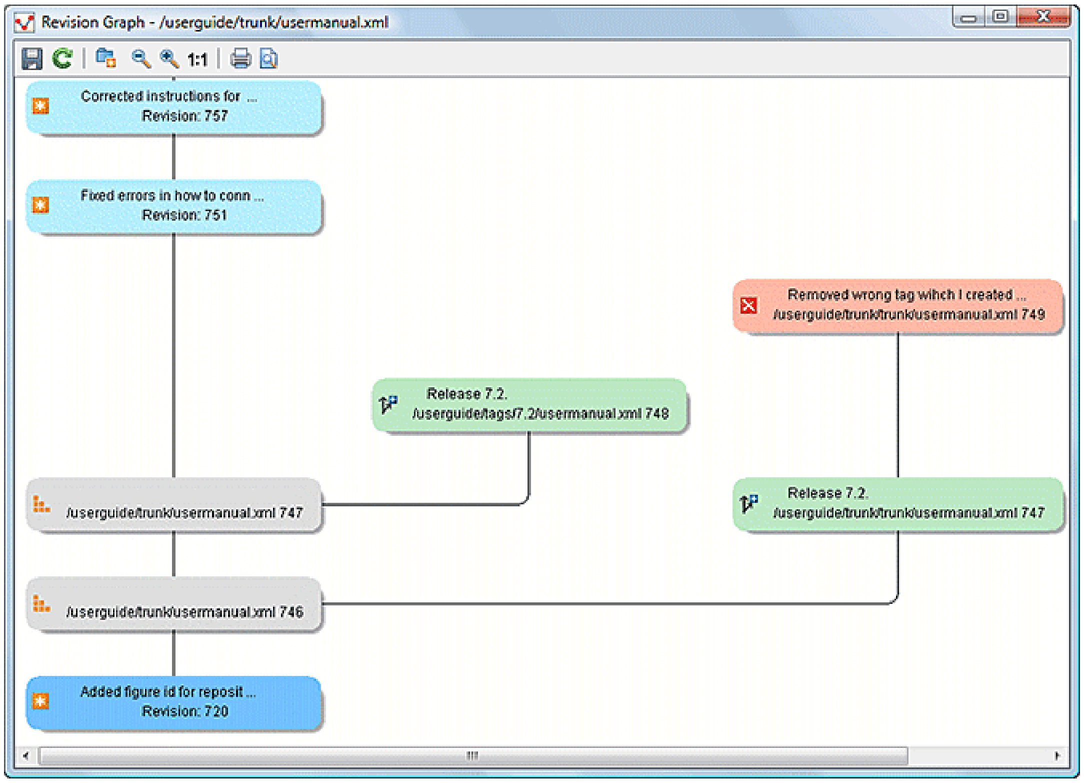

Revision Graph - /userguide/trunk/usermanual.xml
1:1
Corrected instructions for ... Revision: 757
Fixed errors in how to conn ... Revision: 751
Removed wrong tag wihch I created ... /userguide/trunk/trunk/usermanual.xml 749
Release 7.2. /userguide/tags/7.2/usermanual.xml 748
/userguide/trunk/usermanual.xml 747
Release 7.2. /userguide/trunk/trunk/usermanual.xml 747
/userguide/trunk/usermanual.xml 746
Added figure id for reposit ... Revision: 720

Figure 48: 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.

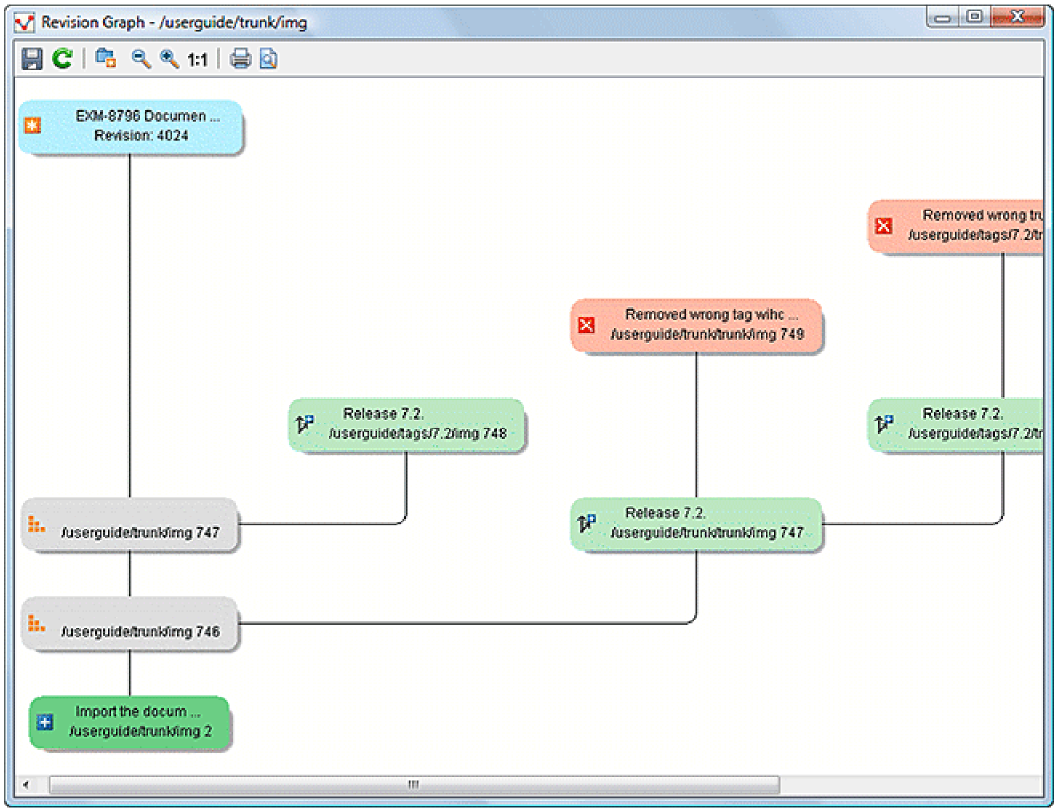


Figure 49: The Revision Graph of a Directory (Direct Changes)

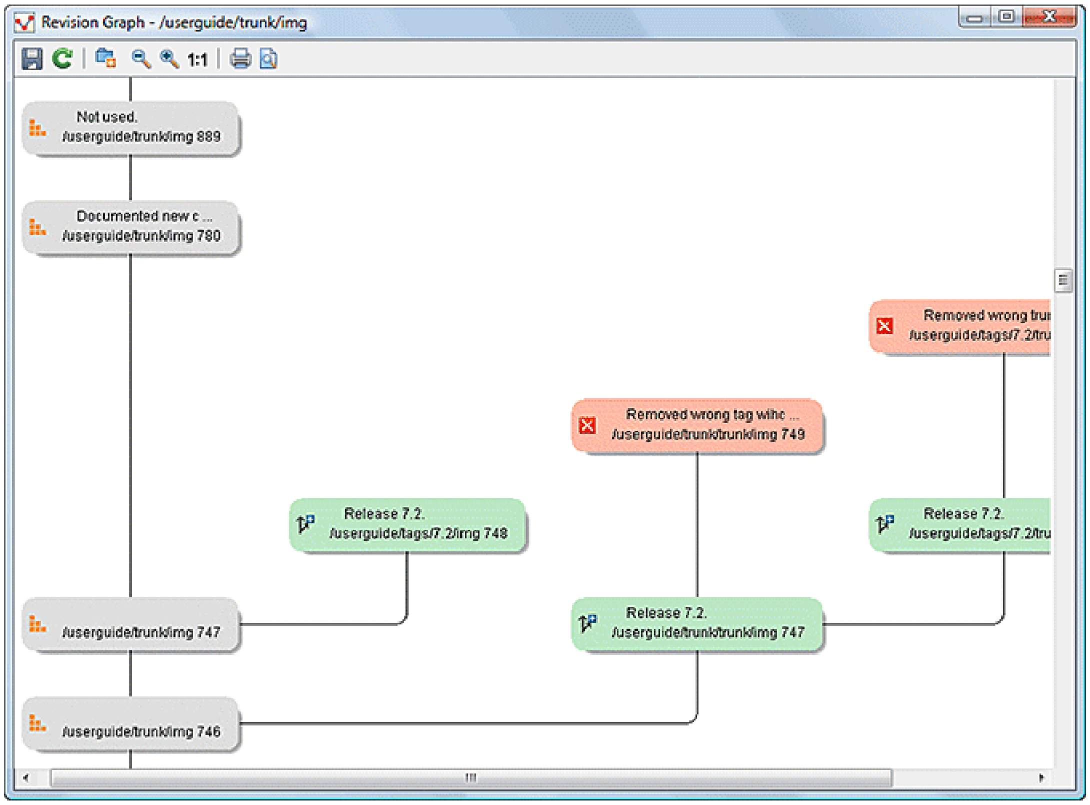


Figure 50: The Revision Graph of a Directory (Also Indirect Changes)

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.

The contextual menu of any of the graph nodes contains 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.

Export...

Opens *the* ***Export*** *dialog box* that allows you to configure options for exporting a folder from the repository to the local file system.

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.

Syncro SVN Client SVN Preferences

The options used in the SVN client are saved and loaded independently from the Syncro SVN Client options. However, if Syncro SVN Client cannot determine a set of SVN options to be loaded at startup, some of the preferences are imported from the XML Editor options (such as the License key and HTTP Proxy settings).

There is also an additional set of preferences applied to the SVN client that are set in global SVN files. There are two editing actions available in the **Global Runtime Configuration** submenu of the **Options** menu. These actions, **Edit 'config' file** and **Edit 'servers' file**, contain parameters that act as defaults applied to all the SVN client tools that are used by the same user on their login account.

Entering Local Paths and URLs

The Syncro SVN Client includes a variety of option configuration pages or wizards that contain text boxes where you specify paths to local resources or URLs of items inside remote repositories. The Syncro SVN Client provides support in these text boxes to make it easier to specify these paths and URLs.

Local Item Paths

The text boxes used for specifying local item paths support the following:

- *Absolute Paths* - In most cases, the Syncro SVN Client expects absolute paths for local file system items.
- *Relative Paths* - The Syncro SVN Client only accepts relative paths in the form `~[/...]`, where ~ is the user home directory.
- *Path Validation* - Syncro SVN Client validates the path as you type and invalid text becomes red.

- *Drag and Drop* - You can drag files and folders from the file system or other applications and drop them into the text box.
- *Automatic Use of Clipboard Data* - If the text box is empty when its dialog box is opened, any data that is available in the system clipboard is used as long as it is valid for that text box.

Repository Item URLs

- *Local Repository Paths* - You can use local paths (absolute or relative) to access local repositories. When you use the **Browse** button, the Syncro SVN Client will convert the file path to a `file://` form of URL as long as the location is a real repository.
  - *Absolute Paths* - In most cases, the Syncro SVN Client expects absolute paths for local file system items.
  - *Relative Paths* - The Syncro SVN Client only accepts relative paths in the form `~[/...]`, where ~ is the user home directory.
- *Peg Revisions* - For URL text boxes found inside dialog boxes where you are pulling information from the repository, you can *use peg revisions at the end of the URLs* (for example, `URL@rev1234`).

  Note: If you try to use a *peg revision* number in a dialog box where you are sending information to the repository then the peg revision number will become part of the name of the item rather than searching for the specified revision. For example, in the URL `http://host/path/inside/repo/item@100`, the item name is considered to be `item@100`.

  Tip: You can even use *peg revisions* with local repository paths. For example, `C:\path\to\local\repo@100` will be converted to `file:///C:/path/to/local/repo@100` and the **Repository browser** will display the content of the local repository as it is at revision `100`.
- *URL Validation* - Syncro SVN Client validates the URLs as you type and invalid text becomes red. Even paths to local repositories are not accepted unless using the **Browse** button to convert them to valid URLs.
- *Drag and Drop* - You can drag URLs from other applications or text editors and drop them into the URL text box. You can also drag folders that point to local repositories, from the local file system or from other applications, and they are automatically converted to valid `file://` type URLs.
- *Automatic Use of Clipboard Data* - If the URL text box is empty when its dialog box is opened, any data that is available in the system clipboard is used as long as it is valid for that text box. Even valid local paths will be automatically converted to `file://` type URLs.

Note: The text boxes that are in the form of a combo box also allow you to select previously used URLs, or URLs defined in the **Repositories** view.

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: `[HOME_DIR]\AppData\Roaming\Subversion`
- Mac OS X and Linux: `[HOME_DIR]/.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.

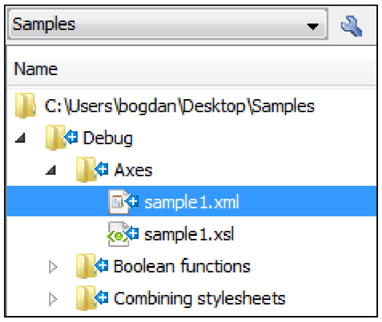


Figure 51: An incoming structure of folders from the repository

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).

Accessing Old Items from a Repository

Usually, you point to an item from a repository using a URL. However, sometimes this might not be enough, because the URL alone might point to a different item than the one you want and a *peg revision* is needed.

A Subversion repository tracks any change made to its items by using *revisions*, which contain information like the name of the author who made the changes, the date when they were made, and a number that uniquely identifies each of them. During time, an item from a specific location in a repository evolves as a result of committing different changes to it. When an item is deleted, its entire life cycle (all changes made to it, from the moment it was created) still remains recorded in the history of the repository. If a new item is created, with the same name and in the same location of the repository as a previously existing one, then both items are identified by the same URL even though they are located in different time frames. This is when a *peg revision* comes in handy. A *peg revision* is nothing more than a normal revision, but the difference between them is made by their usage. Many of the Subversion commands accept also a peg revision as a way to precisely identify an item in time, beside an *operative revision* (the revision we are interested in, regarding the used command).

Let's assume that:

- we created a new repository file `config`, identified by the URL `http://host.com/myRepository/dir/config`.
- the file has been created at revision `10`.
- during time, the file was modified by committing revisions `12, 15, 17`.

To access a specific version of the file identified by the `http://host.com/myRepository/dir/config` URL, we need to use a corresponding revision (the operative revision):

- if we use a revision number less than `10`, an error is triggered, because the file has not been created yet.
- if we use a revision number between `10` and `19`, we will obtain the specific version we are interested in.

    Note: Although the file was modified in revisions `12, 15, 17`, it existed in all revisions between `10` and `19`. Starting with a revision at which the file is modified, it has the same content across all revisions generated in the repository until another revision in which it is modified again.

At this point, we delete the file, creating revision `20`. Now, we cannot access any version of the file, because it does not exist anymore in the latest repository revision. This is due to the fact that Subversion automatically uses the `HEAD` revision as a peg revision (it assumes any item currently exists in the repository if not instructed otherwise). However, using any of the revision numbers from the `10-19` interval (when the file existed) as a peg revision (beside the operative revision), will help Subversion to properly identify the time frame when the file existed, and access the file version corresponding to the operative revision. If we use a revision number greater than `19`, this will also trigger an error.

Continuing our example, let's suppose that at revision `30` we create a directory, incidentally called `config`, in the same repository location as our deleted file. This means that our new directory will be identified by the same repository address: `http://host.com/myRepository/dir/config`. If we use only this URL in any Subversion command, we will access the new directory. We will also access the same directory if we use as peg revision any revision number equal with or greater than `30`. But, if we are interested in accessing an old version of the previously existing file, then we must use as a peg revision one of the revisions at which it existed (`10-19`), just like in the previous case.

Checksum Mismatch Error

A *Checksum Mismatch* error could happen if an operation that sends or retrieves information from the repository to the working copy is interrupted. This means that there is a problem with the synchronization between a local item and its corresponding remote item.

If you encounter this error, try the following:

1. Identify the parent directory of the file that caused the error (the file name should be displayed in the error message).

    Note: If the parent directory is the root of the working copy or if it contains a large amount of items it is recommended that you check out the working copy again, rather than continuing with the rest of this procedure.

2. Identify the *current depth* of that directory.
3. Update the parent directory using the **Update to revision/depth** action that is available from the contextual menu or the **Working copy** menu.

    a. For the **Depth** option, select **This folder only (empty)**.

        Warning: If you have files with changes in this directory, those changes could be lost. You should commit your changes or move the files to another directory outside the working copy prior to proceeding with this operation.

4. After clicking **OK** the contents of the directory will be erased and the directory is be marked as having *an empty depth*.
5. Once again, update the same directory using the **Update to revision/depth** action.

    a. This time, for the **Depth** option, select the depth that was previously identified in step 2.

6. If you moved modified files to another directory outside the working copy, move them back to the original location inside the working copy.

If this procedure does not solve the error, you need to check out the working copy again and move possible changes from the old working copy to the new one.

Comparing and Merging Documents

In large teams composed of developers or technical writers, the use of a shared repository for the source or document files is a must. Often times multiple authors may be editing the same file at the same time. It is easy to manage multiple changes by using the Syncro SVN Client comparison and merging tools.

It can be difficult to recognize which files and folders have been modified. If your data has changed, you can use the helpful Syncro SVN Client features for comparing files and directories to accurately identify and process changes in your files and folders. These tools are powerful, easy-to-use, and produce fast, thorough results.

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 types of comparison tools: **Compare Directories** or **Compare Files**. These utilities are available as stand-alone applications and can be opened from the Syncro SVN Client installation folder (`diffDirs.exe` and `diffFiles.exe`). The Syncro SVN Client also integrates the file comparison tool with its **Compare** view.

Note: File comparison and merging actions can also be performed on files inside ZIP-based archives.

The **Compare Files** tool can also be used to compare XML fragments. They can be compared and merged by copying and pasting the fragments into both sides of the comparison window, without selecting files.

The comparison tools can also be started by 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 OS X). To specify files or directories to compare, you can pass command-line arguments to each of these shells. The arguments can point to file or folder paths in directories or archives (supported formats: *zip*, *docx*, and *xlsx*).

For example, to start the comparison between the two Windows directories `c:\Program Files` and `c:\ant`, use the following command:

```
diffDirs.bat "c:\Program Files" "c:\ant"
```

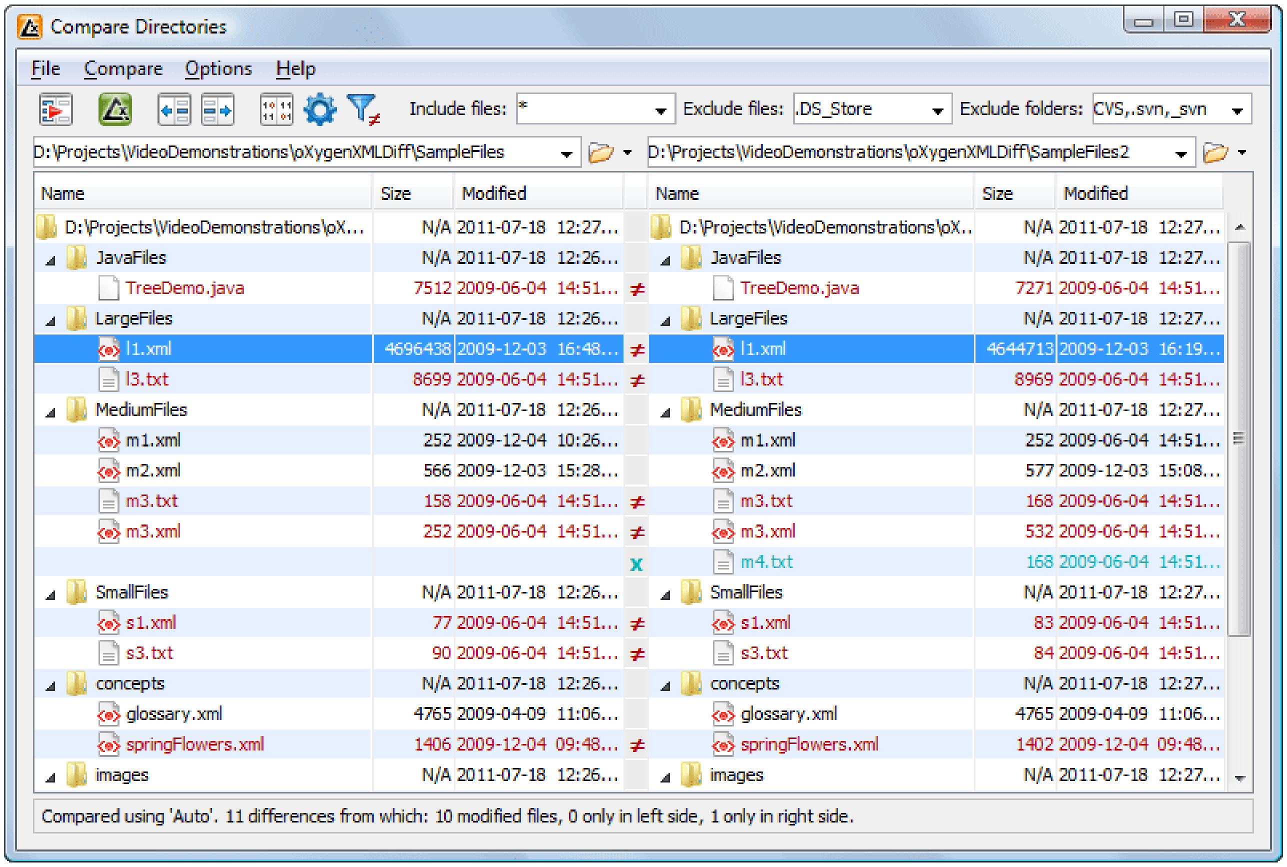
If there are spaces in the path names, surround the paths with quotes. If you pass only one argument, you are prompted to manually choose the second directory or archive. This is also true for the files comparison utility.

Directories Comparison

The **Compare Directories** tool allows you to compare and manage changes to files and folders within the structure of your directories.

The directories comparison results are presented as a tree of files and directories. The directories and folders that contain files that differ are expanded automatically so that you can focus directly on the differences. You can merge the contents of the directories by using the copy actions. If you double-click (or press **Enter**) on a line with a pair of files, Syncro SVN Client starts a *file comparison* between the two files, using the **Compare Files** tool.

Note: The comparison is only available for file type associations that are known by Syncro SVN Client.



Include files: *
Exclude files: .DS_Store
Exclude folders: CVS,.svn,_svn

D:\Projects\VideoDemonstrations\oXygenXMLDiff\SampleFiles
D:\Projects\VideoDemonstrations\oXygenXMLDiff\SampleFiles2

Name	Size	Modified		Name	Size	Modified
D:\Projects\VideoDemonstrations\oX...	N/A	2011-07-18 12:27...		D:\Projects\VideoDemonstrations\oX..	N/A	2011-07-18 12:27...
JavaFiles	N/A	2011-07-18 12:26...		JavaFiles	N/A	2011-07-18 12:27...
TreeDemo.java	7512	2009-06-04 14:51...	≠	TreeDemo.java	7271	2009-06-04 14:51...
LargeFiles	N/A	2011-07-18 12:26...		LargeFiles	N/A	2011-07-18 12:27...
l1.xml	4696438	2009-12-03 16:48...	≠	l1.xml	4644713	2009-12-03 16:19...
l3.txt	8699	2009-06-04 14:51...	≠	l3.txt	8969	2009-06-04 14:51...
MediumFiles	N/A	2011-07-18 12:26...		MediumFiles	N/A	2011-07-18 12:27...
m1.xml	252	2009-12-04 10:26...		m1.xml	252	2009-06-04 14:51...
m2.xml	566	2009-12-03 15:28...		m2.xml	577	2009-12-03 15:08...
m3.txt	158	2009-06-04 14:51...	≠	m3.txt	168	2009-06-04 14:51...
m3.xml	252	2009-06-04 14:51...	≠	m3.xml	532	2009-06-04 14:51...
			x	m4.txt	168	2009-06-04 14:51...
SmallFiles	N/A	2011-07-18 12:26...		SmallFiles	N/A	2011-07-18 12:27...
s1.xml	77	2009-06-04 14:51...	≠	s1.xml	83	2009-06-04 14:51...
s3.txt	90	2009-06-04 14:51...	≠	s3.txt	84	2009-06-04 14:51...
concepts	N/A	2011-07-18 12:26...		concepts	N/A	2011-07-18 12:27...
glossary.xml	4765	2009-04-09 11:06...		glossary.xml	4765	2009-04-09 11:06...
springFlowers.xml	1406	2009-12-04 09:48...	≠	springFlowers.xml	1402	2009-12-04 09:48...
images	N/A	2011-07-18 12:26...		images	N/A	2011-07-18 12:27...

Compared using 'Auto'. 11 differences from which: 10 modified files, 0 only in left side, 1 only in right side.

Figure 52: 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 actions:

- **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).

Options Menu

- **Preferences** - Opens the preferences.
- **Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page where you can configure keyboard shortcuts available for menu items.
- **Reset Global Options** - Resets options to their default values. Note that this option appears only when the tool is executed as a stand-alone application.
- **Import Global Options** - Allows you to import an options set that 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 actions:

Help (F1)

Opens the **Help** dialog.

Use online help (Enabled by default)

If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Syncro SVN Client attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

Report problem...

Opens a dialog that allows the user to write the description of a problem that was encountered while using the application. You are able to change the URL where the reported problem is sent by using the *com.oxygenxml.report.problems.url* system property. The report is sent in XML format through the `report` parameter of the POST HTTP method.

Support Center

Opens the Syncro SVN Client Support Center web page in a browser.

Compare Toolbar

The toolbar contains the following actions:

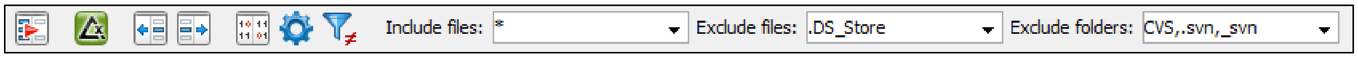

Include files: *
Exclude files: .DS_Store
Exclude folders: CVS,.svn,_svn

Figure 53: 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 directory comparison preferences page.

Show Only Modifications

Displays a more uncluttered file structure by hiding all identical files.

File and folder filters

Differences can be filtered using three combo boxes: **Include files**, **Exclude files**, and **Exclude folders**. They come with predefined values and are editable to allow 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.

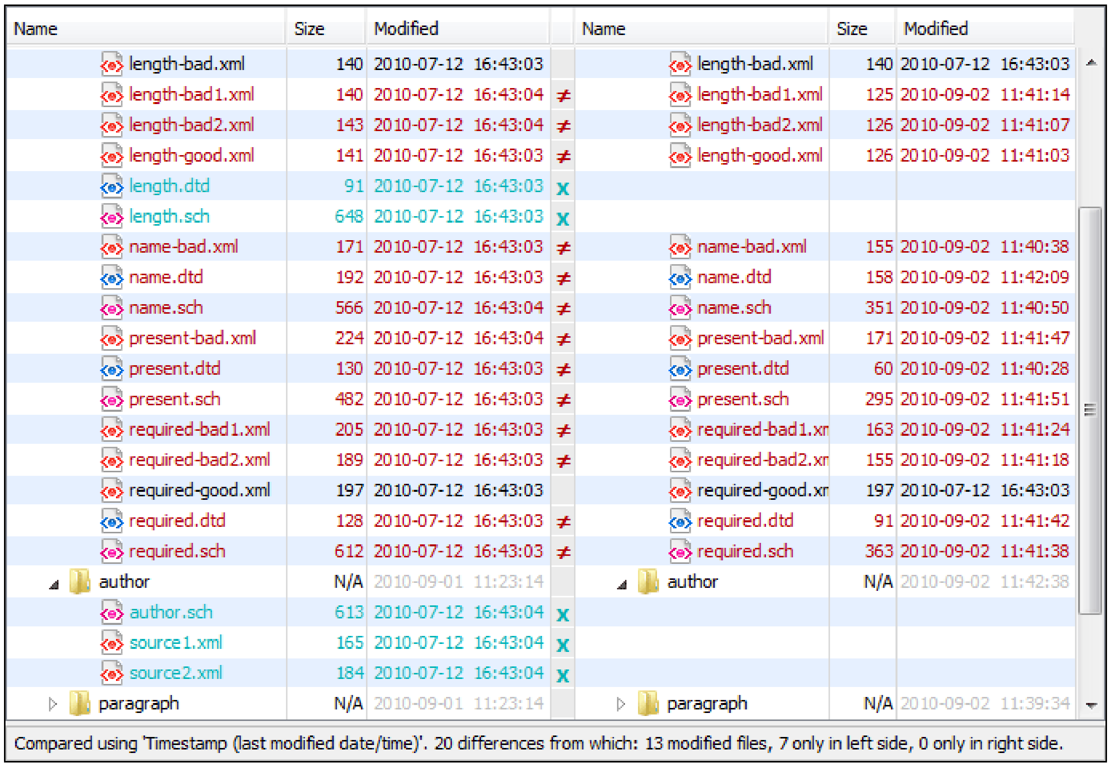
Directories Selector

To open the directories you want to compare, select a folder from each **Browse for local directory** button. The **Compare Directories** tool keeps track of the folders you are currently working with and those that you opened in this window. You can see and select them from the two drop-down lists.

If you want to compare the content of two archives, you can select the archives from the **Browse for archive file** button.

Comparison Results

The directory comparison results are presented using two tree-like structures showing the files and folders, including their name, 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.xr	163	2010-09-02 11:41:24
required-bad2.xml	189	2010-07-12 16:43:03	≠	required-bad2.xr	155	2010-09-02 11:41:18
required-good.xml	197	2010-07-12 16:43:03		required-good.xr	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 54: Comparison Results

A column that contains graphic symbols separates the two tree-like structures. The graphic symbols can be one of the following:

- An **X** symbol, when a file or a folder exists in only one of the compared directories.
- A symbol, when a file exists in both directories but the content differs. The same sign appears when a collapsed folder contains differing files.

Compare Images

If you double-click a line containing two different images, the **Compare images** window is displayed. This dialog presents the images in the left and right sides, scaled to fit the available view area. You can use the contextual menu actions to scale the images to their original size or scale them down to fit in the view area.

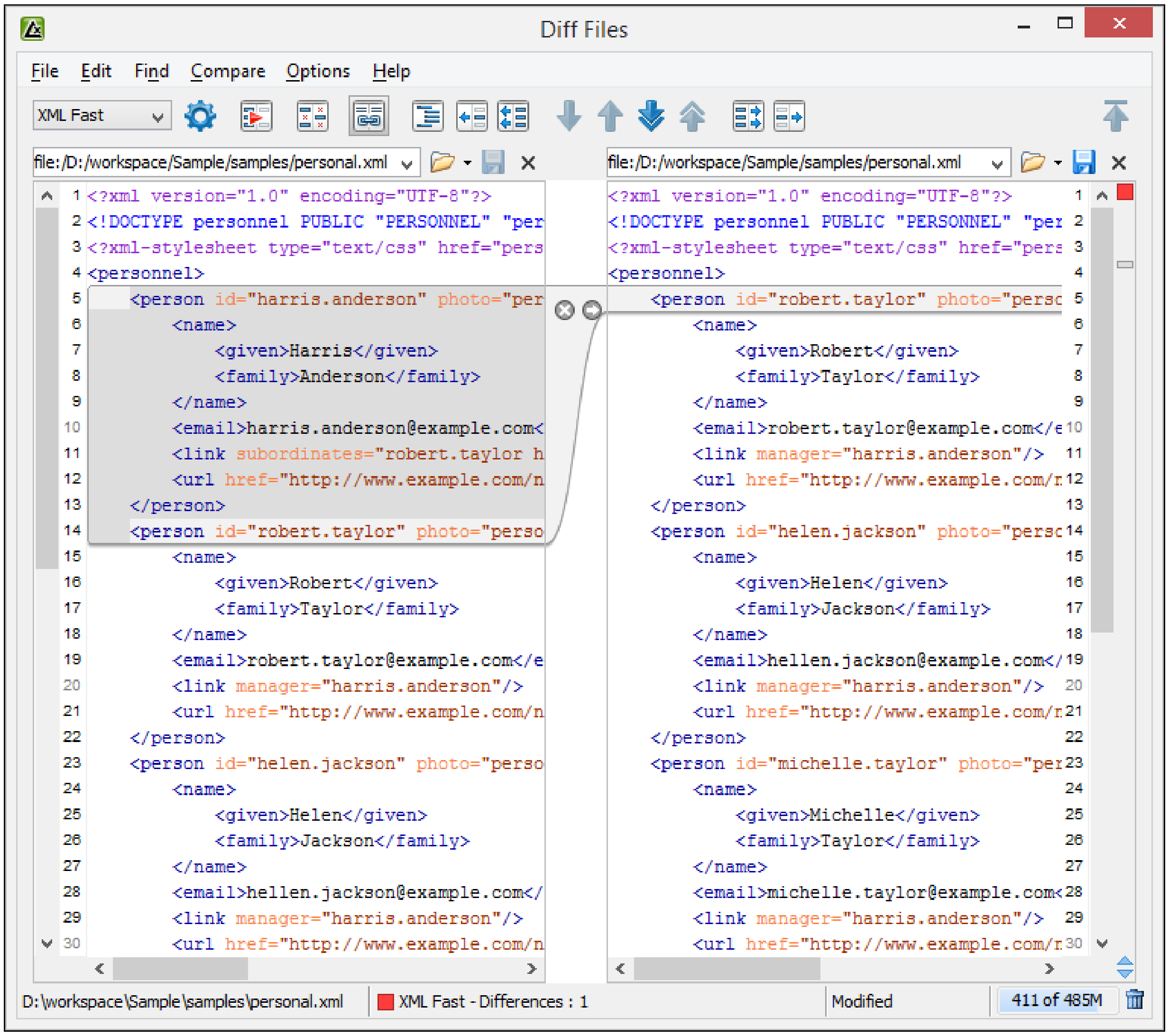
The supported image types are: *GIF*, *JPG*, *JPEG*, *PNG*, and *BMP*.

Files Comparison

The **Compare Files** tool can be used to compare files or XML file fragments. To compare two files side by side, open the **Diff Files** dialog from **Tools** > **Compare Files**. Using the browse drop-down list, 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 quickly identify the locations of the differences.

To compare XML file fragments, you need to copy and paste the fragments you want to compare into each side, without selecting a file. If a file is already selected, you need to close it, using the **Close (Ctrl+W (Command+W on OS X))** button, before pasting the fragments.

You can edit both the source and the target file. The differences are refreshed when you save the modified document.



file:/D:/workspace/Sample/samples/personal.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE personnel PUBLIC "PERSONNEL" "per
<?xml-stylesheet type="text/css" href="pers
<personnel>
    <person id="harris.anderson" photo="per
        <name>
            <given>Harris</given>
            <family>Anderson</family>
        </name>
        <email>harris.anderson@example.com<
        <link subordinates="robert.taylor h
        <url href="http://www.example.com/n
    </person>
    <person id="robert.taylor" photo="perso
        <name>
            <given>Robert</given>
            <family>Taylor</family>
        </name>
        <email>robert.taylor@example.com</e
        <link manager="harris.anderson"/>
        <url href="http://www.example.com/n
    </person>
    <person id="helen.jackson" photo="perso
        <name>
            <given>Helen</given>
            <family>Jackson</family>
        </name>
        <email>hellen.jackson@example.com</
        <link manager="harris.anderson"/>
        <url href="http://www.example.com/n
```

file:/D:/workspace/Sample/samples/personal.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE personnel PUBLIC "PERSONNEL" "per
<?xml-stylesheet type="text/css" href="pers
<personnel>
    <person id="robert.taylor" photo="persc
        <name>
            <given>Robert</given>
            <family>Taylor</family>
        </name>
        <email>robert.taylor@example.com</e
        <link manager="harris.anderson"/>
        <url href="http://www.example.com/r
    </person>
    <person id="helen.jackson" photo="persc
        <name>
            <given>Helen</given>
            <family>Jackson</family>
        </name>
        <email>hellen.jackson@example.com</
        <link manager="harris.anderson"/>
        <url href="http://www.example.com/r
    </person>
    <person id="michelle.taylor" photo="per
        <name>
            <given>Michelle</given>
            <family>Taylor</family>
        </name>
        <email>michelle.taylor@example.com<
        <link manager="harris.anderson"/>
        <url href="http://www.example.com/r
```

D:\workspace\Sample\samples\personal.xml | XML Fast - Differences : 1 | Modified | 411 of 485M

Figure 55: The Compare Files Window

Adjacent changes are grouped into blocks of changes. This layout allows you to easily identify and focus on a group of related changes.

When you select a change, a widget containing actions that can be used to copy or append changes from either of the two sides is displayed:

Append left change to right and Append right change to left

Copies the content of the selected change from one side and appends it on the other, after the content of the corresponding change. As a result, the side that the arrows point to will contain the changes from both sides.

Copy change from left to right and Copy change from right to left

Replaces 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 **File** menu of the files comparison tool contain the following actions:

Source

The file that is displayed in the left side of the application window.

- **Source** > **Open** - Browses for a source file.
- **Source** > **Open URL** - Opens the 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 that is displayed in the right side of the application window.

- **Target** > **Open** - Browses for a target file.
- **Target** > **Open URL** - Opens the 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 in the **Edit** menu:

Cut

Cut the selection from the currently focused **Compare** editor to the clipboard.

Copy

Copy the selection from the currently focused **Compare** editor to the clipboard.

Paste

Paste content from the clipboard into the currently focused **Compare** editor.

Undo

Undo changes in the currently focused **Compare** editor.

Redo

Redo changes in the currently focused **Compare** editor.

Find Menu

The **Find** menu actions are as follows:

Find/Replace

Perform *find/replace* operations in the currently focused **Editor**.

Find Next

Go to the next match using the same options as the last *find* operation. This action runs in both editor panels.

Find Previous

Go to the previous match using the same options as the last *find* operation. This action runs in both editor panels.

Compare Menu

The following actions are available in the **Compare** menu:

Perform Files Differencing

Performs a comparison between the source file and target file.

Next Block of Changes (Ctrl+Period (Command+Period on OS X))

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.

Note: A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes (Ctrl+Comma (Command+Comma on OS X))

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.

Next Change (Ctrl+Shift+Period (Command+Shift+Period on OS X))

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 or when there are no changes.

Previous Change (Ctrl+Shift+Comma (Command+Shift+Comma on OS X))

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 or when there are no changes.

Last Change (Ctrl+E (Command+E on OS X))

Jumps to the last change.

First Change (Ctrl+B (Command+B on OS X))

Jumps to the first change.

Copy All Changes from Right to Left

Copies all changes from the target file in the right side to the source file in the left side.

Copy All Changes from Left to Right

Copies all changes from the source file in the left side to the target file in the right side.

Copy Change from Right to Left

Copies the selected difference from the target file in the right side to the source file in the left side.

Copy Change from Left to Right

Copies the selected difference from the source file in the left side to the target file in the right side.

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.

Format and Indent Both Files (Ctrl+Shift+P (Command+Shift+P on OS X))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

Options Menu

- **Preferences** - Opens the preferences.
- **Menu Shortcut Keys** - Opens the **Menu Shortcut Keys** option page where you can configure keyboard shortcuts available for menu items.

- **Reset Global Options** - Resets options to their default values. Note that this option appears only when the tool is executed as a stand-alone application.
- **Import Global Options** - Allows you to import an options set that 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 actions:

Help (F1)

Opens the **Help** dialog.

Use online help (Enabled by default)

If this option is enabled, when you select **Help** or press **F1** while hovering over any part of the interface, Syncro SVN Client attempts to open the help documentation in online mode. If this option is disabled or an internet connection fails, the help documentation is opened in offline mode.

Report problem...

Opens a dialog that allows the user to write the description of a problem that was encountered while using the application. You are able to change the URL where the reported problem is sent by using the *com.oxygenxml.report.problems.url* system property. The report is sent in XML format through the `report` parameter of the POST HTTP method.

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 or XML fragments.

The following actions are available:

Perform Files Differencing

Performs a comparison between the source file and target file.

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 so that a selected difference can be seen on both sides of the application window. This action enables/disables the previously described behavior.

Format and Indent Both Files (Ctrl+Shift+P (Command+Shift+P on OS X))

Formats and indents both files before comparing them. Use this option for comparisons that contain long lines that make it difficult to spot differences.

Copy Change from Right to Left

Copies the selected difference from the target file in the right side to the source file in the left side.

Copy All Changes from Right to Left

Copies all changes from the target file in the right side to the source file in the left side.

Next Block of Changes (Ctrl+Period (Command+Period on OS X))

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.

Note: A change block groups one or more consecutive lines that contain at least one change.

Previous Block of Changes (Ctrl+Comma (Command+Comma on OS X))

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.

Next Change (Ctrl+Shift+Period (Command+Shift+Period on OS X))

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 or when there are no changes.

Previous Change (Ctrl+Shift+Comma (Command+Shift+Comma on OS X))

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 or when there are no changes.

Copy All Changes from Left to Right

Copies all changes from the source file in the left side to the target file in the right side.

Copy Change from Left to Right

Copies the selected difference from the source file in the left side to the target file in the right side.

First Change (Ctrl+B (Command+B on OS X))

Jumps to the first change.

Files Selector

To open the source and target files to use for comparison, use one of the following options from the browse drop-down list:

- **Browse for local** - Opens a dialog to browse for files in your local file system.
- **Browse for remote** - Opens the **Open URL** dialog to browse for remote files.
- **Browse for archive** - Opens the **Archive Browser** dialog to browse for archives.
- **Browse Data Source Explorer** - Opens the **Data Source Explorer** dialog to browse database sources.
- **Search for file** - Opens the **Find Resource** dialog for advanced search capabilities.

The comparison tool keeps track of the files you are currently working with and those that you opened in this window. You can see and select them from the two combo boxes.

You can also save the changes in either file by clicking the corresponding **Save** button.

You can close compared files by using the **Close** button. To compare XML fragments you need to close the compared files in order to copy and paste the fragments into each side of the panel.

File Contents Panel

Selected files are opened in two side-by-side editors. A text perspective is used to offer a better view of the differences. You can also compare XML fragments by copying and pasting them into both sides of the panel, without selecting files. To compare XML fragments, if files are already opened you need to close them in order to paste the fragments into the panels.

The two editors are constantly kept in sync. Therefore, if you scroll through the text in one side, the other one also scrolls to show the differences side-by-side. The differences are indicated with highlights connected through colored areas. To navigate through differences, do one of the following:

- Use the navigation buttons on the toolbar.
- Select the navigation options from the **Compare** menu.
- Select a block of differences by clicking its small colored indicator in the overview ruler located in the right-most part of the window. At the top of the overview ruler there is a success indicator that turns green where there are no differences, or red if differences are found.
- Click a colored area in between the two text editors.

You can edit the content in either side of the editor and the differences are refreshed when you save the modified document. If you save modified fragments, rather than a file, a dialog opens that allows you to save the changes as a new document.

Both editors provide a contextual menu that contains *edit*, *merge*, and *navigation* actions.

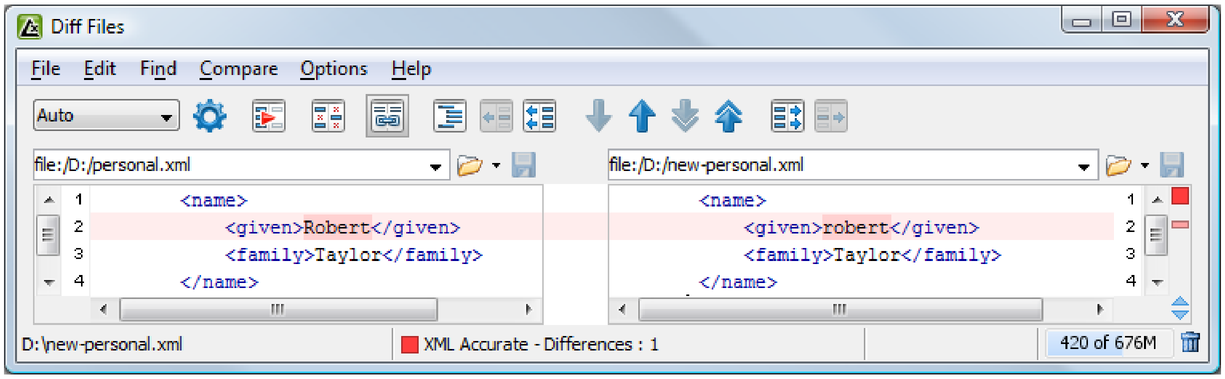
The **Find/Replace** dialog is displayed by pressing **Ctrl+F (Command+F on OS X)**. **Find/Replace** options are also available:

- **F3** - Performs a forward search, using the last search configuration.
- **Shift+F3** - Performs a backward search, using the last search configuration.

If the compared blocks of text are too large and you want to see the differences at a more fine-tuned level, you can choose options in the **Compare** menu to do a *word level comparison* or *character level comparison*.

Word Level Comparison

This option is only available if differences exist between the source and the target file. You can do a word level comparison by selecting the **Show word level details** option from the **Compare** menu.



file:/D:/personal.xml

```
<name>
    <given>Robert</given>
    <family>Taylor</family>
</name>
```

file:/D:/new-personal.xml

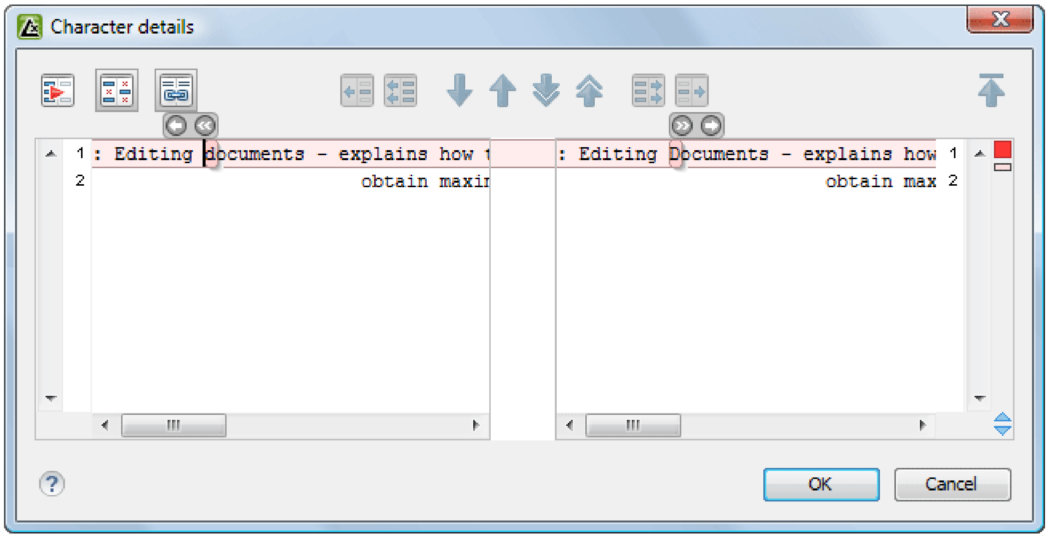
```
<name>
    <given>robert</given>
    <family>Taylor</family>
</name>
```

D:\new-personal.xml   XML Accurate - Differences : 1   420 of 676M

Figure 56: Word Level Comparison

Character Level Comparison

This option is only available if modifications exist between the source and target file. You can do a character level comparison by selecting the **Show Character Level details** option from the **Compare** menu.



```
: Editing documents - explains how t
                         obtain maxin
```

```
: Editing Documents - explains how
                         obtain max
```

Figure 57: Character Level Comparison

Chapter
4

Configuring Syncro SVN Client

Topics:

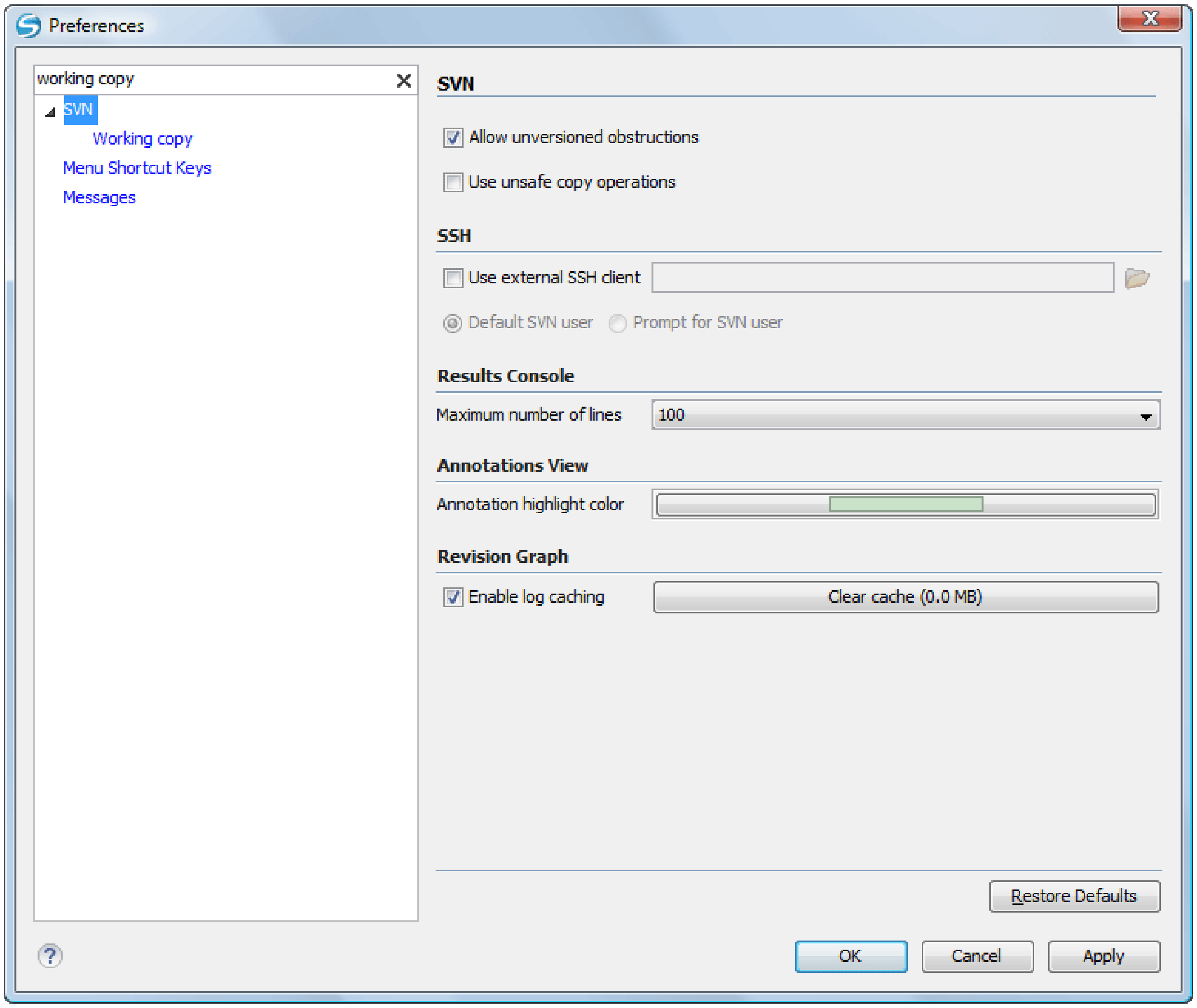
This chapter presents all the user preferences that allow you to configure the application.

Preferences

You can configure Syncro SVN Client options using the **Preferences** dialog.

To open the preferences dialog, go to **Options** > **Preferences**.

You can select the preference page you are interested in from the tree on the left of the **Preferences** dialog. You can filter the tree by typing in the filter box above the tree.



SVN

[✓] Allow unversioned obstructions

[ ] Use unsafe copy operations

SSH

[ ] Use external SSH client

[✓ Default SVN user] Prompt for SVN user

Results Console

Maximum number of lines: 100

Annotations View

Annotation highlight color: 🟩

Revision Graph

[✓] Enable log caching

Clear cache (0.0 MB)

Figure 58: The Search field from the Preferences dialog

You can restore options to their default values by pressing the **Restore Defaults** button, available in each preferences page.

Press ? or **F1** for help on any preferences page.

Global options and license information are stored in the following locations:

- [user-home-folder]\Application Data\com.oxygenxml for Windows XP
- [user-home-folder]\AppData\Roaming\com.oxygenxml for Windows Vista/7
- [user-home-folder]/Library/Preferences/com.oxygenxml for Mac OS X
- [user-home-folder]/.com.oxygenxml for Linux

Global Preferences

The global options cover a number of aspects of the overall operation of .To configure the **Global** options *open the **Preferences** dialog* and go to **Global**.

The following options are available in the **Global** preferences page:

- **Automatic Version Checking** - This option sets whether Syncro SVN Client will check for a new version on startup.
- **Language** - This option sets the language used in the user interface to English, French, German, Dutch, or Japanese. You must restart Syncro SVN Client for the change to take effect.
- **Other language** - This option sets the language used in the user interface using an interface localization file. You can use this option to set the language of the user interface to a language that is not shipped with Syncro SVN Client.

  Note: If some interface labels are not rendered correctly after restarting the application, (for example Chinese or Korean characters are not displayed correctly), make sure that your operating system has the appropriate language pack installed (for example the East-Asian language pack).

- **Line separator** - This option sets the line separator used when saving files. Use **System Default** to select the normal line separator for your OS. If you want the existing file separator of a file to be maintained, regardless of your current OS, check **Detect the line separator on file open**.
- **Detect the line separator on file open** - When this option is selected, the editor detects the line separator when a file is loaded and it uses it when the file is saved. New files are saved using the line separator defined by the**Line separator** option.
- **Show memory status** - When this option is selected, the memorySyncro SVN Client uses is displayed in the status bar. To free memory, click the **Free unused memory** button located at the right side of the status bar. The memory status bar turns yellow or red when Syncro SVN Client uses too much memory. You can change the amount of memory available to Oxygen by *changing the parameters of the application launcher*.
- **Show Java vendor warning at startup** - If this option is selected, Syncro SVN Client displays a warning on startup if a non-recommended version of the Java virtual machine is being used.
- **Show hidden files and directories** - If this option is selected, Syncro SVN Client shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on OS X.

Appearance Preferences

This preferences page contains a number of options that allow you to change the appearance of the user interface of Syncro SVN Client. To configure the **Appearance** options *open the **Preferences** dialog* and click on **Appearance**.

The following options are available in the **Appearance** preferences page:

- **Look and Feel** - This option allows you to change the graphic style (look and feel) of the user interface. Depending on the operating system, you can choose between different predefined style options.
- **Theme** - This option allows you to set a color theme that will be applied over the entire user interface. You can choose between predefined color themes, and you can also change various options within the selected theme. Links to pages for the various options are displayed in the **Appearance** preferences page. The result of the applied modifications is displayed in the **XML Preview** area.

  Note: In Windows, if the **Look and Feel** and **Theme** options are set to their default values, Syncro SVN Client inherits the high contrast theme colors that are set in the operating system.

Fonts Preferences

Syncro SVN Client lets you choose the fonts used in the **Text** editor mode. To configure the **Fonts** options, *open the **Preferences** dialog* and go to **Fonts**.

The following options are available:

- **Editor** - Sets the fonts used in the editor.

- **Note:** On Mac OS X, the default font, Monaco, cannot be rendered in bold.
- **Text antialiasing** - This option sets the 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** - This option sets the font used in text boxes in the interface. After changing the font, restart the application for the change to take full effect.
- **GUI** - This option sets the font used for user interface labels. After changing the font, restart the application for the change to take full effect.

Encoding Preferences

Syncro SVN Client lets you configure how character encodings are recognized when opening files and which encodings are used when saving files. To configure encoding options, *open the **Preferences** dialog* and go to **Encoding**. The following encoding options are available:

- **Fallback character encoding** - Default character encoding of non-XML documents if their character encoding cannot be determined from other sources (like, for example, specified in the document itself, or determined by the file type).

  Note: For certain document types, the following encoding detection rules are used:

  - For XML, DTD and CSS documents, Syncro SVN Client tries to collect the character encoding from the document. If no such encoding is found, then *UTF-8* is used.
  - For JavaScript, JSON, SQL, XQuery, and RNC, the *UTF-8* encoding is used.

- **Encoding errors handling** - This setting specifies how to handle characters that cannot be represented in the character encoding that is used when the document is opened. The available options are:
  - **REPORT** - displays an error identifying 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 error is ignored and the character is not included in the document displayed in the editor.

    Attention: If you edit and save the document, the characters that cannot be represented in the specified encoding are dropped.

  - **REPLACE** - the character is replaced 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

Syncro SVN Client lets you configure how the editor appears. To configure the appearance of the text editor, *open the **Preferences** dialog* and go to **Editor**.

The following options are available:

- **Selection background color** - Sets the background color of selected text.
- **Selection foreground color** - Sets the text color of selected text.
- **Editor background color** - Sets the background color for both text editor and Diff Files editors.
- **Editor caret color** - Sets the color of the caret.
- **Highlight current line** - Sets the foreground color of the line numbers displayed in the editor panels.
- **Show line numbers** - Shows line numbers 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 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

Syncro SVN Client lets you control how files are opened and saved. To configure the **Open / Save** options, *open the **Preferences** dialog* and go to **Editor** > **Open / Save**.

Open

The following options apply to opening files:

- **When bidirectional (BIDI) text is detected** - Allows you to choose what happens when you try to open a file that contains BIDI Unicode characters. You can choose one of the following:
  - **Enable bidirectional editing mode**
  - **Disable bidirectional editing mode**
  - **Prompt for each document**
- **Disable bidirectional text support for documents larger than (Characters)** - Enabling bidirectional text editing support can affect performance on large files. When this option is selected, bidirectional editing is turned off for files exceeding the specified size, even if the **When bidirectional (BIDI) text is detected** option is set to **Enable bidirectional editing mode**.
- **Show BIDI limit warning** - It this option is selected, Syncro SVN Client will warn you if bidirectional editing support is turned off and a file exceeds the specified limit.

Save Hooks Preferences

Syncro SVN Client includes an option for automatically compiling LESS stylesheets. To set this option, *open the **Preferences** dialog* and go to **Editor** > **Open / Save** > **Save hooks**.

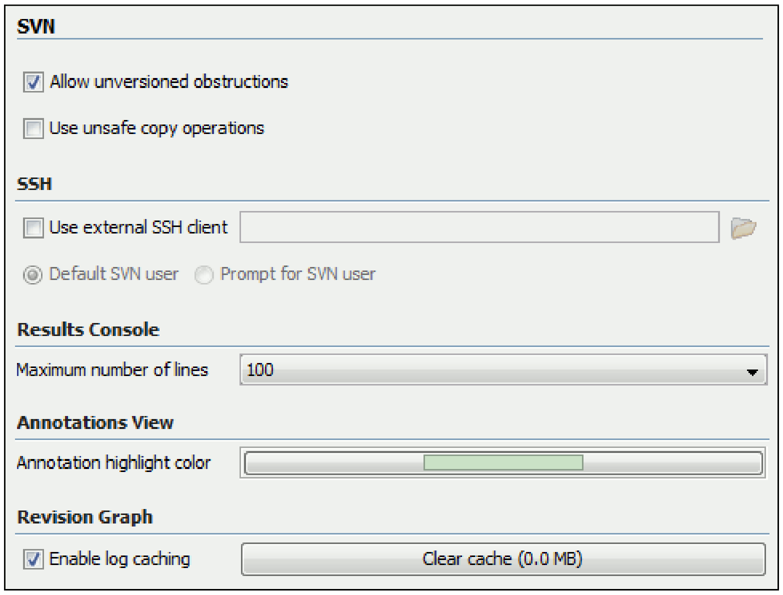
The following option can be enabled or disabled:

- **Automatically compile LESS to CSS when saving** - If enabled, when you save a LESS file it will automatically be compiled to CSS (disabled by default).

  Important: If this option is enabled, when you save a LESS file, the CSS file that has the same name as the LESS file is overwritten without warning. Make sure all your changes are made in the LESS file. Do not edit the CSS file directly, as your changes might be lost.

SVN Preferences

To configure the **SVN** options, *open the **Preferences** dialog* and go to **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.



SVN

- [✓] Allow unversioned obstructions
- [ ] Use unsafe copy operations

SSH

- [ ] Use external SSH client
- Default SVN user / Prompt for SVN user

Results Console

Maximum number of lines: 100

Annotations View

Annotation highlight color: 🟩

Revision Graph

- [✓] Enable log caching

Figure 59: The SVN Preferences Panel

The SVN preferences are the following:

- **Enable symbolic link support** (*available only on Mac OS X and Linux*) - Apache Subversion™ has the ability to put a symbolic link under version control, via the usual SVN `add` command. The Subversion repository has no internal concept of a symbolic link, it stores a versioned symbolic link as an ordinary file with a `svn:special` property attached. The SVN client (on Unix) sees the property and translates the file into a symbolic link in the working copy.

  Note: 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 reference the 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 configure the **Working Copy** preferences, *open the **Preferences** dialog* and go to **SVN** > **Working Copy**. The option that you are able to configure in this preferences page are specific to SVN working copies:

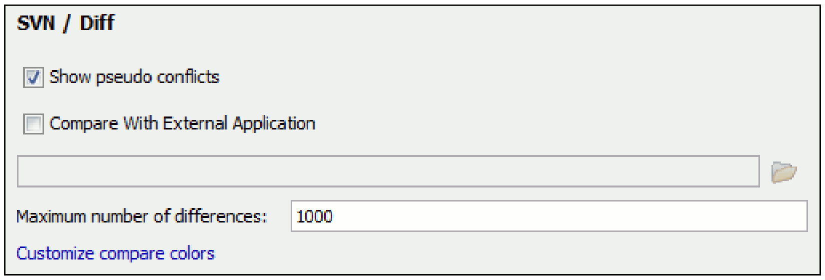
- **Working copies location** - Allows you to define a location where you keep your working copies. This location is automatically suggested when you checkout a new working copy.
- **Working copy administrative directory** - Allows you to customize the directory name where the svn entries are kept for 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:
  - **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.
  - **Never upgrade** - Older format working copies are left untouched. No attempt to upgrade the format is made.

  Note: SVN 1.6 and older working copies still need to be upgraded before loading them.

- **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.
- **Allow moving/renaming mixed revision directories** - If enabled, the Syncro SVN Client will allow you to move or rename a directory even if its child items have a different revision. Otherwise, an error message is displayed when there are multiple revisions to avoid unnecessary conflicts. It is recommended to leave this option disabled and to **Update** the subtree to a single revision before moving or renaming it.
- **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

To configure **Diff** options, *open the **Preferences** dialog* and go to **Diff**.


SVN / Diff

☑ Show pseudo conflicts

☐ Compare With External Application

Maximum number of differences: 1000

Customize compare colors

Figure 60: 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*.

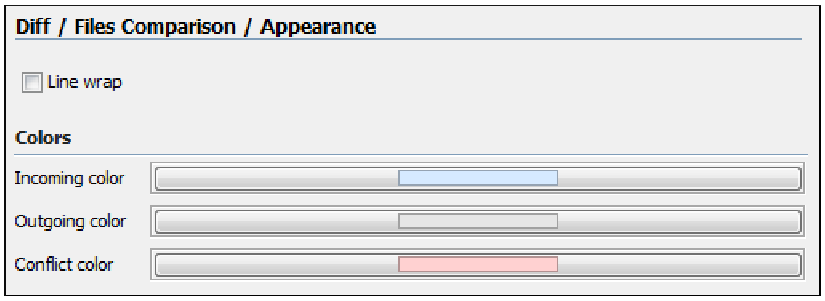
  The parameters ${firstFile} and ${secondFile} specify the positions of the two compared files in the command line for the external diff application. The parameter ${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison. The working copy version of 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.

  Important: If the path to the external compare application includes spaces (or any of the subsequent options or arguments), then each of these paths or *tokens* must be double-quoted for the Syncro SVN Client to correctly parse and identify them. For example, `C:\Program Files\compareDir\app name.exe` must be written as `"C:\Program Files\compareDir\app name.exe"`.

- **Maximum number of differences** - sets the maximum number of differences allowed in the view.

Appearance Preferences

To configure the **Files Comparison / Appearance** preferences, *open the **Preferences** dialog* and go to **Diff** > **Files Comparison** > **Appearance**. This preferences page offers the following options:


Diff / Files Comparison / Appearance

☐ Line wrap

Colors

Incoming color

Outgoing color

Conflict color

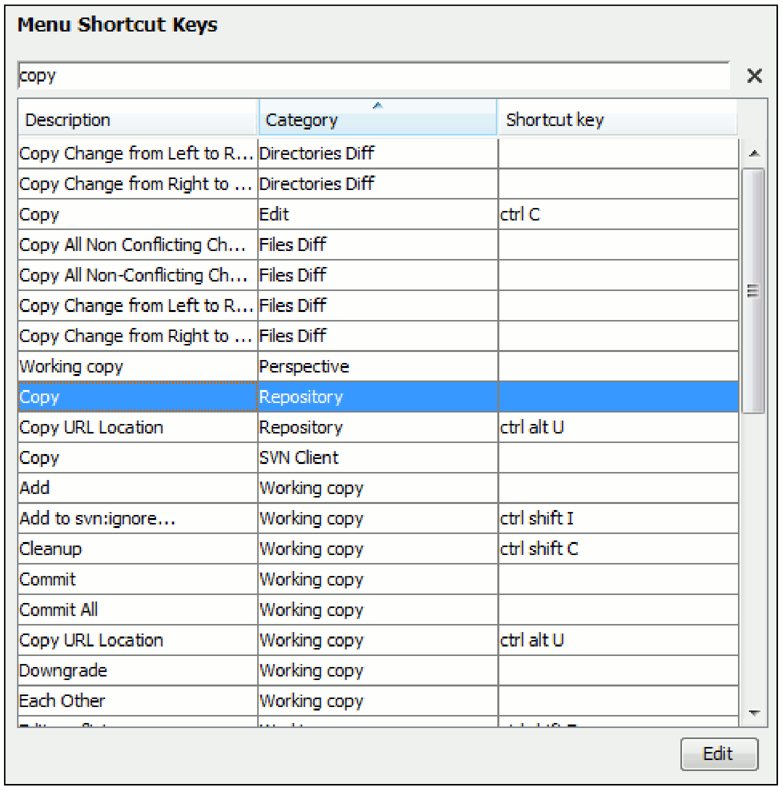
Figure 61: Files Comparison Appearance Preferences Panel

- **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 configure the **Menu Shortcut Keys** options, *open the **Preferences** dialog* and go to **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.



Description	Category	Shortcut key
Copy Change from Left to R…	Directories Diff	
Copy Change from Right to …	Directories Diff	
Copy	Edit	ctrl C
Copy All Non Conflicting Ch…	Files Diff	
Copy All Non-Conflicting Ch…	Files Diff	
Copy Change from Left to R…	Files Diff	
Copy Change from Right to …	Files Diff	
Working copy	Perspective	
Copy	Repository	
Copy URL Location	Repository	ctrl alt U
Copy	SVN Client	
Add	Working copy	
Add to svn:ignore…	Working copy	ctrl shift I
Cleanup	Working copy	ctrl shift C
Commit	Working copy	
Commit All	Working copy	
Copy URL Location	Working copy	ctrl alt U
Downgrade	Working copy	
Each Other	Working copy	

Figure 62: The Menu Shortcut Keys Preferences Panel

The **Menu shortcut Keys** preferences page also contains the shortcuts that you define at *document type* level.

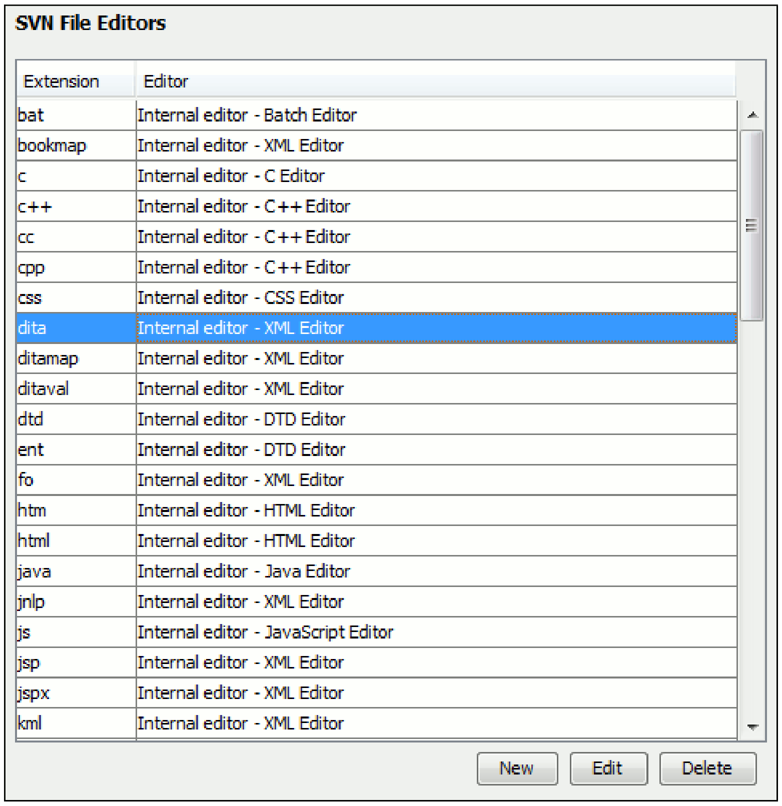
Note: A shortcut defined at *document type* level overwrites a default shortcut.

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. To configure SVN file formats, *open the **Preferences** dialog* and go to **SVN File Editors**.

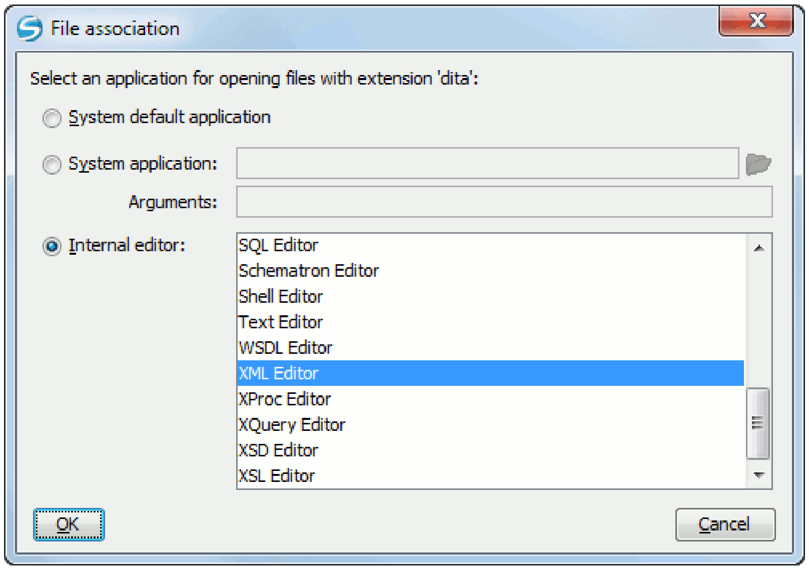


SVN File Editors

Extension	Editor
bat	Internal editor - Batch Editor
bookmap	Internal editor - XML Editor
c	Internal editor - C Editor
c++	Internal editor - C++ Editor
cc	Internal editor - C++ Editor
cpp	Internal editor - C++ Editor
css	Internal editor - CSS Editor
dita	Internal editor - XML Editor
ditamap	Internal editor - XML Editor
ditaval	Internal editor - XML Editor
dtd	Internal editor - DTD Editor
ent	Internal editor - DTD Editor
fo	Internal editor - XML Editor
htm	Internal editor - HTML Editor
html	Internal editor - HTML Editor
java	Internal editor - Java Editor
jnlp	Internal editor - XML Editor
js	Internal editor - JavaScript Editor
jsp	Internal editor - XML Editor
jspx	Internal editor - XML Editor
kml	Internal editor - XML Editor

Figure 63: 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.



File association

Select an application for opening files with extension 'dita':

System default application

System application:

Arguments:

[✓ Internal editor:]

SQL Editor
Schematron Editor
Shell Editor
Text Editor
WSDL Editor
XML Editor
XProc Editor
XQuery Editor
XSD Editor
XSL Editor

Figure 64: 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.

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 configure the **Proxy** options, *open the **Preferences** dialog* and go to **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. A host needs to be written as a fully qualified domain name (e.g. *myhost.example.com*) or as a domain name (e.g. *example.com*). Use comma as a separator between multiple hosts.
- **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)/WebDAV Preferences

To set the **HTTP(S)/WebDAV** preferences , *open the **Preferences** dialog* and go to **Network 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

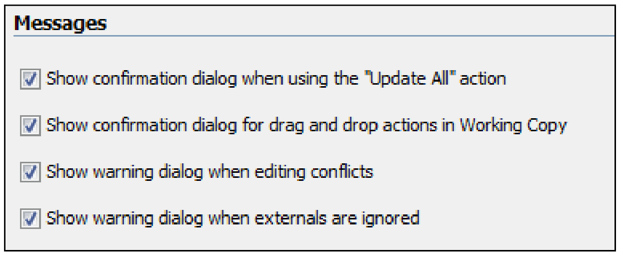
To configure the **SSH** options, *open the **Preferences** dialog* and go to **Connection settings** > **SSH**. The following options are available:

- **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 configure the **Messages** options, *open the **Preferences** dialog* and go to **Messages**. This preferences page allows you to disable the following warning messages which may appear in the application:


Messages

- [✓] Show confirmation dialog when using the "Update All" action
- [✓] Show confirmation dialog for drag and drop actions in Working Copy
- [✓] Show warning dialog when editing conflicts
- [✓] Show warning dialog when externals are ignored

Figure 65: 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*).

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 to the factory defaults.

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 OS X
- [user-home-folder]/. com.syncrosvnclient for Linux

The name of the `options` file of Syncro SVN Client10.1 is oxyOptionsSvn8.0.xml.

Reset Global Options

Options > **Reset Global Options**

Customizing Default Options

Syncro SVN Client has an extensive set of options that you can configure. When Syncro SVN Client in installed, these options are set to default values. You can provide a different set of default values for an installation using an *options file*.

Creating an *options file*

To create an *options file*:

1. Open Syncro SVN Client. You may wish to use a fresh install for this procedure, to make sure that you do not copy personal option settings to the group.
2. *Open the* ***Preferences*** *dialog* .
3. Go through the options and set them to the desired defaults. Make sure that you are setting global options, not project options in each page.
4. Close the **Preferences** dialog.
5. Go to **Options** > **Export Global Options** and create an options file.
6. Go to back to the main preferences page and click **Export Global Options** to create an options file.

Providing Default Option Values

Use either one of the following ways to configure an Syncro SVN Client installation to use customized default options from an XML configuration file:

- Set the path to the options file as the value of the `com.oxygenxml.default.options` .

  The path must be specified with an URL or a file path relative to the application installation folder:

  ```
  -Dcom.oxygenxml.default.options=options/default.xml
  ```

- In the [OXYGEN_DIR] installation folder, create a folder called `preferences`. Copy the options file in the .

Note: Make sure that the options configuration file has either the `.xml` extension (for example: `default-options.xml`) or an `.xpr` extension depending on the way in which it was created (from the global options or saved at project level).

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+Click (Command+Click on OS X)** (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: For a Mac kit bundled with Java 7, look for the **VMOptionArray** key and add the -Xmx parameter in a new string `element` from the `array` element. For example, for 1500 MB use the following:

  ```
  <string>-Xmx1500m</string>
  ```

  Tip: 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
```

You can also set a system property through a parameter prefixed with `-Doxy` in the command line used to start the application:

```
syncroSVNClient.exe "-Doxyproperty.name=value"
```

All system properties are displayed in the **System properties** tab of the **About** dialog.

Disabling DPI Scaling

Some users may prefer the look of smaller icons in a HiDPI display. To achieve this, display scaling needs to be disabled for high DPI settings. To disable the DPI scaling, set the following property in `.vmoptions` (or in the `.bat` script):

```
sun.java2d.dpiaware=false
```

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"\
 -Xmx600m\
 ...
```

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

```



java -Xmx600m\
```

Chapter 5

Common Problems

Topics:

This section lists the most commonly found problems and their solutions.

Performance Problems

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

Display Problems on Linux or 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.

Common Problems and Solutions

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_DIR]/lib/fonts` folder. `[JVM_DIR]` 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.

The *DITA to CHM* Transformation Fails

Syncro SVN Client uses the DITA Open Toolkit and the HTML Help compiler (part of the Microsoft HTML Help Workshop) to transform DITA content into *Compiled HTML Help* (or *CHM* in short).

However, the execution of the transformation scenario may still fail. Reported errors include:

- `[exec] HHC5010: Error: Cannot open "fileName.chm". Compilation stopped.` - this error occurs when the CHM output file is opened and the transformation scenario cannot rewrite its content. To solve this issue, close the CHM help file and execute the transformation scenario again.
- `[exec] HHC5003: Error: Compilation failed while compiling fileName` - possible causes of this error are:
  - the processed file does not exist. Fix the file reference before executing the transformation scenario again.
  - the processed file has a name that contains space characters. To solve the issue, remove any spacing from the file name and execute the transformation scenario again.

DITA Map ANT Transformation Because it Cannot Connect to External Location

The transformation is run as an external ANT process so you can continue using the application as the transformation unfolds. All output from the process appears in the **DITA Transformation** tab.

The HTTP proxy settings are used for the ANT transformation so if the transformation fails because it cannot connect to an external location you can check the

Topic References outside the main DITA Map folder

Referencing to a DITA topic, map or to a binary resource (for example: image) which is located outside of the folder where the main DITA Map is located usually leads to problems when publishing the content using the DITA Open Toolkit. The DITA OT does not handle well links to topics which are outside the directory where the published DITA Map is found. By default it does not even copy the referenced topics to the output directory.

You have the following options:

1. Create another DITA Map which is located in a folder path above all referenced folders and reference from it the original DITA Map. Then transform this DITA Map instead.
2. Edit the transformation scenario and in the **Parameters** tab edit the **fix.external.refs.com.oxygenxml** parameter. This parameter is used to specify whether the application tries to fix up such references in a temporary files folder before the DITA Open Toolkit is invoked on the fixed references. The fix up has no impact on your edited DITA content. Only "false" and "true" are valid values. The default value is false.

The PDF Processing Fails to Use the DITA OT and Apache FOP

There are cases when publishing DITA content fails when creating a PDF file. This topic lists some common problems and solutions.

- The FO processor cannot save the PDF at the specified target. The console output contains messages like:

  ```
  [fop] [ERROR] Anttask - Error rendering fo file: C:\samples\dita\temp\pdf\oxygen_dita_temp\topic.fo <Failed
  to open C:\samples\dita\out\pdf\test.pdf>
  Failed to open samples\dita\out\pdf\test.pdf
  .............
  [fop] Caused by: java.io.FileNotFoundException: C:\Users\radu_coravu\Desktop\bev\out\pdf\test.pdf
  (The process cannot access the file because it is being used by another process)
  ```

  Such an error message usually means that the PDF file is already opened in a PDF reader application. The solution is to close the open PDF before running the transformation.

- One of the DITA tables contains more cells in a table row than the defined number of *colspec* elements. The console output contains messages like:

  ```
  [fop] [ERROR] Anttask - Error rendering fo file:
  D:\projects\eXml\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo
  <net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException:
  The column-number or number of cells in the row overflows the number of fo:table-columns specified for the
  table. (See position 179:-1)>net.sf.saxon.trans.XPathException: org.apache.fop.fo.ValidationException: The
  column-number or number of cells in the row overflows the number of fo:table-columns specified for the table.
   (See position 179:-1)
  [fop]  at org.apache.fop.tools.anttasks.FOPTaskStarter.renderInputHandler(Fop.java:657)
  [fop]  at net.sf.saxon.event.ContentHandlerProxy.startContent(ContentHandlerProxy.java:375)
  ............
  [fop] D:\projects\samples\dita\flowers\temp\pdf\oxygen_dita_temp\topic.fo ->
  D:\projects\samples\dita\flowers\out\pdf\flowers.pdf
  ```

  To resolve this issue, correct the *colspec* attribute on the table that caused the issue. To locate the table that caused the issue:

  1. Edit the transformation scenario and set the parameter *clean.temp* to *no*.
  2. Run the transformation, open the `topic.fo` file in Syncro SVN Client, and look in it at the line specified in the error message `(See position 179:-1)`.
  3. Look around that line in the `XSL-FO` file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

- There is a broken link in the generated `XSL-FO` file. The PDF is generated but contains a link that is not working. The console output contains messages like:

  ```
  [fop] 1248 WARN [ main ] org.apache.fop.apps.FOUserAgent - Page 6: Unresolved ID reference
  "unique_4_Connect_42_wrongID" found.
  ```

  To resolve this issue:

  1. Use the **Validate and Check for Completeness** action available in the **DITA Maps Manager** view to find such problems.
  2. If you publish to PDF using a `DITAVAL` filter, select the same DITAVAL file in the **DITA Map Completeness Check** dialog.
  3. If the **Validate and Check for Completeness** action does not discover any issues, edit the transformation scenario and set the *clean.temp* parameter to *no*.
  4. Run the transformation, open the `topic.fo` file in Syncro SVN Client, and search in it for the `unique_4_Connect_42_wrongID` id.
  5. Look around that line in the `XSL-FO` file to find relevant text content which you can use, for example, with the **Find/Replace in Files** action in the **DITA Maps Manager** view to find the original DITA topic for which the table was generated.

The *TocJS* Transformation Doesn't Generate All Files for a Tree-Like TOC

The *TocJS* transformation of a DITA map does not generate all the files needed to display the tree-like table of contents. To get a complete working set of output files you should follow these steps:

1. Run the *XHTML* transformation on the same DITA map. Make sure the output gets generated in the same output folder as for the *TocJS* transformation.
2. Copy the content of `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/basefiles` folder in the transformation's output folder.
3. Copy the `[OXYGEN_DIR]/frameworks/dita/DITA-OT/plugins/com.sophos.tocjs/sample/basefiles/frameset.html` file in the transformation's output folder.
4. Edit `frameset.html` file.
5. Locate element `<frame name="contentwin" src="concepts/about.html">`.

6. Replace `"concepts/about.html"` with `"index.html"`.

Navigation to the web page was canceled when viewing CHM on a Network Drive

When viewing a CHM on a network drive, if you only see the TOC and an empty page displaying “Navigation to the web page was canceled” note that this is normal behaviour. The Microsoft viewer for CHM does not display the topics for a CHM opened on a network drive.

As a workaround, copy the CHM file on your local system and view it there.

Index