Syncro SVN Client 6.2

Contents

Chapter 1

Introduction

Welcome to the User Manual of Syncro SVN Client 6.2 which explains how to use the 6.2 version of the Syncro SVN Client effectively to access Subversion repositories and manage their local working copies. Please note that this manual assumes that you are familiar with the basic concepts of a version control system.

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

Chapter

2

Installation

Topics:

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

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

Installation Requirements

This section contains details about the platform and environment requirements necessary for installing and running the application.

Platform Requirements

The run-time requirements of the application are:

- CPU (processor): minimum - Intel Pentium III™/AMD Athlon™ class processor, 500 *Mhz*; recommended - Dual Core class processor.
- Computer memory: minimum - 256 MB of RAM (1 GB on Windows Vista and Windows 7); recommended - 512 MB of RAM (2 GB on Windows Vista and Windows 7).
- Hard disk space: minimum - 100 MB free disk space; recommended - 200 MB free disk space.

Operating System

Windows	Windows XP, Windows Vista, Windows 7, Windows 2003, Windows Server 2008
Mac OS	Mac OS X version 10.4 or later
Unix/Linux	Any Unix/Linux distribution with an available Java SE Runtime Environment version 1.5 or 1.6 from Oracle (formerly Sun).

Environment Requirements

This section specifies the Java platform requirements and other tools that may be needed for installing the application.

Tools

Installation packages are supplied in compressed archives. Ensure you have installed a suitable archive extraction utility with which to extract the archive. The MD5 sum is available on *the Download page* for every archive. You should check the MD5 sum of the downloaded archive with a MD5 checking tool available on your platform.

Java Virtual Machine Prerequisites

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

- Syncro SVN Client supports only official and stable Java virtual machines with the version number 1.5.0 or later from Oracle, formerly Sun Microsystems (available at *http://www.java.com/en/download/manual.jsp*) and from Apple Computer. The Java Virtual Machine from Apple is pre-installed on Mac OS X computers. For Mac OS X, Java Virtual Machine updates are available at *http://www.apple.com/macosx/features/java/*. Syncro SVN Client may work very well with JVM implementations from other vendors but the eventual incompatibilities will not be solved in further Syncro SVN Client releases. Syncro SVN Client *does not work with the GNU libgcj Java virtual machine*. It is recommended to use a Java Virtual Machine with version 1.6.0 or later for better performance.
- The PATH environment variable is set to the most current Java Virtual Machine installation.
- References to older Java Virtual Machine installations are removed from the PATH.

Installation Instructions

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

Note:

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

Note:

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

Windows Installation

Windows installation procedure.

To install the application on Windows:

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

   The user preferences are stored in the subfolder `com.syncrosvnclient` of the folder that is the value of the APPDATA Windows variable for the user that starts the application.

   Note:

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

Mac OS X Installation

Mac OS X installation procedure.

To install the application on Mac OS X:

1. Create a folder called `syncroSVNClient` on your local disk.
2. Within the `syncroSVNClient` folder, create child folder named in accordance with the version number of the application.

   The folder structure looks as follows: /../syncroSVNClient/6.2/
3. Download the Mac OS X Installation package (`syncroSVNClient.tar.gz`) into this folder.
4. Extract the archive into the same folder.
5. Execute the file named `syncroSVNClient`

   Note:

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

Linux Installation

Linux installation procedure.

To install the application on Linux:

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

Note:

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

All Platforms Installation

All Platforms kit installation procedure.

1. Create a folder called `syncroSVNClient` on your local disk.
2. Within the `syncroSVNClient` folder, create child folder named in accordance with the application version number.
   The directory structure looks as follows: /../syncroSVNClient/6.2/
3. Download the All Platforms Installation package (`syncroSVNClient.tar.gz`) to this folder.
4. Extract the archive to the same folder.
5. Run from a command line the script `syncroSVNClient.bat` on Windows, `syncroSVNClientMac.sh` on Mac OS X, `syncroSVNClient.sh` on Unix / Linux.

Note:

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

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

on Windows,

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

on Mac OS X.

Unattended Installation

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

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

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

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

Custom Settings in Unattended Installation

By default an unattended installation applies the default settings of the installer. If you want to install the application on a large number of computers but you need to change the default values of some settings (like the install folder on disk, whether a desktop icon or a quick launch shortcut is created, the file associations created in the operating system, the name of the program group on the Start menu, etc.) then you should use a special settings file which specifies the new values for these settings. To generate the settings file you have to run the installer in normal attended mode once on a test computer and specify the exact options that you want for the unattended installation. When the installation is completed a file called `response.varfile` and containing your selected options is created in the `.install4j` subfolder of the installation folder, by default C:\Program Files\Syncro SVN Client 6.2\.install4j on Windows. This is a one time process. After that for applying these options on all the computers where an unattended installation is performed you have to specify this file in the command line, for example copy the file in the same location as the installer program and use the command:

- on Windows:

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

- on Linux:

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

Setting a Parameter in the Startup Script

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

```
-Xmx600m
```

Note: On Windows Vista/7 you will first have to copy the `syncroSVNClient.vmoptions` file to a folder with write access (like your Desktop), modify it there with a text editing application (like *Notepad*) and then copy it back to the installation folder, replacing the original file.

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

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

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

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

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

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

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

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

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

```



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

Start the Application

This section specifies the steps for starting the application.

Starting the Application on Windows

Start the application launcher.

Use one of the following two launchers:

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

Starting the Application on Mac OS X

Start the application's launcher.

Use one of the following two methods:

- The shortcut `syncroSVNClient` created on **Desktop** by the installer.
- The command

  ```
  sh syncroSVNClientMac.sh
  ```

  executed from command line. This launcher file is located in the install folder.

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

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

Starting the Application on Linux

Start the application's launcher.

Use one of the following two methods:

- The shortcut `syncroSVNClient` created on **Desktop** by the installer.

- The command

  ```
  sh syncroSVNClient.sh
  ```

  executed from command line. This launcher file is located in the install folder.

Starting the Application with the All Platforms Kit

Start the application's launcher.
Use the following command:

- On Windows:

  ```
  syncroSVNClient.bat
  ```

- On Linux:

  ```
  sh syncroSVNClient.sh
  ```

- On Mac OS X:

  ```
  sh syncroSVNClientMac.sh
  ```

Obtaining and Registering a License Key

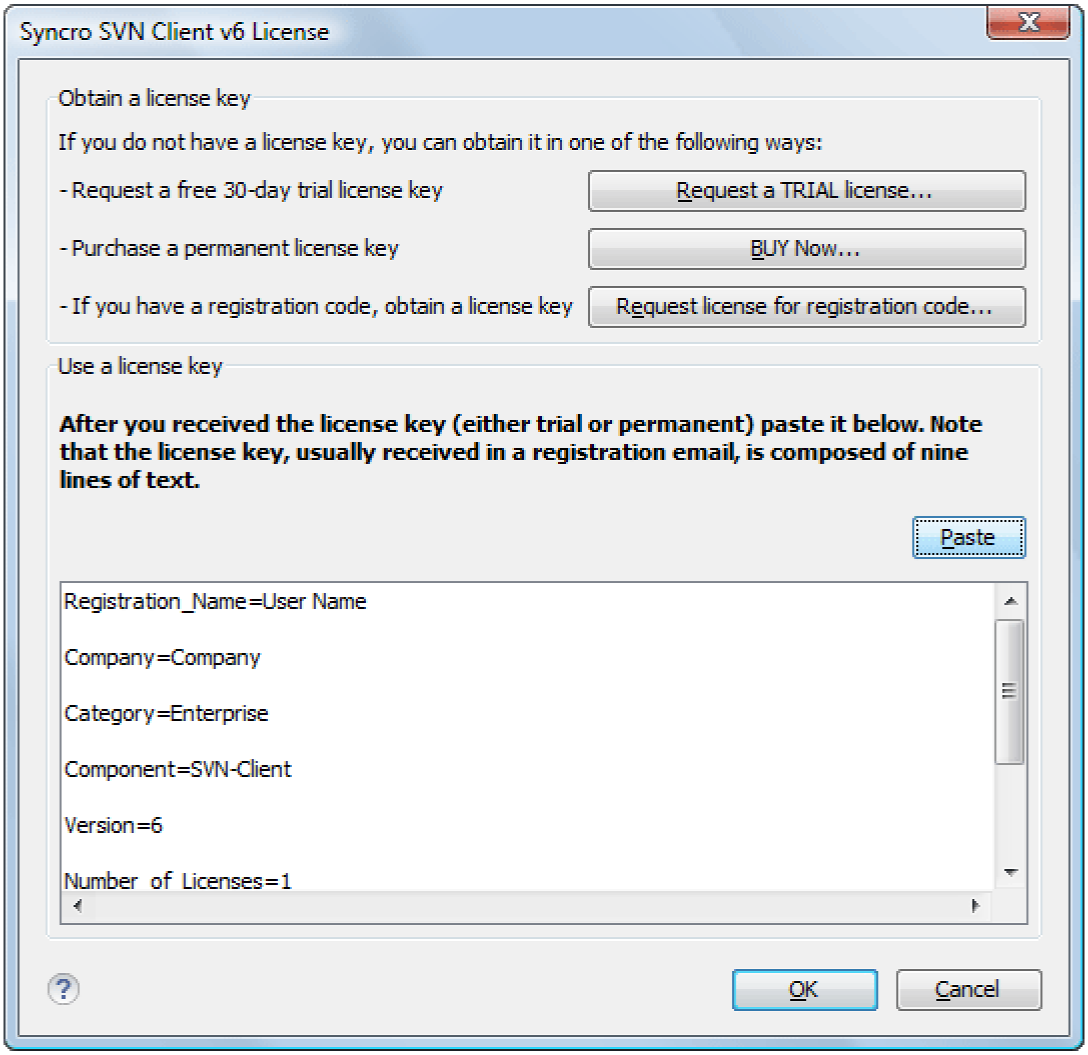
The Syncro SVN Client is not free software and requires a license in order to enable the application.

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

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

Named User License Registration

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


Syncro SVN Client v6 License
Obtain a license key
Request a TRIAL license...
BUY Now...
Request license for registration code...
Use a license key
Paste
?
OK
Cancel

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
- Purchase a permanent license key
- If you have a registration code, obtain a license key

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.

```
Registration_Name=User Name

Company=Company

Category=Enterprise

Component=SVN-Client

Version=6

Number  of  Licenses=1
```

Figure 1: Registration Dialog

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

Named User License Registration with Text File

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

License Registration with an Activation Code

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

Unregistering the License Key

Sometimes you need to unregister your license key, for example to transfer your license key to other computer before other user starts using your current computer.

1. Go to menu **Register**
   This displays the license registration dialog.
2. Make sure the text area for the license key is empty.
3. Make sure the checkbox **Use a license server** is unchecked.
4. Press the **OK** button of the dialog.
   This displays a confirmation dialog.
5. Select between falling back to the license key entered previously (for the case of releasing a floating license and reverting to Named User license) and removing your license key from your user account of the computer.

Upgrading the Syncro SVN Client Application

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

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

Upgrading the Standalone Application

Upgrading to a new version might require a new license key. To check if your license key is compatible with the new version, select **Help** > **Check for New Version** . Note that the application needs an Internet connection to check the license compatibility.

👉 **Note:** If there is a previous version of Syncro SVN Client already installed on your computer, it can coexist with the new one, which means you don't have to uninstall it. The user preferences are stored in a different directory. They will not be removed and will be imported automatically in the new version at the first application launch.

1. Download the install kit of the new version.
2. Create a new folder under `/../syncrosvnClient` e. g. /../syncrosvnClient/6.2
3. Extract the content of the install kit into the new folder.
4. If you have defined Syncro SVN Client in the system PATH, modify it to point to the new installation folder.
5. Start Syncro SVN Client
   This will test that the application can start and that your license is recognized by the upgrade installation.

Checking for New Versions

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

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

Uninstalling the Application

This section contains uninstallation procedures.

Uninstalling the Standalone Application

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

1. Backup all valuable data from the Syncro SVN Client installation folder.
2. Remove the application.
   - On Windows use the appropriate uninstaller shortcut provided with your OS.
   - On Mac OS X and Unix manually delete the installation folder and all its contents.
3. If you want to remove also the user preferences that were configured in the **Preferences** dialog you must remove the folder `%APPDATA%\com.syncrosvnClient` on Windows (usually %APPDATA% has the value [user-home-dir]\Application Data) / the subfolder `.com.syncrosvnClient` of the user home folder on Linux / the subfolder `Library/Preferences/com.syncrosvnClient` of the user home folder on Mac OS X.

Unattended Uninstall

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

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

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

Performance Problems

This section contains the solutions for some common problems that may appear when running the application.

Display Problems on Linux/Solaris

Display problems like screen freeze or momentary menu pop-ups during mouse movements over screen on Linux or Solaris can be solved by *adding the startup parameter* -Dsun.java2d.pmoffscreen=false.

Chapter 3

Syncro SVN Client

Topics:

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

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

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

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

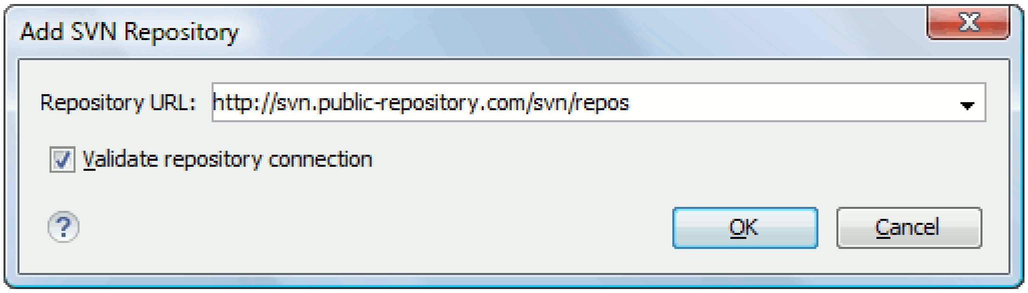
  - **Open (Ctrl + O)** - This action opens the selected file in an editor where you can modify it. The action is active only when a single item is selected. The action opens a file with the internal editor or the external application associated with that file type. In case of a folder the action opens the selected folder with the system application for folders (for example Windows Explorer on Windows, Finder on Mac OS X, etc). Folder opening is available only for folders selected in the ***Working Copy*** *view*. This action works on any file selection from the ***Repositories*** *view*, ***Working Copy*** *view*, ***History*** *view* or ***Directory Change Set*** *view*, depending on which view was last focused when invoking it.
  - **Open with ...(Ctrl + Shift + O)** - Displays the ***Open with*** dialog for specifying the editor in which the selected file is opened. In case multiple files are selected only external applications can be used to open the files. This action works on any file selection from ***Repositories*** *view*, ***Working Copy*** *view*, ***History*** *view* or ***Directory Change Set*** *view*, depending on which view was last focused when invoking it.
  - **Save (Ctrl + S)** - Saves the local file currently opened in the editor or the **Compare** view.
  - **Copy URL Location (Ctrl + Alt + U)** - Copies to clipboard the URL location of the resource currently selected in the **Repositories** view.
  - **Copy/Move to (Ctrl + M)** - Copies or moves to a specified location the resource currently selected either in **Repositories** or **Working copy** view.
  - **Rename (F2)** - Renames the resource currently selected either in **Repositories** or **Working copy** view.
  - **Delete (Delete)** - Deletes the resource currently selected either in **Repositories** or **Working copy** view.
  - **Show SVN Properties (Ctrl + Shift + P)** - Brings up the ***Properties*** *view* and displays the SVN properties for a selected resource from ***Repositories*** *view* or ***Working Copy*** *view*, depending on which view was last focused when invoking it.
  - **File Information (Ctrl + I)** - Provides additional information for a selected resource from the ***Working Copy*** *view*. For more details please see the section *Obtain information for a resource*.
  - **Exit (Ctrl + Q)** - Closes the application.

- **Edit** menu:

  - **Undo (Ctrl + Z)** - Undo edit changes in the local file currently opened in the editor or the **Compare** view.
  - **Redo (Ctrl + Y)** - Redo edit changes in the local file currently opened in the editor or the **Compare** view.
  - **Cut (Ctrl + X)** - Cut selection to clipboard from the local file currently opened in the editor view or the **Compare** view.
  - **Copy (Ctrl + C)** - Copy selection to clipboard from the local file currently opened in the editor or the **Compare** view.
  - **Paste (Ctrl + V)** - Paste selection from clipboard in the local file currently opened in editor or the **Compare** view.
  - **Find/Replace (Ctrl + F)** - Perform find / replace operations in the local file currently opened in the editor or the **Compare** view.
  - **Find Next (F3)** - Go to the next find match using the same find options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.
  - **Find Previous (Shift + F3)** - Go to the previous find match using the same find options of the last find operation. The action runs in the editor panel and in any non-editable text area, for example the **Console** view.

- **Repository** menu:

  - **New Repository Location (Ctrl + Alt + N)** - 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

Figure 2: Add SVN Repository

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)** - Context-dependent action that allows you to edit the selected repository location by means of the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.
- **Change the Revision to Browse** **(Ctrl + Alt + Shift + B)** - Context-dependent action that allows you to change the selected repository revision by means of 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)** - Allows you to remove the selected repository location from the view. It shows you a confirmation dialog before removal. It is active only when a repository location root is selected.
- **Refresh** - Refreshes the resource selected in the **Repositories** view.
- **Check Out** **(Ctrl + Alt + Shift + C)** - Allows you to copy resources from a repository into your local file system. To use this operation, you must select a repository root location or a folder from a repository, but never a file. If you don't select anything, you can specify an URL to a folder resource from a repository in the **Check Out** dialog that appears when performing this operation. To read more about this operation, see the section *Check out a working copy*.
- **Export** - Exports a folder from the repository to the local file system.
- **Import** sub-menu:
  - **Import Folder Content** **(Ctrl + Alt + Shift + M)** - Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.
  - **Import File(s)** **(Ctrl + Alt + I)** - Imports the files selected from the files system into the selected folder from the repository.

- **Working Copy** menu:
  - **Add / Remove Working Copy** - Opens dialog with a list of working copies that the Subversion client is aware of. In this dialog you can add existing working copies or remove no longer needed ones.
  - **Switch to** - Selects one of the following view modes: **All Files**, **Modified**, **Incoming**, **Outgoing**, or **Conflicts**.
  - **Refresh** **(F5)** - Refreshes the state of the selected resources or of the entire working copy if there is no selection.
  - **Synchronize** **(Ctrl + 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 + U)** - Updates all the selected resources that have incoming changes to the HEAD revision. If one of the selected resources is a directory then the update for that resource will be recursive.

- **Update to revision/depth** - Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the *sparse checkouts* section.
- **Commit** - Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.
- **Update all (Ctrl + 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 + Shift + V)** - Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Subversion's pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.
- **Edit conflict (Ctrl + 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 + Shift + R)** - Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.
- **Mark as Merged (Ctrl + Shift + M)** - Instructs the Subversion system that you resolved the pseudo-conflict by merging the changes and you want to commit the resource. Read the *Merge conflicts* section for more information about how you can solve the pseudo-conflicts.
- **Override and Update** - Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.
- **Override and Commit** - Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.
- **Add to version control (Ctrl + Alt + V)** - Adds the selected resources to version control. A directory will be added recursively to version control. It is not mandatory to explicitly add resources to version control but it is recommended. At commit time unversioned resources will have to be manually selected in the commit dialog. This action is only active on unversioned resources.
- **Add to "svn:ignore" (Ctrl + Alt + I)** - Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the *svn:ignore* property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.
- **Cleanup (Ctrl + Shift + C)** - Performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.
- **Locking**:
  - **Scan for locks (Ctrl + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see *Scanning for locks*.
  - **Lock (Ctrl + K)** - Allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force(*steal*) the lock. The action is active only on files under version control. For more details on the use of this action see *Locking a file*.
  - **Unlock (Ctrl + Alt + K)** - Releases(unlocks) the exclusive access to a file from the repository. You can also choose to unlock it by force(*break the lock*).

  - **Expand all <u>(Ctrl + Alt + X)</u>** - Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.
  - **Collapse all <u>(Ctrl + Alt + Z)</u>** - Collapses all descendants of the selected folder. The same behavior is obtained by double-clicking on a expanded folder.
- **Compare** menu:
  - **Perform Files Differencing <u>(Ctrl + D)</u>** - Performs file differencing on request.
  - **Go to First Modification <u>(Ctrl + B)</u>** - Navigates to the first difference.
  - **Go to Previous Modification <u>(Ctrl + Shift + N)</u>** - Navigates to the previous difference.
  - **Go to Next Modification <u>(Ctrl + N)</u>** - Navigates to the next difference.
  - **Go to Last Modification <u>(Ctrl + E)</u>** - Navigates to the last difference.
  - **Copy All Non-Conflicting Changes from Right to Left** - This action copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.
  - **Copy Change from Right to Left <u>(Ctrl + Shift + Comma)</u>** - This action copies the selected change from the right editor to the left editor.
  - **Show Modification Details at Word Level** - Because the differences are computed using a line differencing algorithm sometimes is useful to see exactly what words are different in a changed section.
  - **Show Modification Details at Character Level** - Useful when you want to find out exactly what characters are different between the two analyzed sections.
  - **Ignore Whitespaces** - Enables or disables the whitespace ignoring feature. Ignoring whitespace means that before the strings are compared they are first normalized and then the whitespace at the beginning and the end of the strings is trimmed.
- **History** menu:
  - **Show History <u>(Ctrl + H)</u>** - 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 <u>(Ctrl + Shift + A)</u>** - Complex action that does the following operations:
    - opens the selected resource in the **Annotations** editor;
    - displays corresponding annotations list in the **Annotations** view;
    - displays the history of the selected resource.
  - This operation is available for any resource selected from **Repositories** view, **Working Copy** view, **History** view or **Directory Change Sets** view, depending on which view was last focused when this action was invoked.
  - **Revision Graph <u>(Ctrl + Shift + G)</u>** - 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:
  - **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 <u>(Ctrl + J)</u>** - 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 <u>(Ctrl + Alt + W)</u>** - Allows you to change the repository location of a working copy or only of a versioned item of the working copy within the same repository. It is available when the selected item of the working copy

  is a versioned resource, except an external folder. To read more about this action, see the section *Switching the Repository Location*.
  - **Relocate** - Allows you to change the base URL of the root folder of the working copy to a new URL, when the base URL of the repository changed, for example the repository itself was relocated to a different server. This operation is available for a selected item of the working copy tree that is a versioned folder. To read more about this operation, see the section *Relocate a Working Copy*.
  - **Create patch** **(Ctrl + Alt + P)** - Allows you to create a file containing all the differences between two resources, based on the `svn diff` command. To read more about creating patches, see *the section about patches*.
  - **Working copy format** - this submenu contains the following two operations:
    - **Upgrade** - Allows you to upgrade the format of the current working copy to the newest one known by Syncro SVN Client, to allow you to benefit of all the new features of the client.
    - **Downgrade** - Allows you to downgrade the format of the current working copy to an older format. The formats allowed to downgrade to are SVN 1.5 and SVN 1.4. This is useful in case you wish to use older SVN clients with the current working copy, or, by mistake, you have upgraded the format of an older working copy by using a newer SVN client.

    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.
  - **Reset Layout** - Resets all the views to their default position.
- **Help** menu:
  - **Help (F1)** - Opens the **Help** dialog.
  - **Dynamic Help** - Shows the **Dynamic Help** view.
  - **Check for New Versions** - Checks the availability of new Syncro SVN Client versions.
  - **Browse Syncro SVN Client Website** - Opens the Syncro SVN Client website in a browser.
  - **Register** - Opens the registration dialog.
  - **Improvement Program Options** - Allows you to activate or deactivate the Syncro Soft Product Improvement Program.
  - **Report Problem** - Opens a dialog that allows the user to write the description of a problem that was encountered while using the application.
  - **Support Center** - Opens the Support Center web page in a browser.
  - **About** - Opens the **About** dialog.

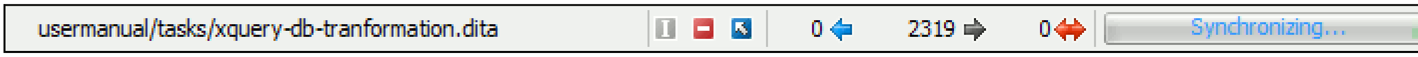
Main Toolbar

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

- **Check Out** - Checks out a working copy from a repository. The repository URL and the working copy format must be specified.
- **Synchronize** - Synchronizes the current working copy with the repository.
- **Update All** - Updates all resources of the working copy that have an older revision that repository.
- **Commit All** - Commits all resources of working copy that have a newer version compared to that of the repository.
- **Refresh** - Refreshes the whole content of the current working copy from disk starting from the root folder. At the end of the operation, the modified files and folders that were not committed to repository yet, are displayed in the **Working Copy** view.
- **Compare** - The selected resource is compared with:
  - the *BASE* revision, when the selected resource is:
    - locally modified and the **All Files** view mode is currently selected (no matter if there are incoming changes);
    - locally modified and there are no incoming changes when any other view mode is selected.
  - the remote version of the same resource, when remote information is available after a **Synchronize** operation (only when one of **Modified**, **Incoming**, **Outgoing** and **Conflicts** view modes is selected).
  - the working copy revision, when the selected resource is from the **History** view;
- **Show History** - Displays the history of the selected resource (from the **Working Copy** or **Repository** views) in the **History** view.
- **Show Annotation** - Displays the annotations of the selected resource. The selected resource can be in the **Working Copy** or the **History** views.
- **Revision Graph** - Displays the revision graph of the selected resource. The selected resource can be in the **Working Copy** or the **Repositories** views.

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.



usermanual/tasks/xquery-db-tranformation.dita

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 **Checkout** 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 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 copies and must share their work. This is done via a Subversion repository. Syncro SVN Client supports the versions 1.3, 1.4, 1.5 and 1.6 of the SVN repository format.

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

To create a repository location, click the **New Repository Location** toolbar button or right click inside the view and select **New Repository Location...** from the popup menu. On Windows, the context menu can be displayed on a right click with the mouse or with the keyboard by pressing the special context menu key available on Windows keyboards. This action opens the **Add SVN Repository** dialog which prompts you for the URL of the repository you want to connect to. No authentication information is requested at the time the location is defined. It is left to the Subversion client to request the user and password information when it is needed. The main benefit of allowing Subversion to manage your password in this way is that it prompts you for a new password only when your password changes.

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

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

The **Edit SVN Repository** dialog works in the same way as the **Add SVN Repository** dialog. It shows the previously defined repositories URLs and it allows you to change them.

To remove a repository location, click the ✕ **Remove Repository Location** toolbar button or right click inside the view on a repository entry and select **Remove Repository Location...** from the popup menu. A confirmation dialog is displayed to make sure that you do not accidentally remove locations.

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

To set the reference revision number of an SVN repository right-click on the repository in the list displayed in the ***Repositories** view* and select the **Change the Revision to Browse...** action. The revision number of the repository is used for displaying the contents of the repository when it is viewed in the ***Repositories** view*. Only the files and folders that were present in the repository at the moment when this revision number was generated on the repository are displayed as contents of the repository tree. Also this revision number is used for all the file open operations executed directly from the ***Repositories** view*.

Authentication

Five protocols are supported: *HTTP*, *SVN*, *HTTPS*, *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 Subversion's default directory:

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

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

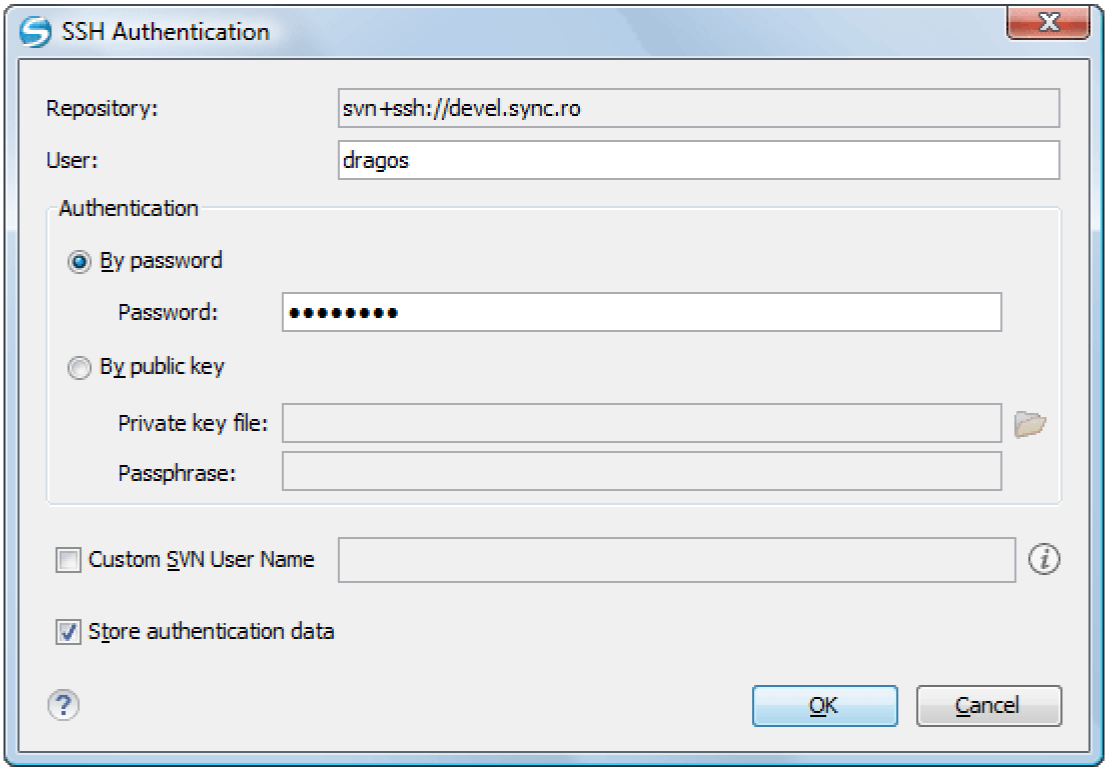
Note: When you reset the authentication data, restart the application in order for the change to take effect.

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

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

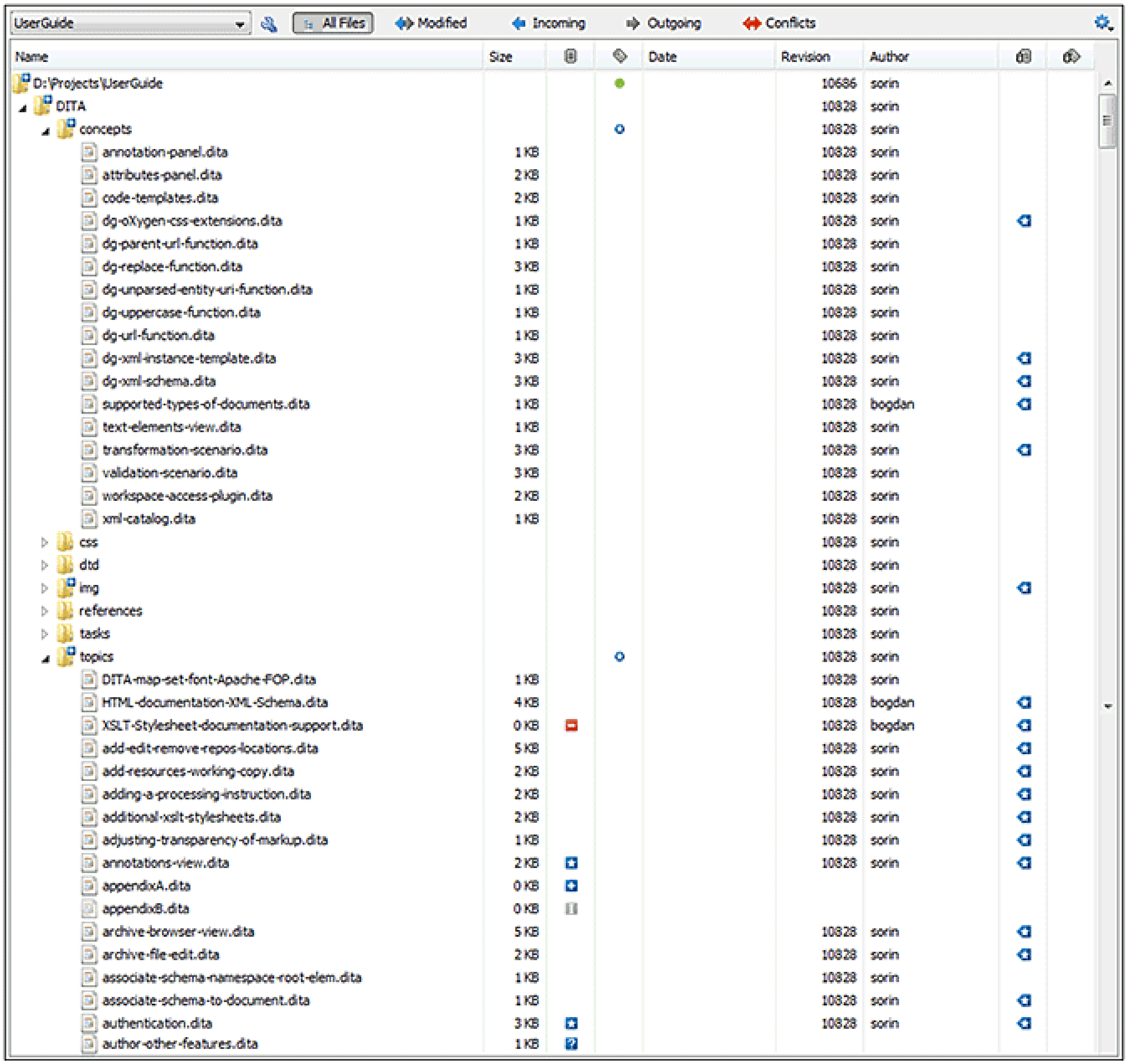
☑ Store authentication data

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.

Defining a Working Copy

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



Name	Size			Date	Revision	Author		
D:\Projects\UserGuide			●		10686	sorin		
DITA					10828	sorin		
concepts			○		10828	sorin		
annotation-panel.dita	1 KB				10828	sorin		
attributes-panel.dita	2 KB				10828	sorin		
code-templates.dita	2 KB				10828	sorin		
dg-oXygen-css-extensions.dita	1 KB				10828	sorin	←	
dg-parent-url-function.dita	1 KB				10828	sorin		
dg-replace-function.dita	3 KB				10828	sorin		
dg-unparsed-entity-uri-function.dita	1 KB				10828	sorin		
dg-uppercase-function.dita	1 KB				10828	sorin		
dg-url-function.dita	1 KB				10828	sorin		
dg-xml-instance-template.dita	3 KB				10828	sorin	←	
dg-xml-schema.dita	3 KB				10828	sorin	←	
supported-types-of-documents.dita	1 KB				10828	bogdan	←	
text-elements-view.dita	1 KB				10828	sorin		
transformation-scenario.dita	3 KB				10828	sorin	←	
validation-scenario.dita	3 KB				10828	sorin		
workspace-access-plugin.dita	2 KB				10828	sorin		
xml-catalog.dita	1 KB				10828	sorin		
css					10828	sorin		
dtd					10828	sorin		
img					10828	sorin	←	
references					10828	sorin		
tasks					10828	sorin		
topics			○		10828	sorin		
DITA-map-set-font-Apache-FOP.dita	1 KB				10828	sorin		
HTML-documentation-XML-Schema.dita	4 KB				10828	bogdan	←	
XSLT-Stylesheet-documentation-support.dita	0 KB	⊟			10828	bogdan	←	
add-edit-remove-repos-locations.dita	5 KB				10828	sorin	←	
add-resources-working-copy.dita	2 KB				10828	sorin	←	
adding-a-processing-instruction.dita	2 KB				10828	sorin	←	
additional-xslt-stylesheets.dita	2 KB				10828	sorin	←	
adjusting-transparency-of-markup.dita	1 KB				10828	sorin	←	
annotations-view.dita	2 KB	⊞			10828	sorin	←	
appendixA.dita	0 KB	⊞						
appendixB.dita	0 KB	▯						
archive-browser-view.dita	5 KB				10828	sorin	←	
archive-file-edit.dita	2 KB				10828	sorin	←	
associate-schema-namespace-root-elem.dita	1 KB				10828	sorin		
associate-schema-to-document.dita	1 KB				10828	sorin	←	
authentication.dita	3 KB	⊞			10828	sorin	←	
author-other-features.dita	1 KB	?						

Figure 5: Working Copy View

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

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

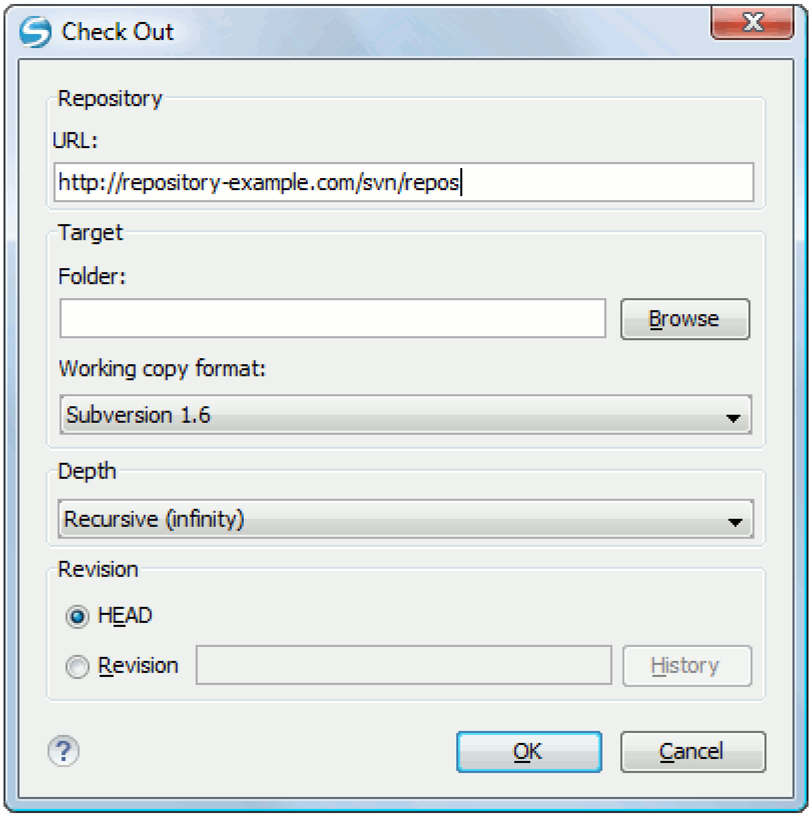
Check Out a Working Copy

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

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

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

   The **Check Out** dialog is displayed:



Check Out

Repository

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

Target

Folder:

Working copy format: Subversion 1.6

Depth

Recursive (infinity)

Revision

● HEAD

○ Revision

Figure 6: Check Out Dialog

3. Click on the **Browse** button.
4. Select the location where the working copy is created.
5. Select the version of the working copy format: SVN 1.4, SVN 1.5 or SVN 1.6.
6. Select the depth for the checkout folder in the **Depth** combo box.

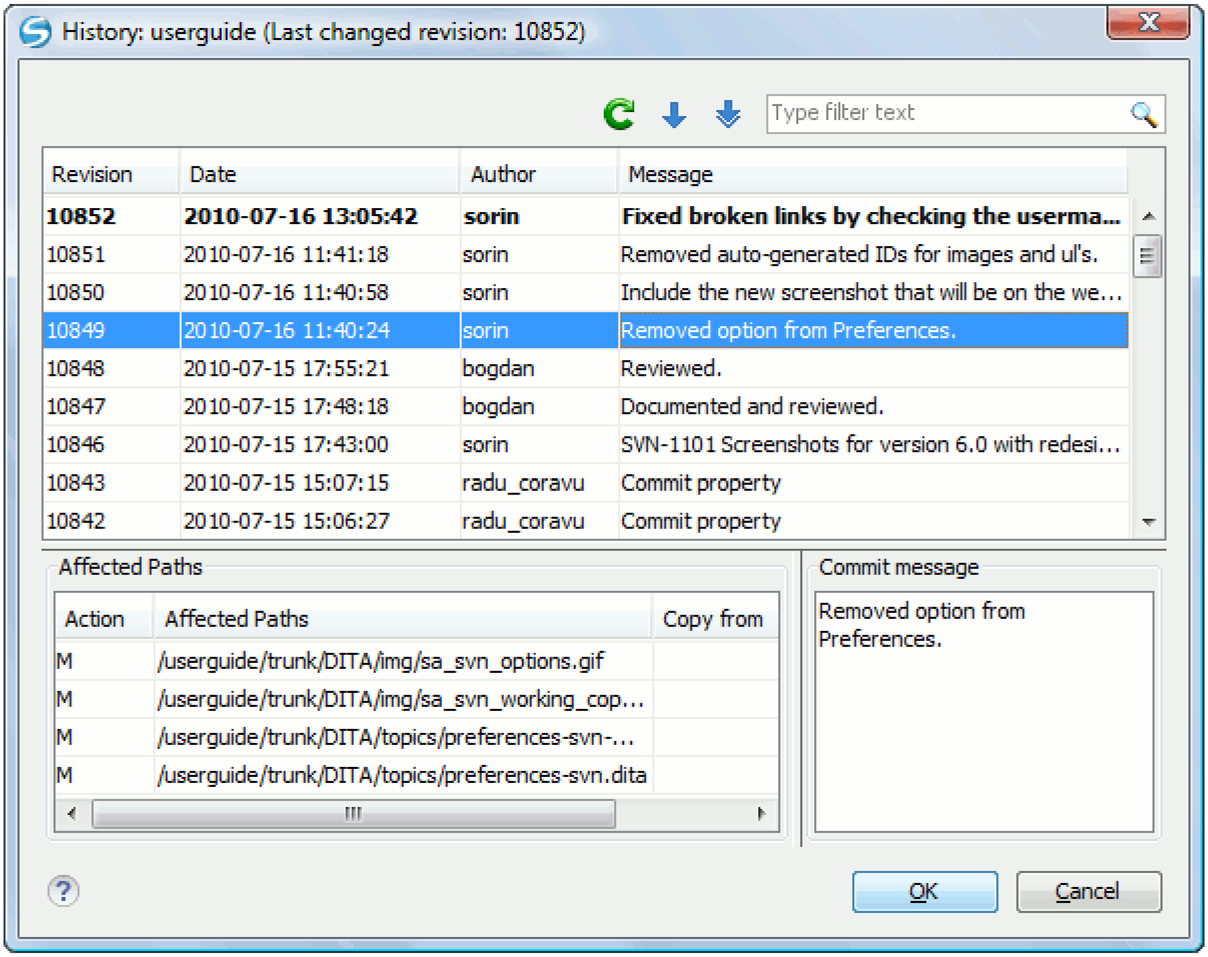
   This allows you to specify the recursion level into child resources. It is used if you want to check out only a portion of a working copy and then bring in a future update operation previously ignored files and subdirectories. You can find out more about checkout depth in the *sparse checkouts* section.
7. Select the revision number that is checked out.

By default the last (HEAD) revision is checked out. If you need another revision, you have to select the **Revision** radio button. To specify the revision number you can simply type the revision number in the corresponding text field or click on the **History** button which opens *the **History** dialog*.

After a check out operation, the new working copy will be added to the list in the ***Working Copy** view*. The working copy content is displayed in that view.

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*. It presents information about revision, commit date, author, and commit comment.



History: userguide (Last changed revision: 10852)

Type filter text

Revision	Date	Author	Message
10852	**2010-07-16 13:05:42**	**sorin**	**Fixed broken links by checking the userma...**
10851	2010-07-16 11:41:18	sorin	Removed auto-generated IDs for images and ul's.
10850	2010-07-16 11:40:58	sorin	Include the new screenshot that will be on the we...
10849	2010-07-16 11:40:24	sorin	Removed option from Preferences.
10848	2010-07-15 17:55:21	bogdan	Reviewed.
10847	2010-07-15 17:48:18	bogdan	Documented and reviewed.
10846	2010-07-15 17:43:00	sorin	SVN-1101 Screenshots for version 6.0 with redesi...
10843	2010-07-15 15:07:15	radu_coravu	Commit property
10842	2010-07-15 15:06:27	radu_coravu	Commit property

Affected Paths

Action	Affected Paths	Copy from
M	/userguide/trunk/DITA/img/sa_svn_options.gif	
M	/userguide/trunk/DITA/img/sa_svn_working_cop...	
M	/userguide/trunk/DITA/topics/preferences-svn-...	
M	/userguide/trunk/DITA/topics/preferences-svn.dita	

Commit message

Removed option from Preferences.

Figure 7: 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.

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

- **Compare with previous version** - Makes a diff between the selected revision and the previous one. If there is no external application specified for executing diff operations, the built-in diff tool is applied. The same action is also executed when double clicking a file in the **Affected Paths** area.
- **Open** - Opens the revision in the editor panel.
- **Save revision to ...** - Saves the revision to a new file.

- **Revert changes from this revision** - The changes committed by the selected revision are reverted in the current version of the file in the working copy. If the committed changes represented in fact an SVN delete operation, the result is restoring the deleted file in the working copy.
- **Update to revision** - Makes the selected revision the current revision in the working copy.
- **Show History** - Displays the history of the selected revision.
- **Show Annotation** - Opens the **Annotations** view for the selected revision.

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 Subversion. If you have a brand new project that you want to import into your repository, then see the section *Import resources into the repository*. The following procedure assumes that you have an existing valid working copy on your file system.

1. Click on the **Add / Remove Working Copy** toolbar button 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.
4. Optionally you can press the **Edit** button to change the name of the working copy that is displayed in the **Working Copy** view.

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

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

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

Manage Working Copy Resources

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

Edit Files

You can edit files from the ***Working Copy*** *view* by double clicking them or by right clicking them and choosing **Open** from the contextual menu.

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

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

Add Resources to Version Control

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

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

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

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*, 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 command can be found in the **Edit** submenu of the context menu from the ***Working Copy*** *view*. When you delete a resource from the Subversion working copy, it is removed from the file system and it is also marked as deleted. If unversioned, added or modified resources are encountered, a dialog prompts you to confirm their deletion.

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

If you delete a resource from the file system without Subversion's knowledge, you render the working copy in an inconsistent state.

If a resource is deleted from the file system without Subversion's knowledge, your working copy is in an inconsistent state. The resource will be considered and marked as missing (the sign '!'). If a file was deleted, it will be treated in the same way as if it was deleted by Subversion. However if a directory is missing you will be unable to commit. If you *update your working copy*, Subversion will replace the missing directory with the latest version from the repository and you can then delete it the correct way using the **Delete** command.

Note: The **Delete** action is not enabled when the selection contains *missing* resources.

Copy Resources

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

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

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

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

Move Resources

As in the case of the copy command you can perform the move operation on several resources at once. Just select the resources in the ***Working Copy*** *view* and choose the **Move** command 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.

Rename Resources

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

Lock / Unlock Resources

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

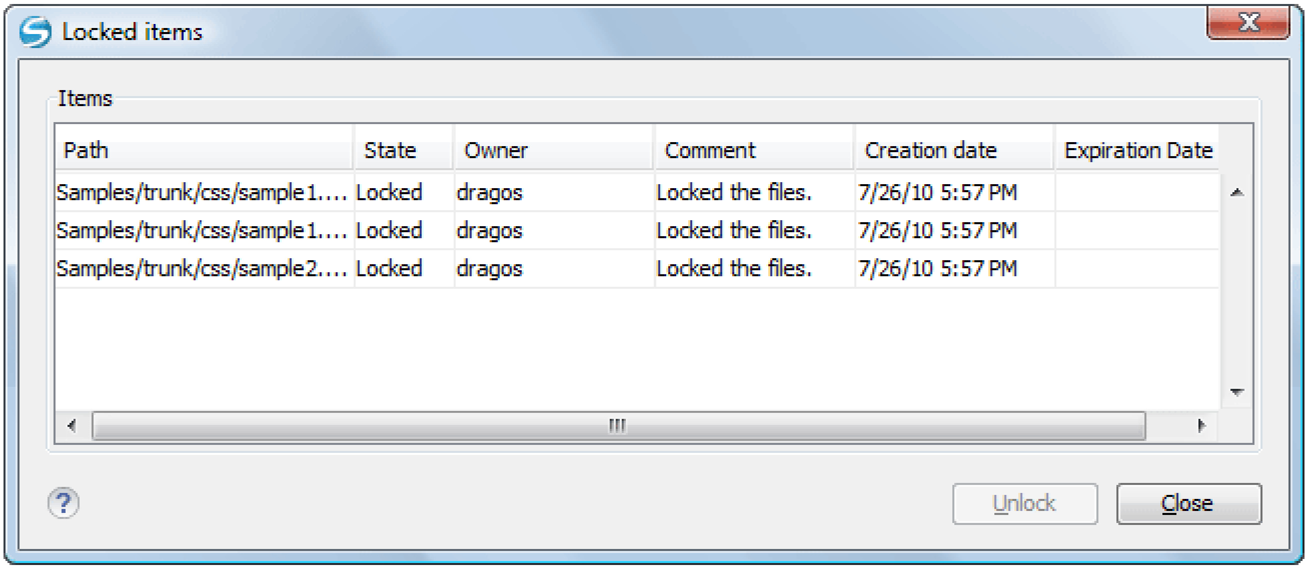
The copy-modify-merge model only works when files are contextually mergeable: this is usually the case of line-based text files (such as source code). However this is not always possible with binary formats, such as images or sounds. In these situations, the users must each have exclusive access to the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

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

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

Scanning for Locks

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



Path	State	Owner	Comment	Creation date	Expiration Date
Samples/trunk/css/sample1....	Locked	dragos	Locked the files.	7/26/10 5:57 PM	
Samples/trunk/css/sample1....	Locked	dragos	Locked the files.	7/26/10 5:57 PM	
Samples/trunk/css/sample2....	Locked	dragos	Locked the files.	7/26/10 5:57 PM	

Figure 8: 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:

- **Other** - If someone else locked the file.
- **Locked** - If the current user locked the file.
- **Broken** - If the current user locked the file but it was forcefully unlocked by someone else afterwards.
- **Stolen** - If the current user locked the file but it was forcefully locked by someone else afterwards.

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

Locking a File

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

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

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

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

Unlocking a File

A file can be unlocked from the 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 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*.

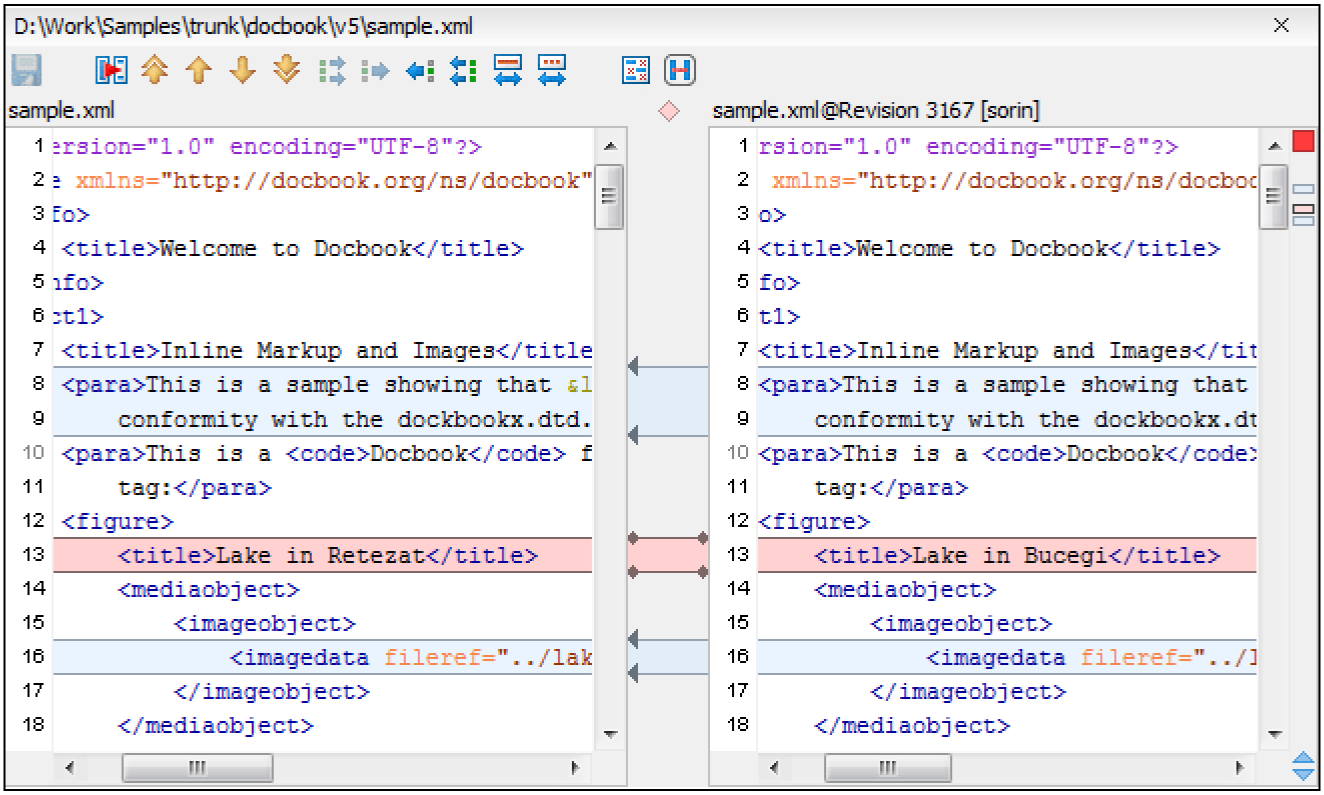
View Differences

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*.



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

sample.xml

```
ersion="1.0" encoding="UTF-8"?>
e xmlns="http://docbook.org/ns/docbook"
fo>
 <title>Welcome to Docbook</title>
nfo>
ct1>
 <title>Inline Markup and Images</title
 <para>This is a sample showing that &l
     conformity with the dockbookx.dtd.
 <para>This is a <code>Docbook</code> f
     tag:</para>
 <figure>
     <title>Lake in Retezat</title>
     <mediaobject>
         <imageobject>
             <imagedata fileref="../lak
         </imageobject>
     </mediaobject>
```

sample.xml@Revision 3167 [sorin]

```
rsion="1.0" encoding="UTF-8"?>
 xmlns="http://docbook.org/ns/docboo
o>
<title>Welcome to Docbook</title>
fo>
t1>
<title>Inline Markup and Images</tit
<para>This is a sample showing that
    conformity with the dockbookx.dt
<para>This is a <code>Docbook</code>
    tag:</para>
<figure>
    <title>Lake in Bucegi</title>
    <mediaobject>
        <imageobject>
            <imagedata fileref="../l
        </imageobject>
    </mediaobject>
```

Figure 9: 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, 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 operations on the differences between two versions of a file are available on the toolbar of this view:

- **Save** - Saves the modifications of the local version of the file displayed in the left side of the view.
- **Perform Files Differencing** - Applies the diff operation on the two versions of the file displayed in the view. It is useful after modifying the local version displayed in the left side of the view.
- **Go to First Modification** - Scrolls the view to the topmost difference.
- **Go to Previous Modification** - Scrolls the view to the previous difference. The current difference is painted with a darker color than the other ones.
- **Go to Next Modification** - Scrolls the view to the next difference. The current difference is painted with a darker color than the other ones.
- **Go to Last Modification** - Scrolls the view to the last difference.
- **Copy All Non Conflicting Changes from Left to Right** - Not applicable for editing conflicts so it is disabled.
- **Copy Change from Left to Right** - Not applicable for editing conflicts so it is disabled.
- **Copy Change from Right to Left** - Copies the current difference from the left side to the right side by replacing the highlighted text of the current difference from the left side with the one from the right side.
- **Copy All Non Conflicting Changes from Right to Left** - Applies the previous operation for all the differences.
- **Show Modification Details at Word Level** - Displays a more detailed version of the current difference computed at word level.
- **Show Modification Details at Char Level** - Displays a more detailed version of the current difference computed at character level.
- **Ignore Whitespaces** - The text nodes are normalized before computing the difference so that if two text nodes differ only in whitespace characters they are reported as equal.

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

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

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

After the conflicts are edited and saved in the local version of the file you should run:

- either the action **Mark Resolved** on the file so that the result of the conflict editing process can be committed to the SVN repository,
- or the action **Revert** so that the repository version overwrites all the local modifications.

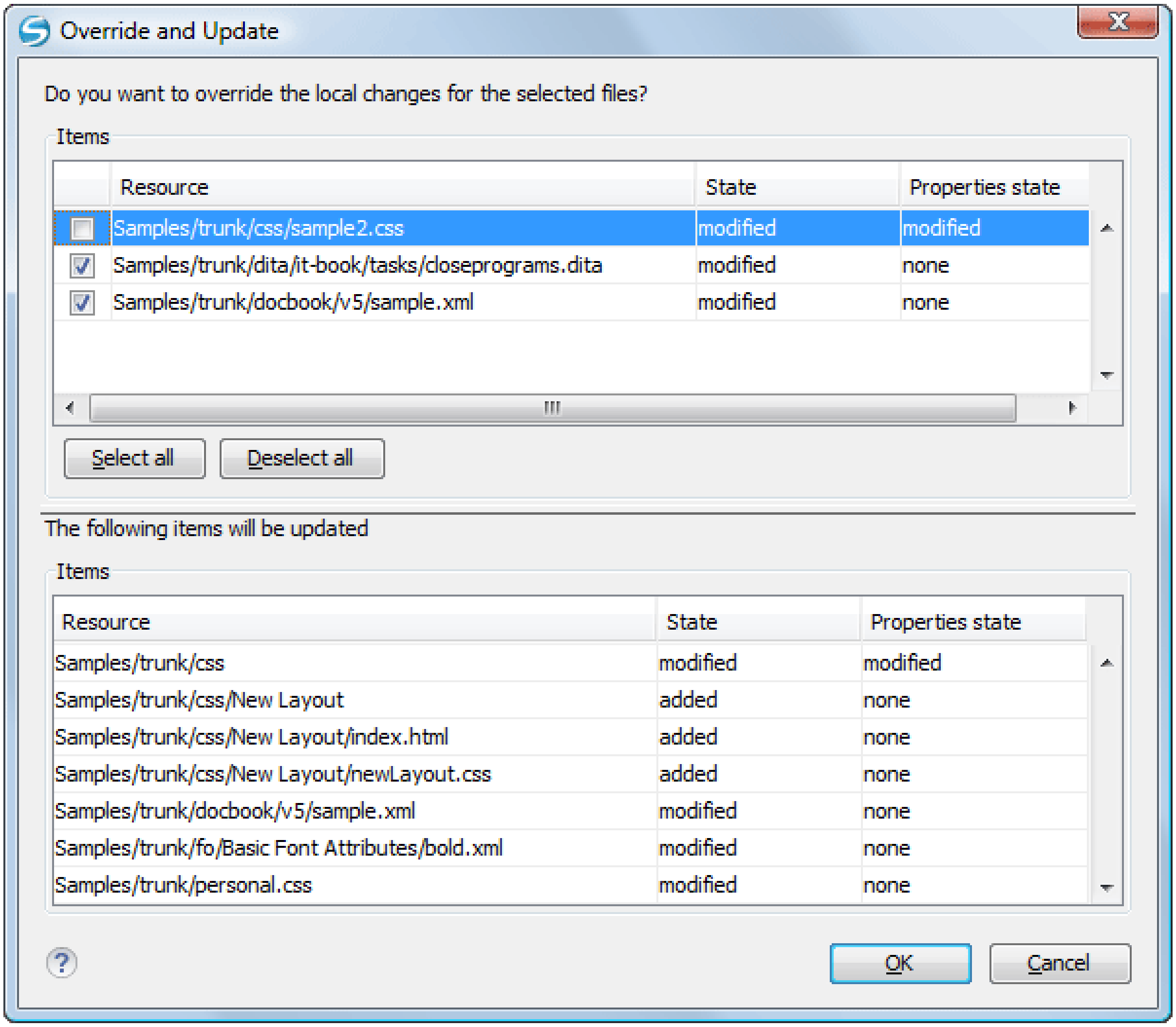
Both actions remove the backup file and other temporary files created with the conflict version of the local file.

Revert Your Changes

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

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

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



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

	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

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 10: Override and update dialog

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

Merge Conflicted Resources

Before you can safely commit your changes to the repository you must first resolve all conflicts. In the case of pseudo-conflicts they can be resolved in most cases with an update operation which will merge the incoming modifications into your working copy resource. In the case of real conflicts, conflicts that persist after an update operation, it is necessary to resolve the conflict using the built-in compare view and editor or, in the case of properties conflict, the ***Properties** view*. Before you can commit you must *mark as resolved* the affected files.

Both pseudo and real conflicts can be resolved without an update. You should open the file in the compare editor and decide which incoming changes need to be copied locally and which outgoing changes must be overridden or modified. After saving your local file you have to use the *Mark as merged* action from the contextual menu before committing.

Drop Incoming Modifications

In the situation when your file is in conflict but you decide that your working copy file and its content is the correct one, you can decide to drop some or all of the incoming changes and commit afterwards. The action **Mark as merged** proves to be useful in this case too. After opening the conflicting files with ***Compare** view*, *Editor* or editing their properties in the **Properties** view and deciding that your file can be committed in the repository replacing the existing one, you should use the **Mark as merged** action. When you want to override completely the remote file with the local file you should run the action **Override and commit** which drops any remote changes and commits your file.

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

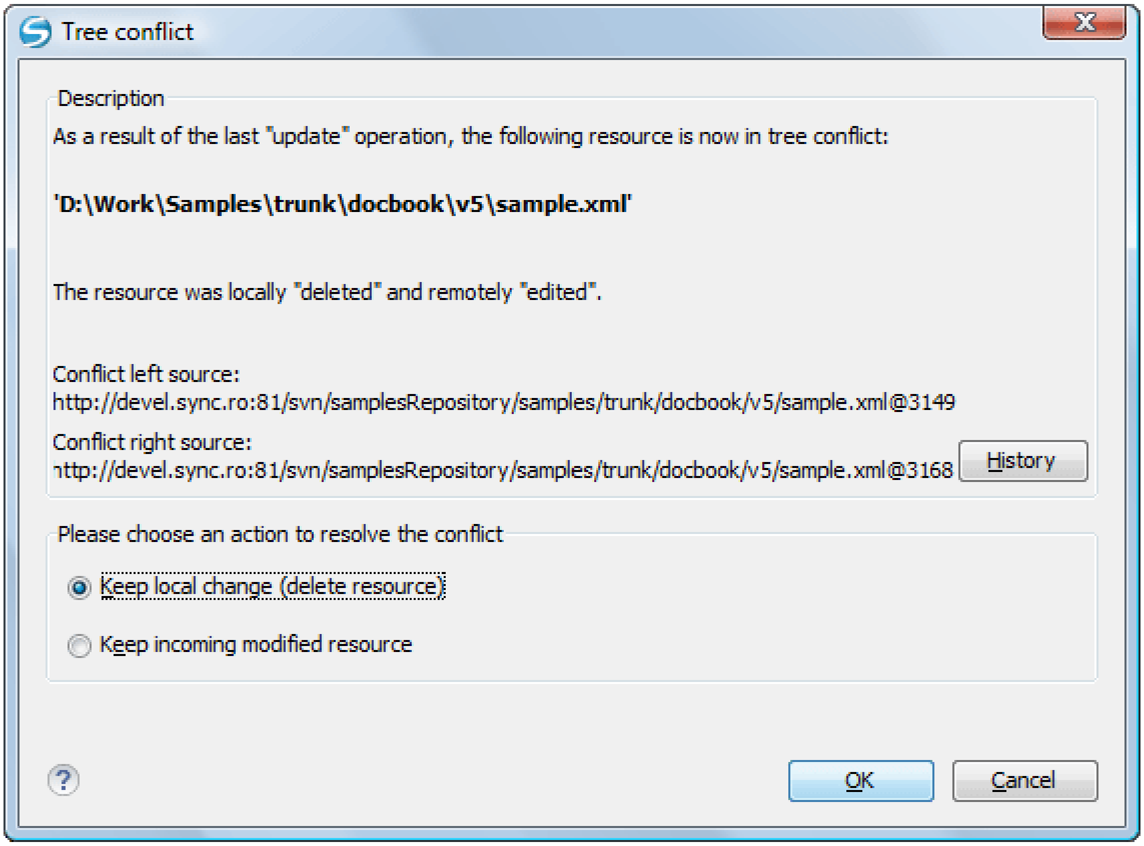
Tree Conflicts

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

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

The same conflict situation can occur after a merge or a switch action. The action ends with an error and the folder containing the file that is now in the tree conflict state is also marked with a conflict icon.

Such a conflict can be resolved in one of the following ways which are available when the user double clicks on the conflicting resource or when running the **Edit conflict** action:



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 11: 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.

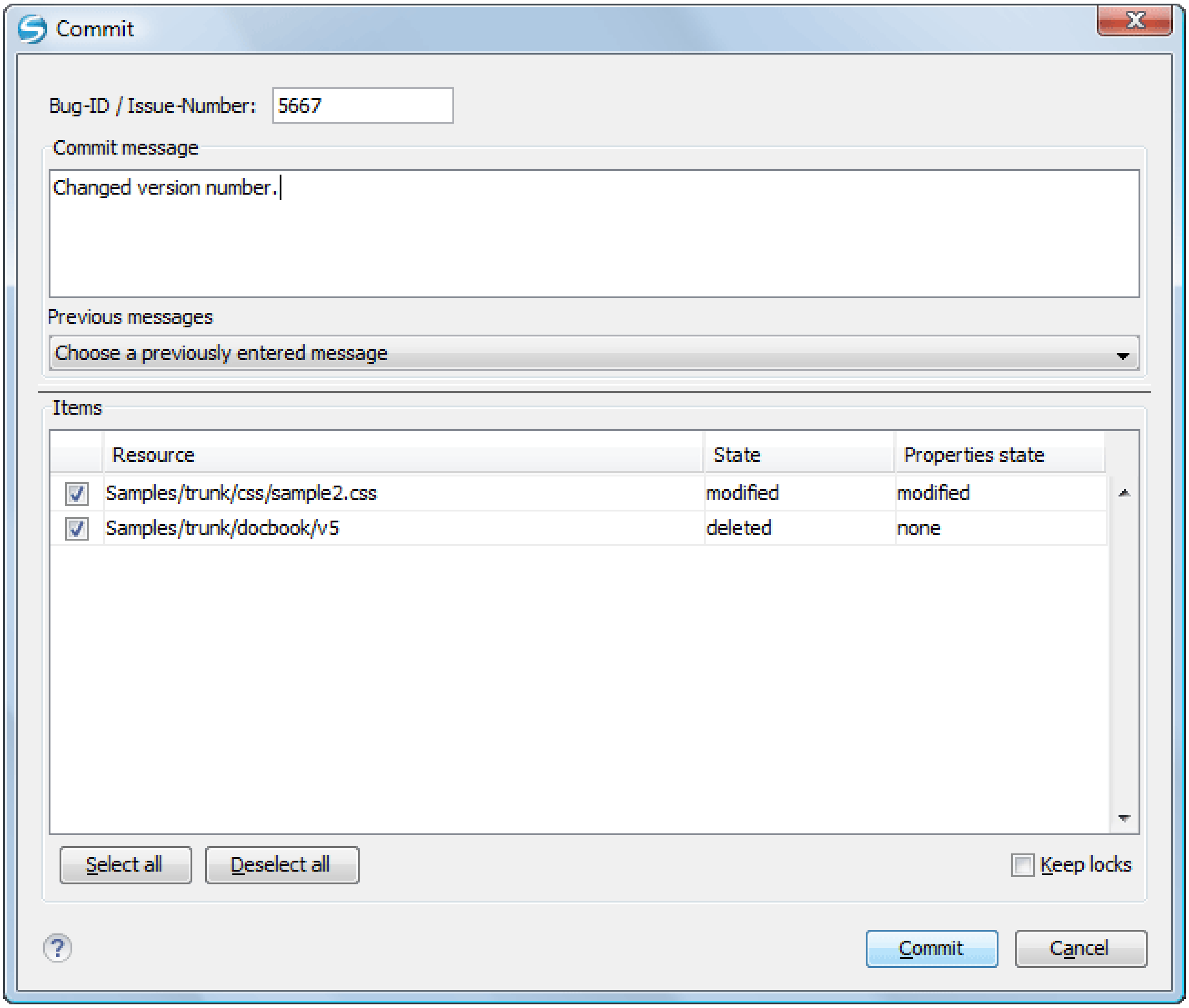
If you select multiple files and folders and then you perform an **Update** operation, all of those files and folders are updated one by one. The Subversion client makes sure that all files and folders belonging to the same repository are updated to the exact same revision, even if between those updates 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 checkout 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 in your local working copy to the repository. After selecting the action from the contextual menu you will see a dialog displaying the resources that can be committed.



Bug-ID / Issue-Number: 5667

Commit message

Changed version number.

Previous messages

Choose a previously entered message

Items

	Resource	State	Properties state
☑	Samples/trunk/css/sample2.css	modified	modified
☑	Samples/trunk/docbook/v5	deleted	none

□ Keep locks

Figure 12: Commit dialog

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

To select all resources, click **Select All**. To deselect all resources, click **Deselect All**. Checking the **Keep locks** option will preserve any locks you have on repository resources. Your working copy must be up-to-date with respect to the resources you are committing. This is ensured by using the **Update** action prior to committing, resolving conflicts and re-testing as needed. If your working copy resources you are trying to commit are *out of date* you will get an appropriate error message.

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

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

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

Integration with Bug Tracking Tools

Users of bug tracking systems can associate the changes they make in the repository resources with a specific ID in their bug tracking system. The only requirement is that the user includes the bug ID in the commit message that he enters in the **Commit** dialog. The format and the location of the ID in the commit message are configured with SVN properties.

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

- **bugtraq:message** - A string property. If it is set *the **Commit** dialog* will display a text field for entering the bug ID. It must contain the string *%BUGID%*, which is replaced with the bug number on commit.
- **bugtraq:label** - A string property that sets the label for the text field configured with the **bugtraq:message** property.
- **bugtraq:url** - A string property that is the URL pointing to the bug tracking tool. The URL string should contain the substring *%BUGID%* which Syncro SVN Client replaces with the issue number. That way the resulting URL will point directly to the correct issue.
- **bugtraq:warnifnoissue** - A boolean property with the values *true*/*yes* or *false*/*no*. If set to *true*, the Syncro SVN Client will warn you if the bug ID text field is left empty. The warning will not block the commit, only give you a chance to enter an issue number.
- **bugtraq:number** - A boolean property with the value *true* or *false*. If this property is set to *false*, then any character can be entered in the bug ID text field. If the property is set to *true* or is missing then only numbers are allowed as the bug ID.
- **bugtraq:append** - A boolean property. If set to *false*, then the bug ID is inserted at the beginning of the commit message. If *yes* or not set, then it's appended to the commit message.
- **bugtraq:logregex** - This property contains one or two regular expressions, separated by a newline. If only one expression is set, then the bug ID's must be matched in the groups of the regular expression string, for example `[Ii]ssue #?(\d+)` If two expressions are set, then the first expression is used to find a string which relates to a bug ID but may contain more than just the bug ID (e.g. `Issue #123` or `resolves issue 123`). The second expression is then used to extract the bug ID from the string extracted with the first expression. An example: if you want to catch every pattern `issue #XXX` and `issue #890, #789` inside a log message you could use the following strings:
  - `[Ii]ssue #?(\d+)(,? ?#?(\d+))+`
  - `(\d+)`

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

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

Obtain Information for a Resource

This section explains how to obtain information for a SVN resource:

Request Status Information for a Resource

While you are working you often need to know which files you have changed, added, removed or renamed, or even which files got changed and committed by others. That's where the **Synchronize** action from ***Working Copy** view* comes in handy. The **Working Copy** view will show you every file that has changed in any way in your working copy, as well as any unversioned files you may have.

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

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

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

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

Request History for a Resource

In 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 should load the history data of a resource in *the **History** view* by running the **Show History** action. This action is available in the **Working Copy** and **Repositories** views and also from the menus with the same name.

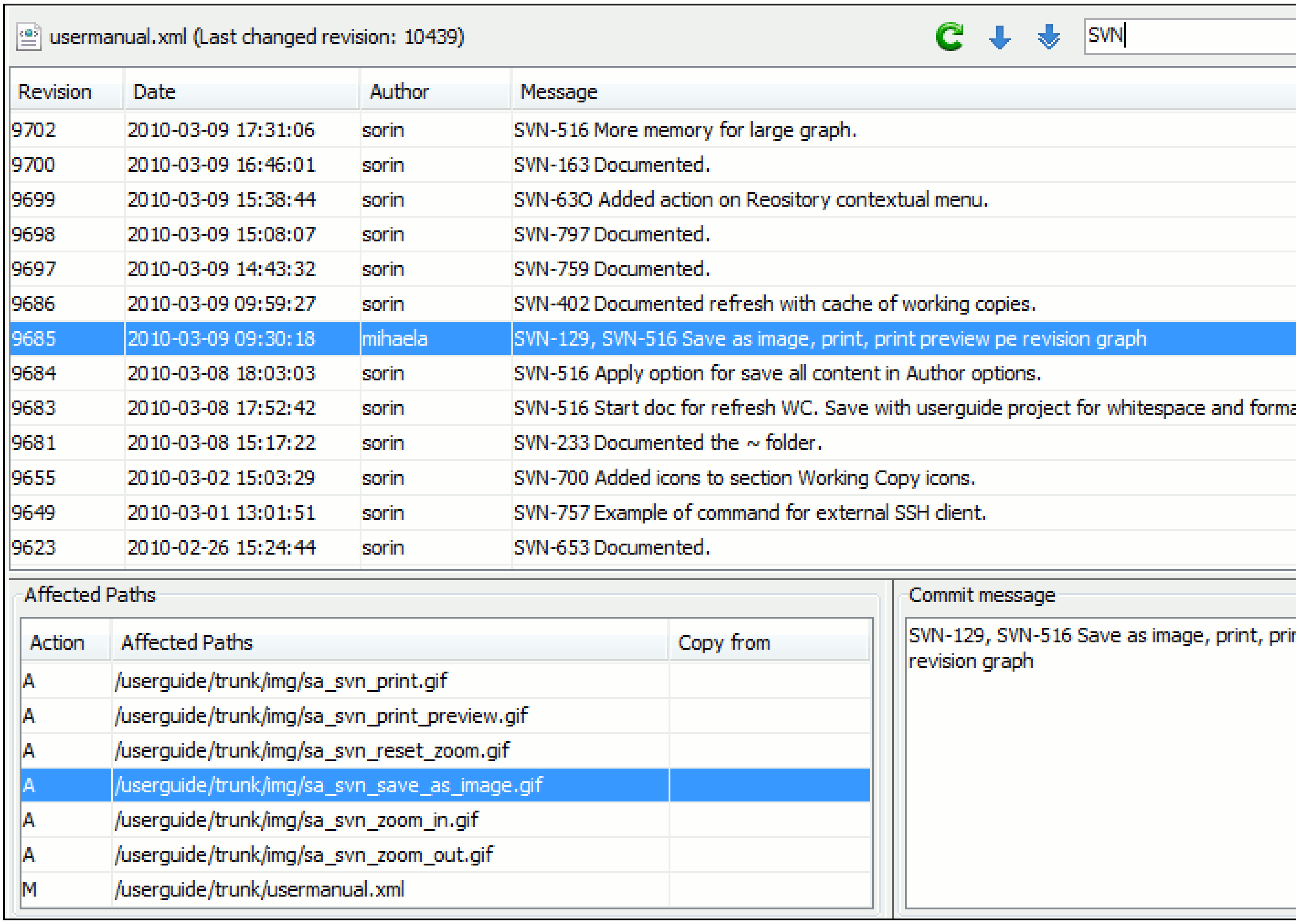
From the **Repositories** view you can display the log history regarding any remote resource residing in repository. From the **Working Copy** view you can display the history of local versioned resources and newly incoming resources.

History View

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

The view consists of three distinct areas:

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



usermanual.xml (Last changed revision: 10439)

SVN

Revision	Date	Author	Message
9702	2010-03-09 17:31:06	sorin	SVN-516 More memory for large graph.
9700	2010-03-09 16:46:01	sorin	SVN-163 Documented.
9699	2010-03-09 15:38:44	sorin	SVN-630 Added action on Reository contextual menu.
9698	2010-03-09 15:08:07	sorin	SVN-797 Documented.
9697	2010-03-09 14:43:32	sorin	SVN-759 Documented.
9686	2010-03-09 09:59:27	sorin	SVN-402 Documented refresh with cache of working copies.
9685	2010-03-09 09:30:18	mihaela	SVN-129, SVN-516 Save as image, print, print preview pe revision graph
9684	2010-03-08 18:03:03	sorin	SVN-516 Apply option for save all content in Author options.
9683	2010-03-08 17:52:42	sorin	SVN-516 Start doc for refresh WC. Save with userguide project for whitespace and forma
9681	2010-03-08 15:17:22	sorin	SVN-233 Documented the ~ folder.
9655	2010-03-02 15:03:29	sorin	SVN-700 Added icons to section Working Copy icons.
9649	2010-03-01 13:01:51	sorin	SVN-757 Example of command for external SSH client.
9623	2010-02-26 15:24:44	sorin	SVN-653 Documented.

Affected Paths

Action	Affected Paths	Copy from
A	/userguide/trunk/img/sa_svn_print.gif	
A	/userguide/trunk/img/sa_svn_print_preview.gif	
A	/userguide/trunk/img/sa_svn_reset_zoom.gif	
A	/userguide/trunk/img/sa_svn_save_as_image.gif	
A	/userguide/trunk/img/sa_svn_zoom_in.gif	
A	/userguide/trunk/img/sa_svn_zoom_out.gif	
M	/userguide/trunk/usermanual.xml	

Commit message

SVN-129, SVN-516 Save as image, print, prir
revision graph

Figure 13: History View

The Contextual Menu of the History View

When a single resource is selected in the **History** view, the contextual menu contains the following actions:

- **Compare** - 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 action is enabled only if the resource is a file.
- **Open With...** - Displays the ***Open with...*** dialog for specifying the editor in which the selected file revision is opened. This action is enabled only if the resource is a file.
- **Get Contents** - Replaces the current content of the local version of the selected file with the content of the selected revision.
- **Save revision to...** - Saves the selected revision to a file so you have an older version of that file. This option is available only when you access the history of a file, and it saves a version of that one file only.
- **Revert changes from this revision** - Reverts changes which were made in the selected revision. The changes are reverted in your working copy so this operation does not affect the repository file. The action undoes the changes made only in selected revision. It does not replace your working copy file with the entire file at the earlier revision. This action is useful for undoing an earlier change when other unrelated changes have been made since the date of the revision. This action is enabled when the resource history was launched for a local working copy resource.
- **Update to revision** - Updates your working copy resource to the selected revision. Useful if you want to have your working copy reflect a time in the past. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy is inconsistent and you are unable to commit your changes.
- **Check out from revision...** - Gets the content of the selected revision into local file system.

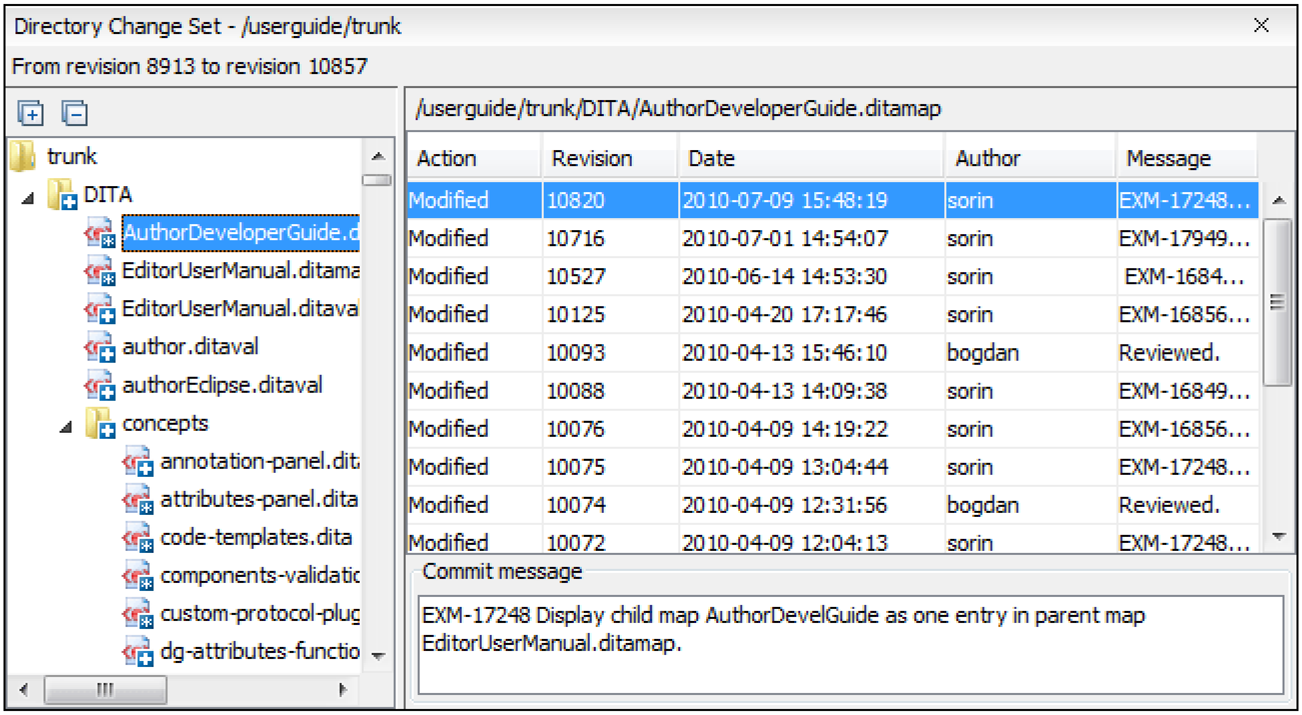
- **Show Annotation...** - Brings up a dialog for selecting the start revision and the end revision of the interval of revisions for which the SVN annotations is computed and marked in the selected resource in the editor panel. This action is available only when you access the history of a file.
- **Change** - Allows you to change commit data for a file:
  - *Author* - Allows you to change the user name that committed the selected file revision.
  - *Message* - Allows you to change the commit message of the selected file 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.

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

/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 14: Directory Change Set View

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

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

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

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

- **Compare with previous version** - Performs a diff operation between the selected revision in the list and the previous one.
- **Open** - Opens the selected revision in the associated editor type.
- **Open with...** - *Displays a dialog* with the available editor types and allows the user to select the editor type for opening the selected revision.
- **Save revision to...** - Saves the selected revision in a file on disk.
- **Show Annotation** - Requests the annotations of the file and *displays them in the **Annotations** view*.

Management of SVN Properties

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

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).

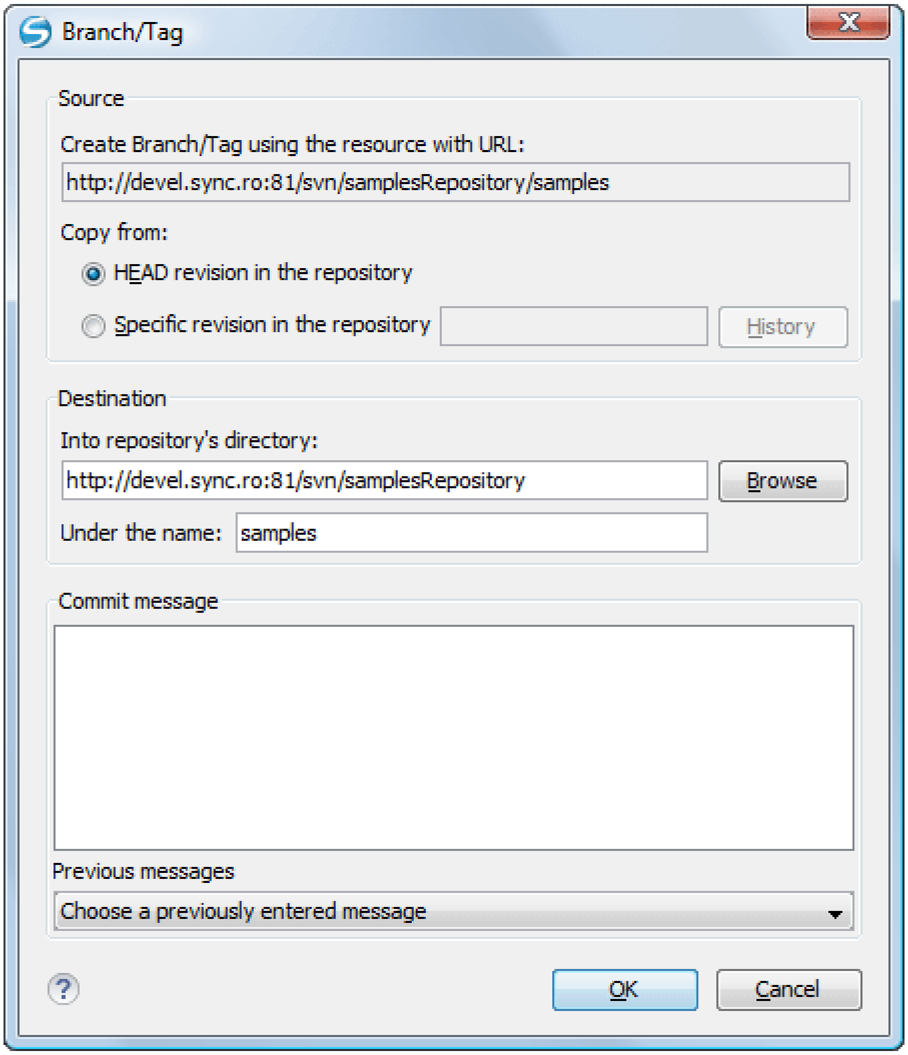
Another feature of version control systems is the ability to take a snapshot of a particular revision, so you can at any time recreate a certain build or environment. This is known as tagging. Tagging is especially useful when making release versions.

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

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

Create a Branch / Tag

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



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

Destination

Into repository's directory: http://devel.sync.ro:81/svn/samplesRepository

Under the name: samples

Commit message

Previous messages: Choose a previously entered message

Figure 15: The Branch / Tag dialog

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

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

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

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

Merging

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

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

The **Merge** action can be found in the **Tools** menu of Syncro SVN Client. The directory selected when you issued the command will be the result directory of the merge operation.

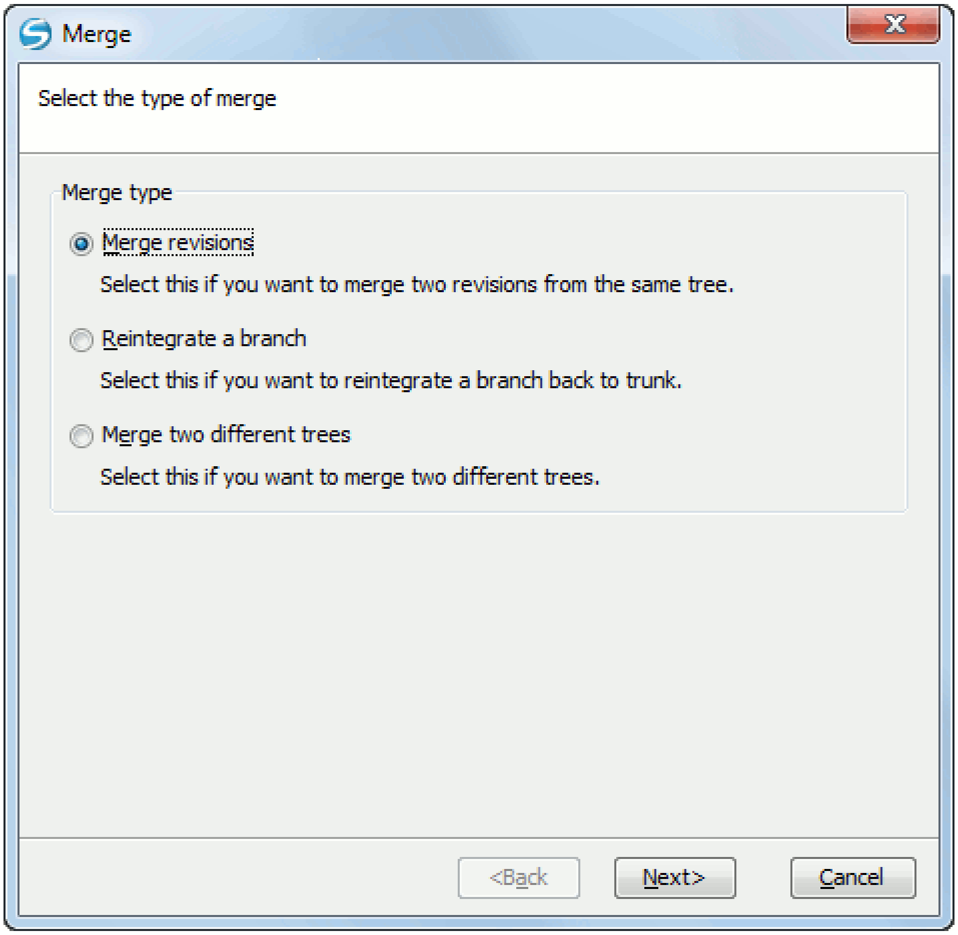
There are three common types of merging which are handled in different ways:

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

Merge Revisions

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

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:



Merge

Select the type of merge

Merge type

● Merge revisions

Select this if you want to merge two revisions from the same tree.

○ Reintegrate a branch

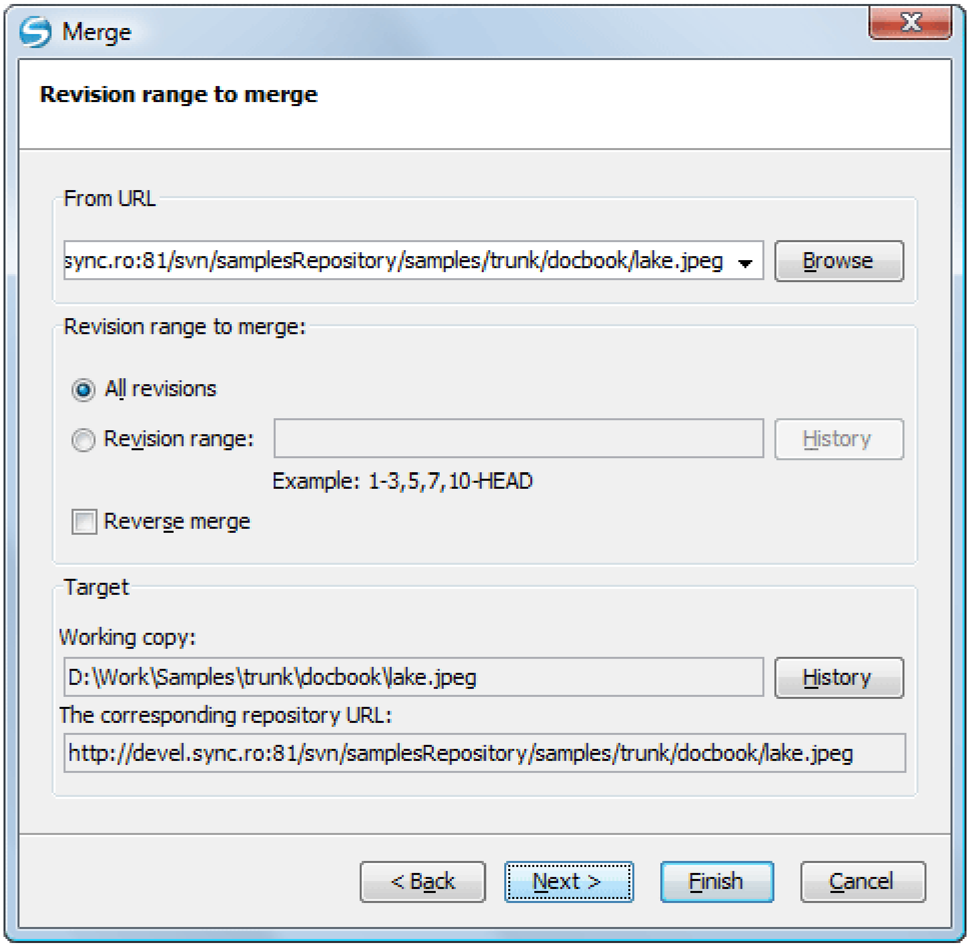
Select this if you want to reintegrate a branch back to trunk.

○ Merge two different trees

Select this if you want to merge two different trees.

Figure 16: The Merge Wizard - The Merge Type

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



Merge

Revision range to merge

From URL

sync.ro:81/svn/samplesRepository/samples/trunk/docbook/lake.jpeg

Revision range to merge:

● All revisions

○ Revision range:

Example: 1-3,5,7,10-HEAD

□ Reverse merge

Target

Working copy:

D:\Work\Samples\trunk\docbook\lake.jpeg

The corresponding repository URL:

http://devel.sync.ro:81/svn/samplesRepository/samples/trunk/docbook/lake.jpeg

Figure 17: The Merge Wizard - Revisions Range

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

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

5. Select the revision range.

   a) Choose between all revisions and a revision range.

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

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

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

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

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

6. Specify the target where the changes of the branch will be integrated.

   a) Enter the working copy folder in the **Working copy** field.

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

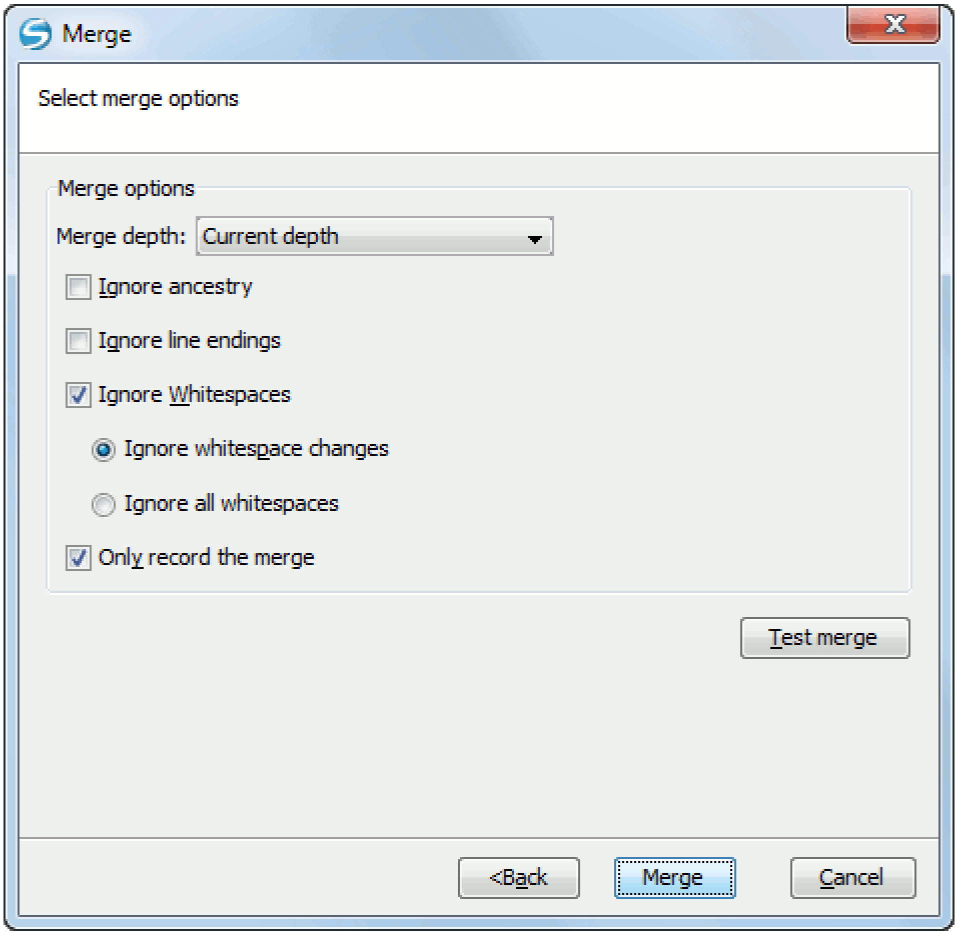
If you have already merged some changes from this branch and you remember the last merged revision, you can select that revision for the working copy using the **History** button. For example, if you have merged revisions 27 to 33 last time, then the start point for this merge operation should be revision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

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

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

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

7. Press the **Next** button.
The **Merge Options** step of the wizard is opened:



Merge

Select merge options

Merge options

Merge depth: Current depth

☐ Ignore ancestry

☐ Ignore line endings

☑ Ignore Whitespaces

● Ignore whitespace changes

○ Ignore all whitespaces

☑ Only record the merge

Figure 18: The Merge Wizard - Advanced Options

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

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

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

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

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

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

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

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

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

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

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

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

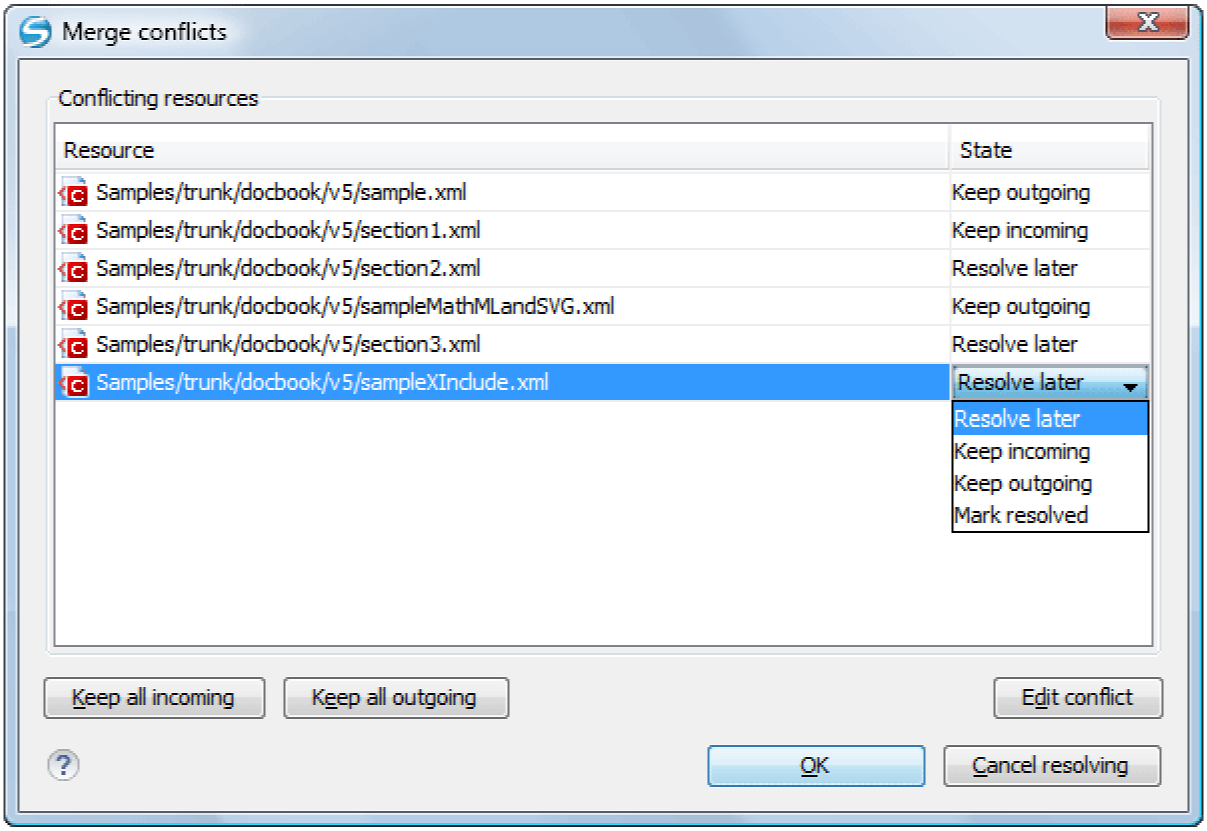
f) Press the **Test merge** button (optional).

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

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

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

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



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 19: Merge Conflicts Dialog

The options to resolve a conflict are:

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

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

Reintegrate a Branch

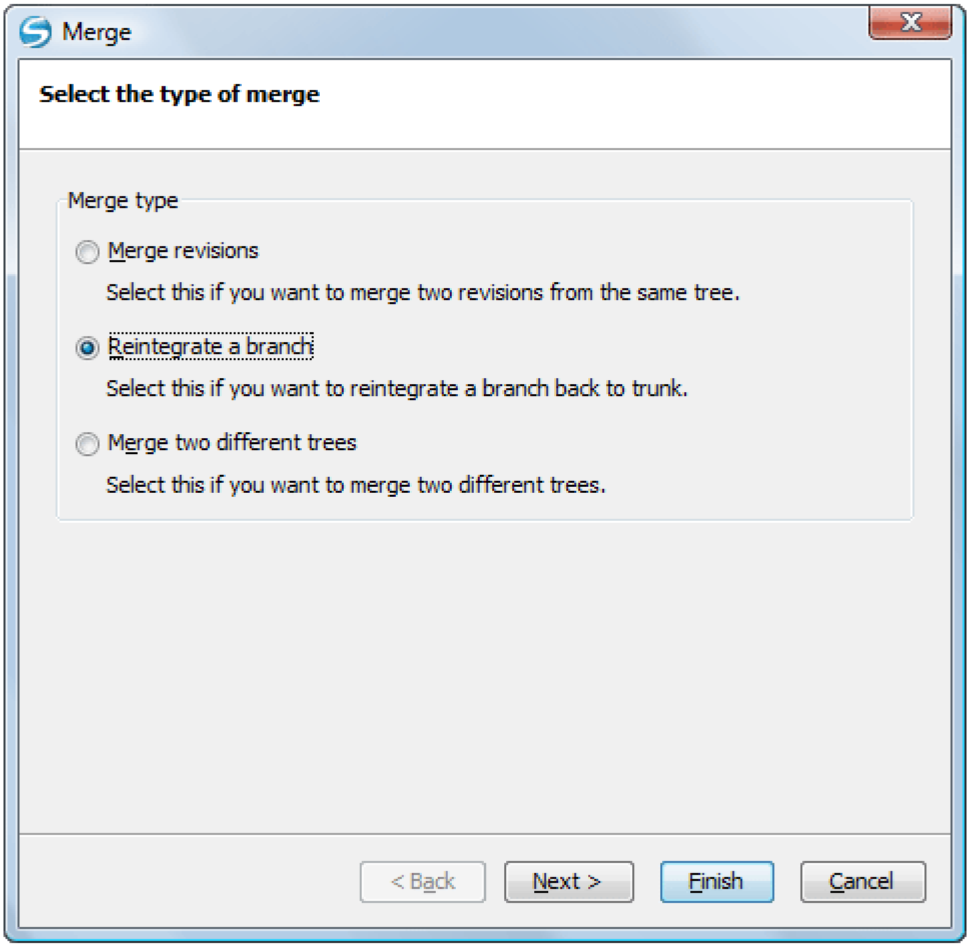
There are some conditions which apply to a reintegrate merge: Firstly, the server must support merge tracking. The working copy must be of depth infinite (no sparse checkouts), and it must not have any local modifications, switched items or items that have been updated to revisions other than HEAD. All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

- The server must support merge tracking.
- The working copy must be of depth infinite (no sparse checkouts), and it must not have any local modifications, switched items or items that have been updated to revisions other than HEAD.
- All changes to trunk made during branch development must have been merged across to the branch (or marked as having been merged).

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

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

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:



Select the type of merge

Merge type

○ Merge revisions

Select this if you want to merge two revisions from the same tree.

● Reintegrate a branch

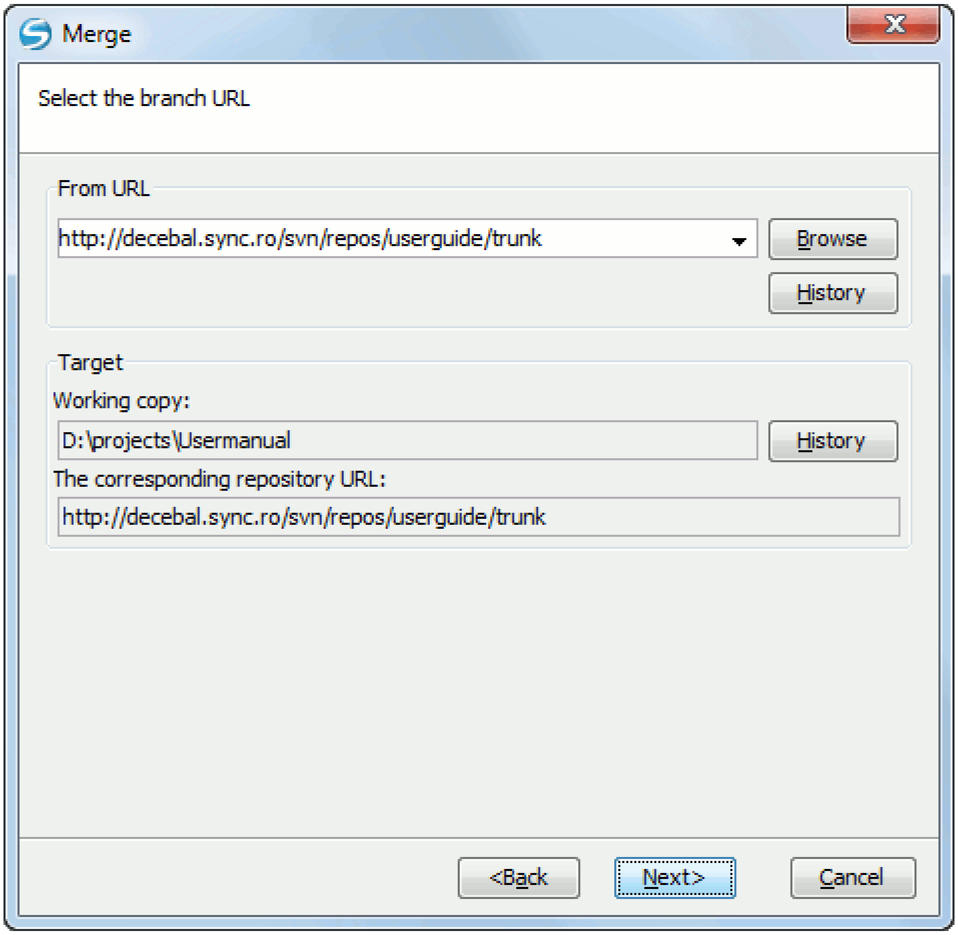
Select this if you want to reintegrate a branch back to trunk.

○ Merge two different trees

Select this if you want to merge two different trees.

Figure 20: The Merge Wizard - The Merge Type

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



Merge

Select the branch URL

From URL

http://decebal.sync.ro/svn/repos/userguide/trunk

Target

Working copy:

D:\projects\Usermanual

The corresponding repository URL:

http://decebal.sync.ro/svn/repos/userguide/trunk

Figure 21: The Merge Wizard - Reintegrate Branch

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

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

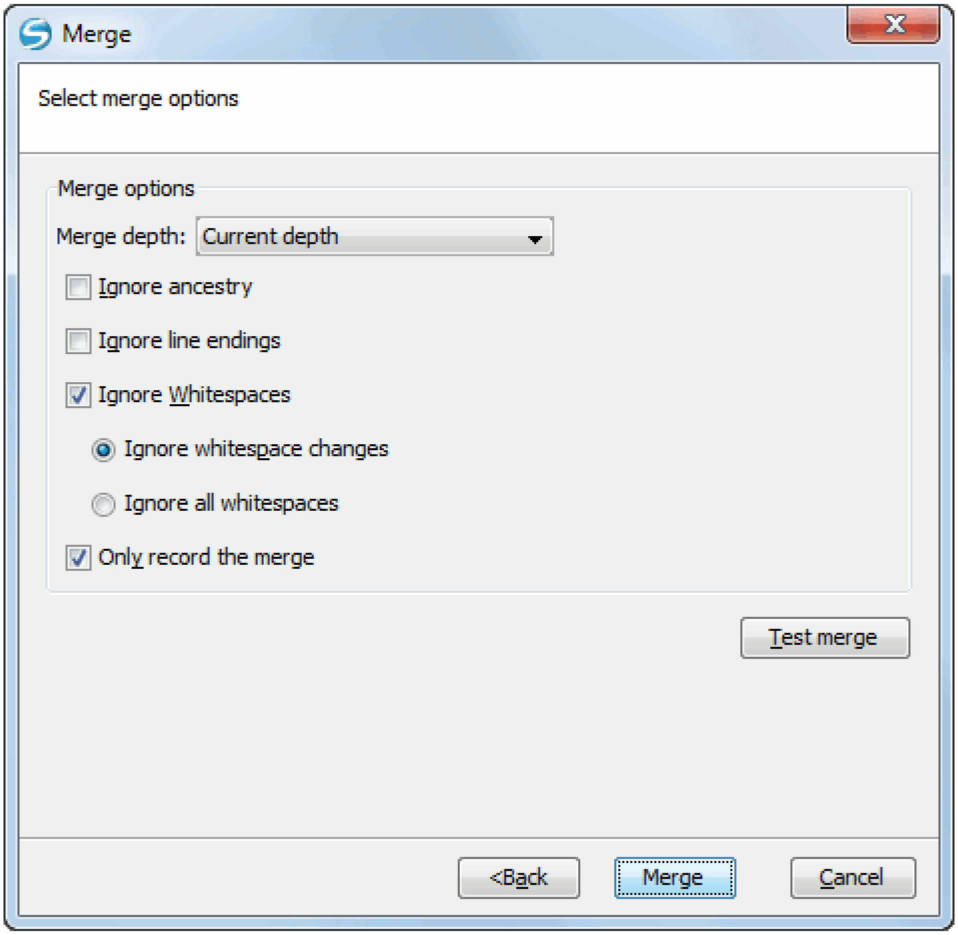
   The History button opens *the* ***History*** *dialog* which allows you to select a revision number of the repository with the changes.

5. Select the target of the operation.

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

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

6. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:



Select merge options

Merge options

Merge depth: Current depth

☐ Ignore ancestry

☐ Ignore line endings

☑ Ignore Whitespaces

● Ignore whitespace changes

○ Ignore all whitespaces

☑ Only record the merge

Figure 22: The Merge Wizard - Advanced Options

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

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

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

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

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

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

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

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

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

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

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

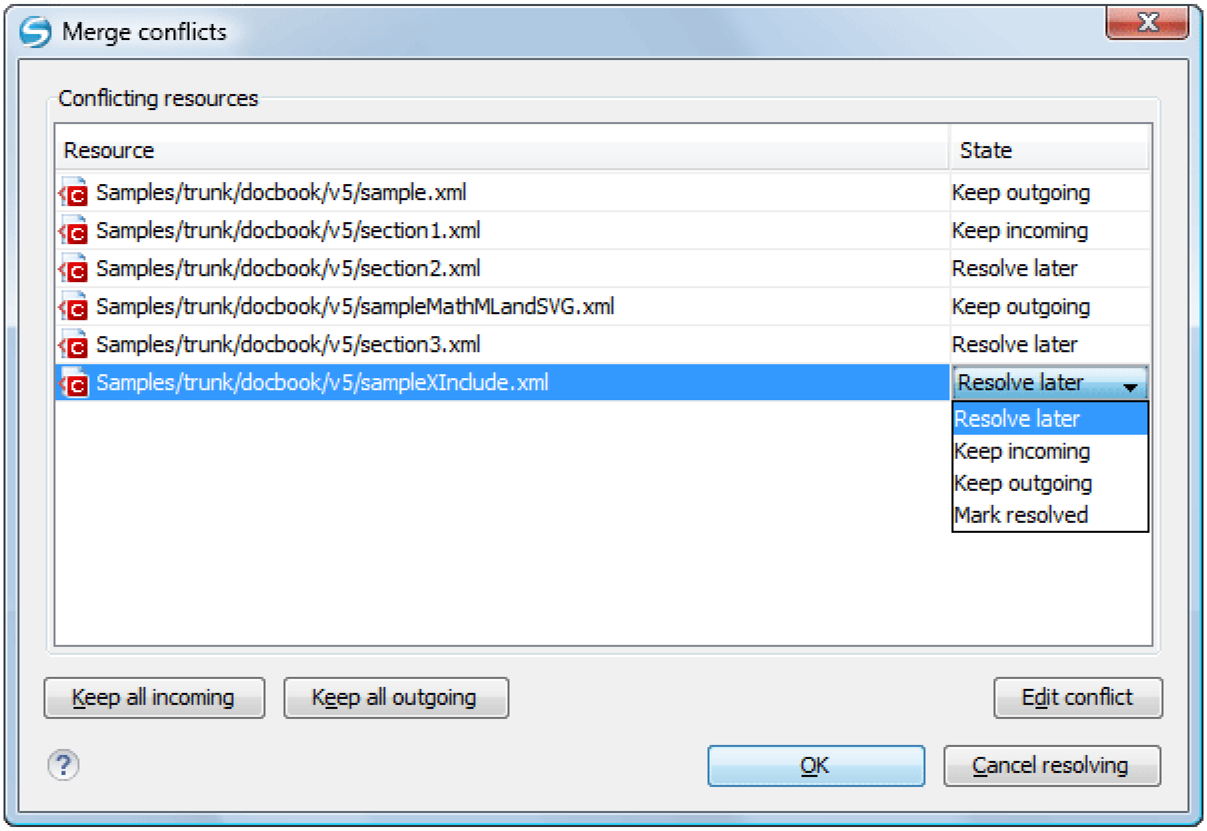
If you are using merge tracking support and you want to mark a revision as having been merged, without actually doing the merge here, check the **Only record the merge** checkbox. You might want to do this for two possible reasons. You make the changes by hand, then mark the change as merged so that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

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

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

8. Press the **Merge** button.
   The merge operation is executed.
9. Optionally resolve the conflicts that were created by the merge operation.

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



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 23: Merge Conflicts Dialog

The options to resolve a conflict are:

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

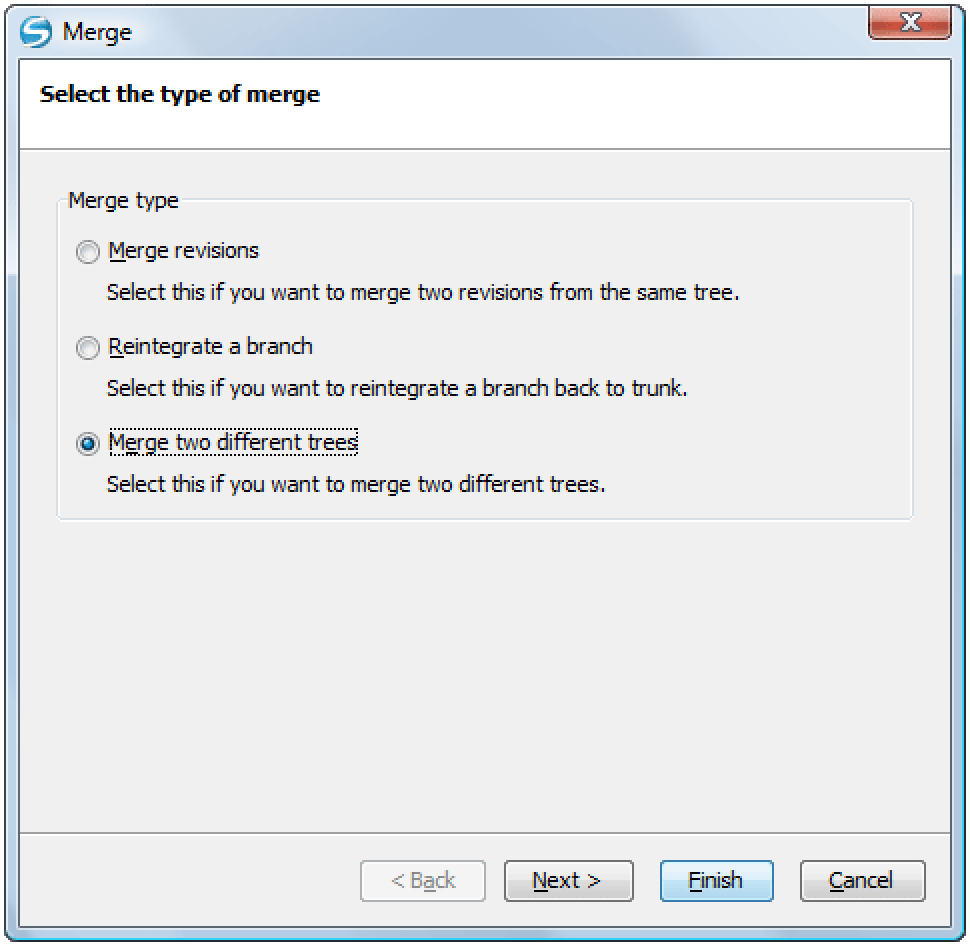
When the merge is completed it's a good idea to look at the result of the merge in the specified working copy and see if it meets your expectations. Because merging is sometimes complicated, when there are major changes, *conflicts may appear*.

Merge Two Different Trees

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

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

1. Go to menu **Tools** > **Merge ...**
   The **Merge** wizard is opened:



Merge

Select the type of merge

Merge type

○ Merge revisions

Select this if you want to merge two revisions from the same tree.

○ Reintegrate a branch

Select this if you want to reintegrate a branch back to trunk.

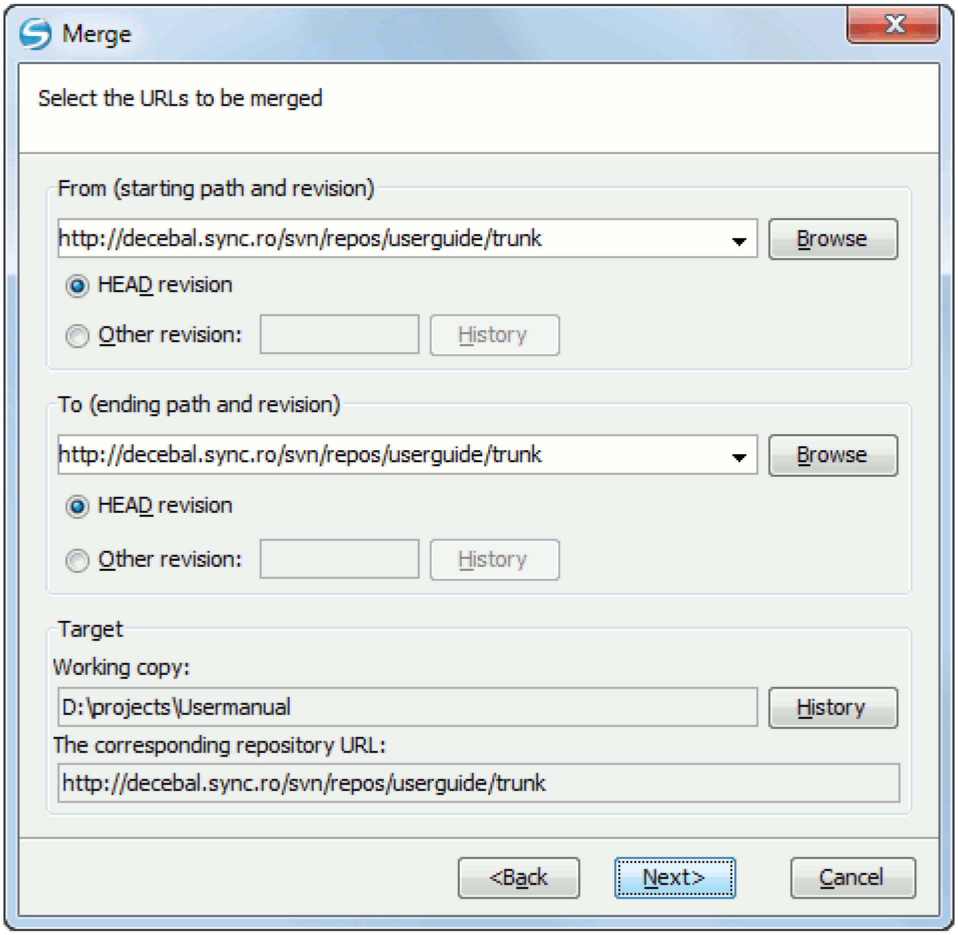
● Merge two different trees

Select this if you want to merge two different trees.

Figure 24: The Merge Wizard - The Merge Type

2. Select the option **Merge two different trees**.

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



Select the URLs to be merged

From (starting path and revision)

http://decebal.sync.ro/svn/repos/userguide/trunk

● HEAD revision

○ Other revision:

To (ending path and revision)

http://decebal.sync.ro/svn/repos/userguide/trunk

● HEAD revision

○ Other revision:

Target

Working copy:

D:\projects\Usermanual

The corresponding repository URL:

http://decebal.sync.ro/svn/repos/userguide/trunk

Figure 25: The Merge Wizard - Merge Two Different Trees

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

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

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

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

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

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

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

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

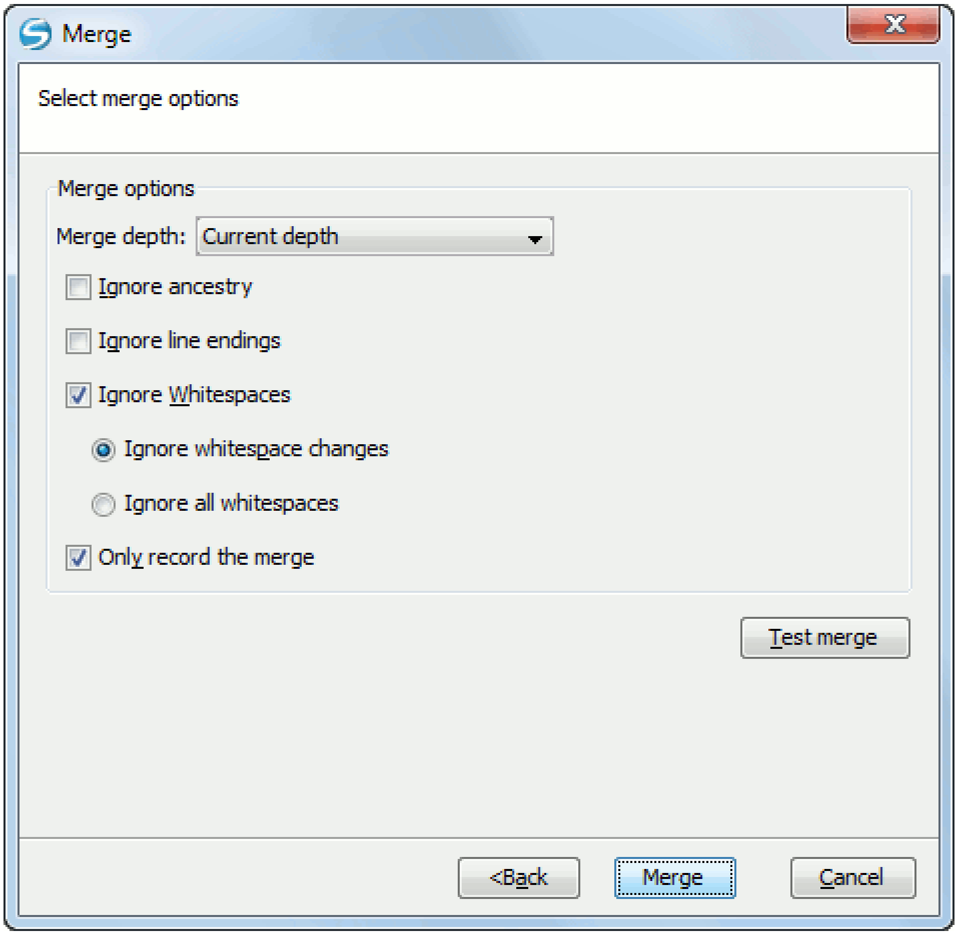
By default the HEAD revision is selected. If you want a previous revision you have to select the **Other revision** option and press *the **History** button* to see a list of all revisions.

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

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

   a) Specify the working copy path in the **Working copy** field.
   b) Specify the repository URL corresponding to the working copy.

7. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:



Merge depth: Current depth

□ Ignore ancestry

□ Ignore line endings

☑ Ignore Whitespaces

● Ignore whitespace changes

○ Ignore all whitespaces

☑ Only record the merge

Figure 26: The Merge Wizard - Advanced Options

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

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

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

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

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

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

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

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

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

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

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

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

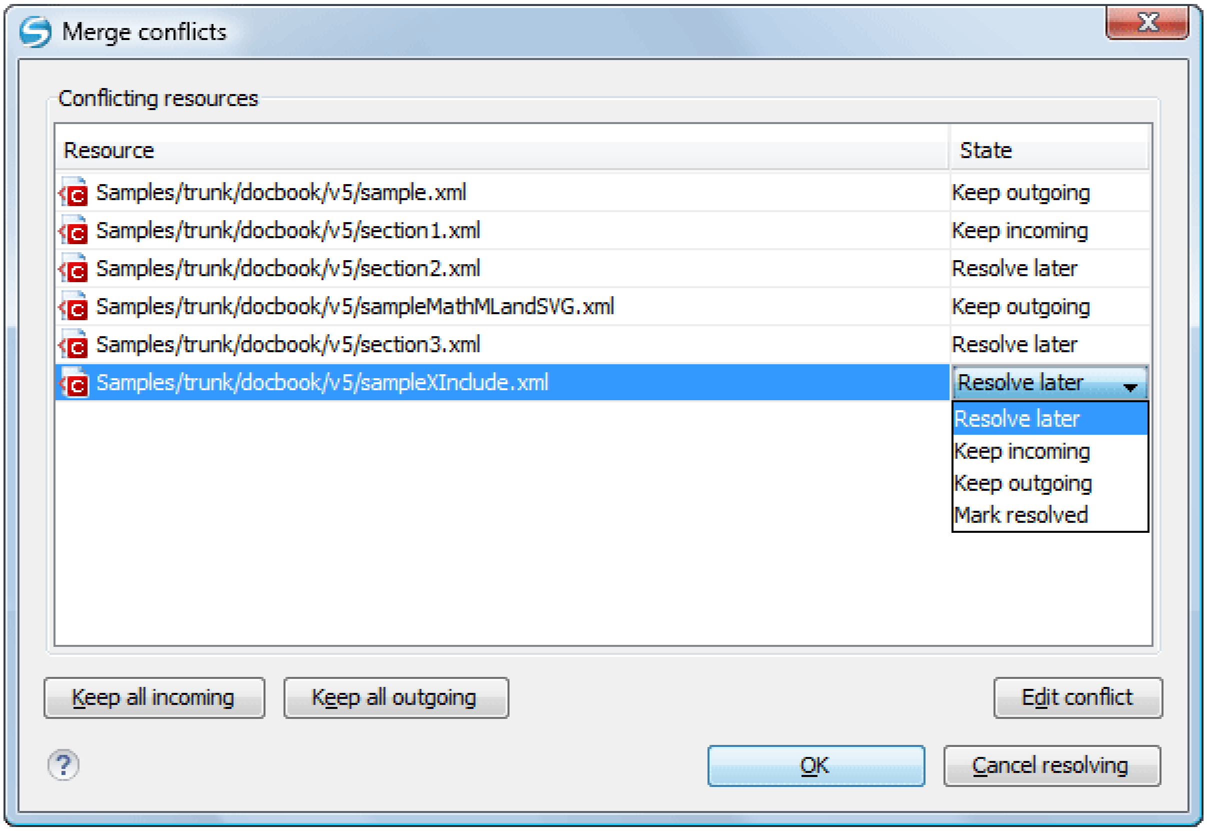
f) Press the **Test merge** button (optional).

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

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

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

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



Merge conflicts

Conflicting resources

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

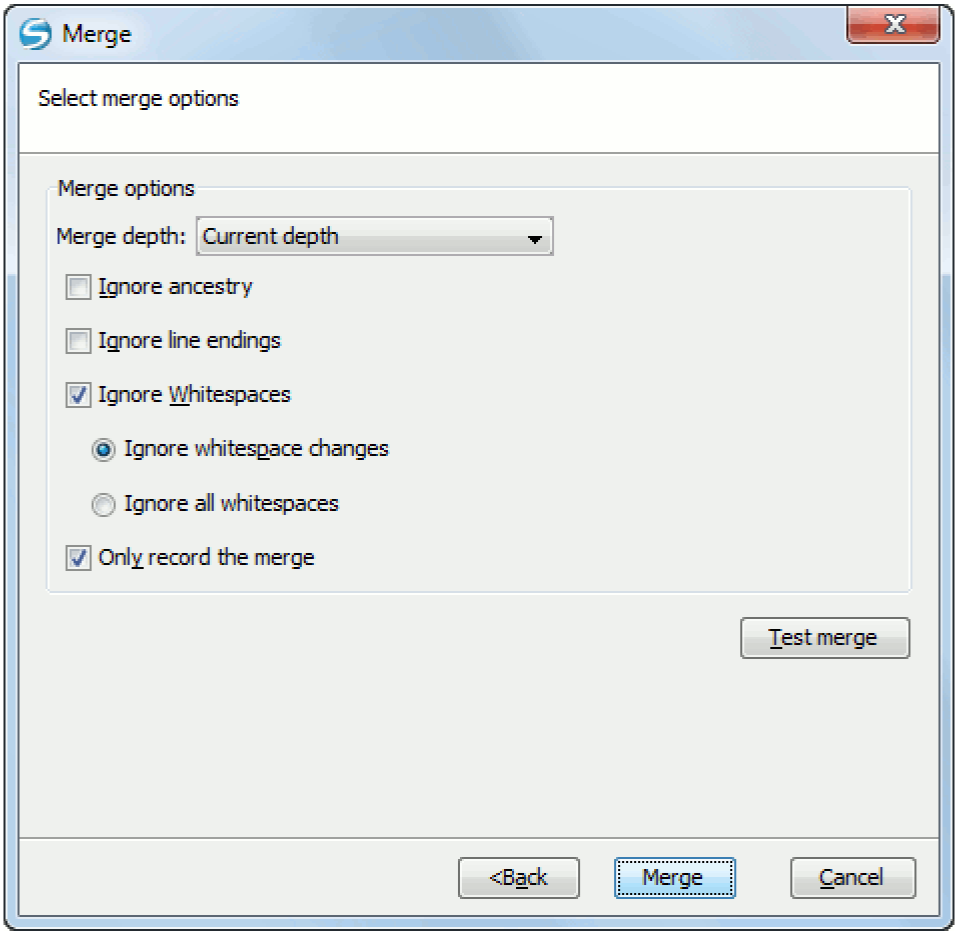
Figure 27: Merge Conflicts Dialog

The options to resolve a conflict are:

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

Merge Options

1. Press the **Next** button.
   The **Merge Options** step of the wizard is opened:



Merge

Select merge options

Merge options

Merge depth: Current depth

☐ Ignore ancestry

☐ Ignore line endings

☑ Ignore Whitespaces

● Ignore whitespace changes

○ Ignore all whitespaces

☑ Only record the merge

Figure 28: The Merge Wizard - Advanced Options

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

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

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

      The *depth* term is described in the *Sparse checkouts* section. The default depth is the depth of the current working copy.
   b) Check the **Ignore ancestry** checkbox (optional).

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

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

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

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

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

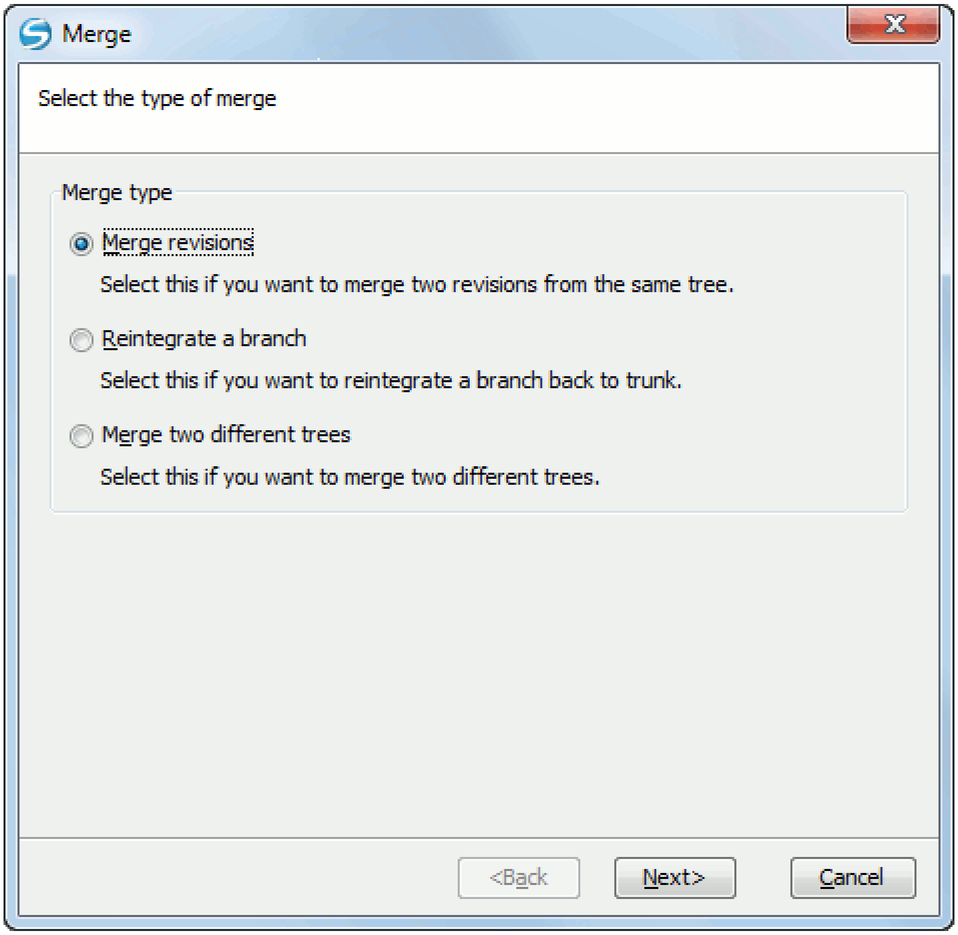
   By pressing the **Test merge** button you do a dry run of the merge operation in order to see what files are affected and how without modifying the working copy at all. This is very helpful in detecting where conflicts may occur.

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

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

Merge Branches Task - First Step

Go to menu **Tools** > **Merge ...**
The **Merge** wizard is opened:



Select the type of merge

Merge type

● Merge revisions

Select this if you want to merge two revisions from the same tree.

○ Reintegrate a branch

Select this if you want to reintegrate a branch back to trunk.

○ Merge two different trees

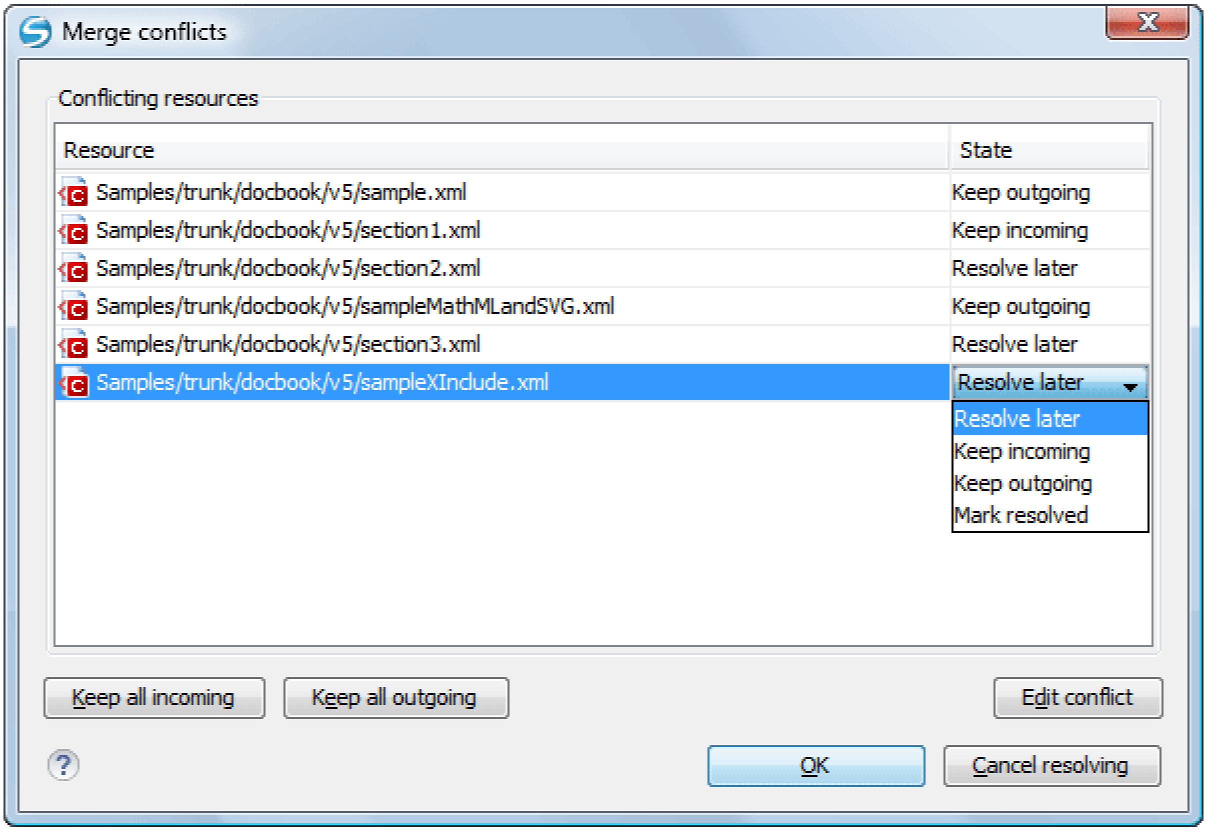
Select this if you want to merge two different trees.

Figure 29: The Merge Wizard - The Merge Type

Resolve Merge Conflicts

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

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



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 30: Merge Conflicts Dialog

The options to resolve a conflict are:

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

Switch the Repository Location

The **Switch** action is useful when the repository location of a working copy or only of a versioned item of the working copy must be changed within the same repository. The action is available on the **Tools** menu when a versioned resource is selected in the current working copy that is displayed in *the **Working Copy** view*.

Relocate a Working Copy

Sometimes the base URL of the repository is changed after a working copy is checked out from that URL, for example the repository itself is relocated to a different server. In such cases you do not have to check out again a working copy from the new repository location. It is easier to change the base URL of the root folder of the working copy to the new URL of the repository. This action is called **Relocate** and is available on the **Tools** menu when the selected item in *the **Working Copy** view* is a versioned folder.

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

Patches

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

What Is a Patch

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

Creating a patch in Subversion implies the access to two states (revisions) of a project:

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

Create a Patch Between Working Copy and Repository Revision

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

1. Go to menu **Tools** > **Create patch** .
   This opens the **Create patch** wizard:

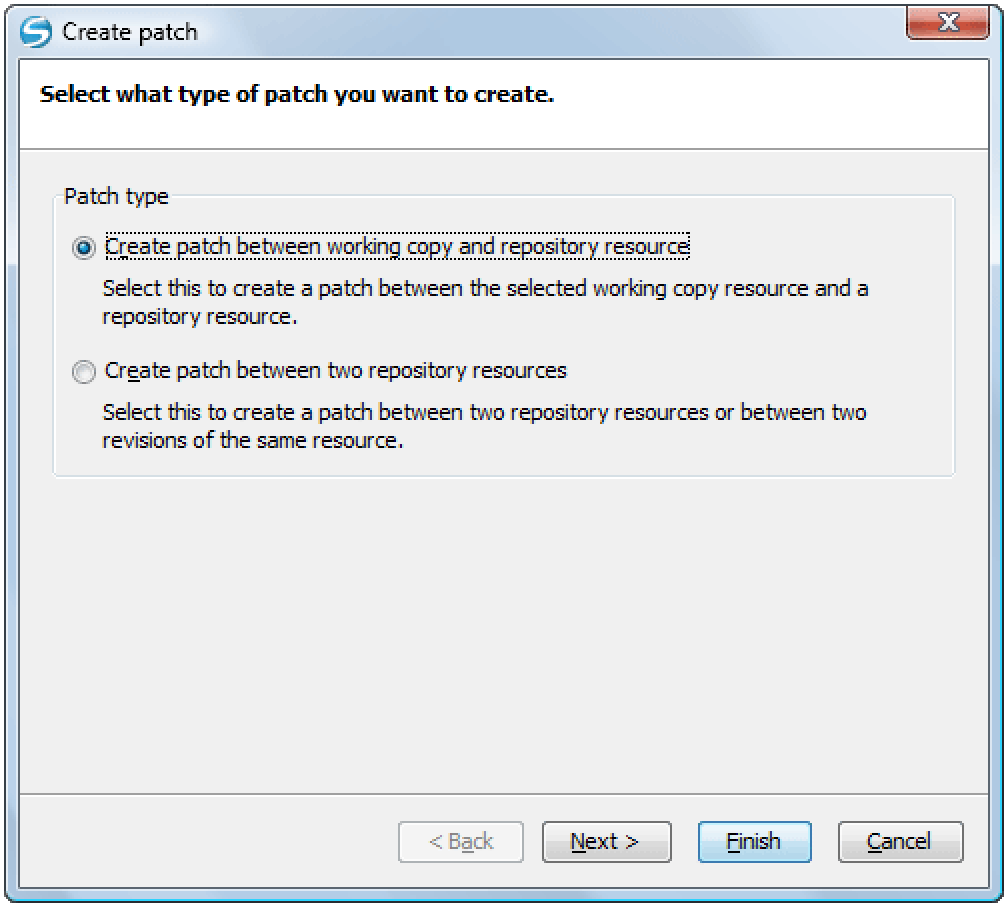
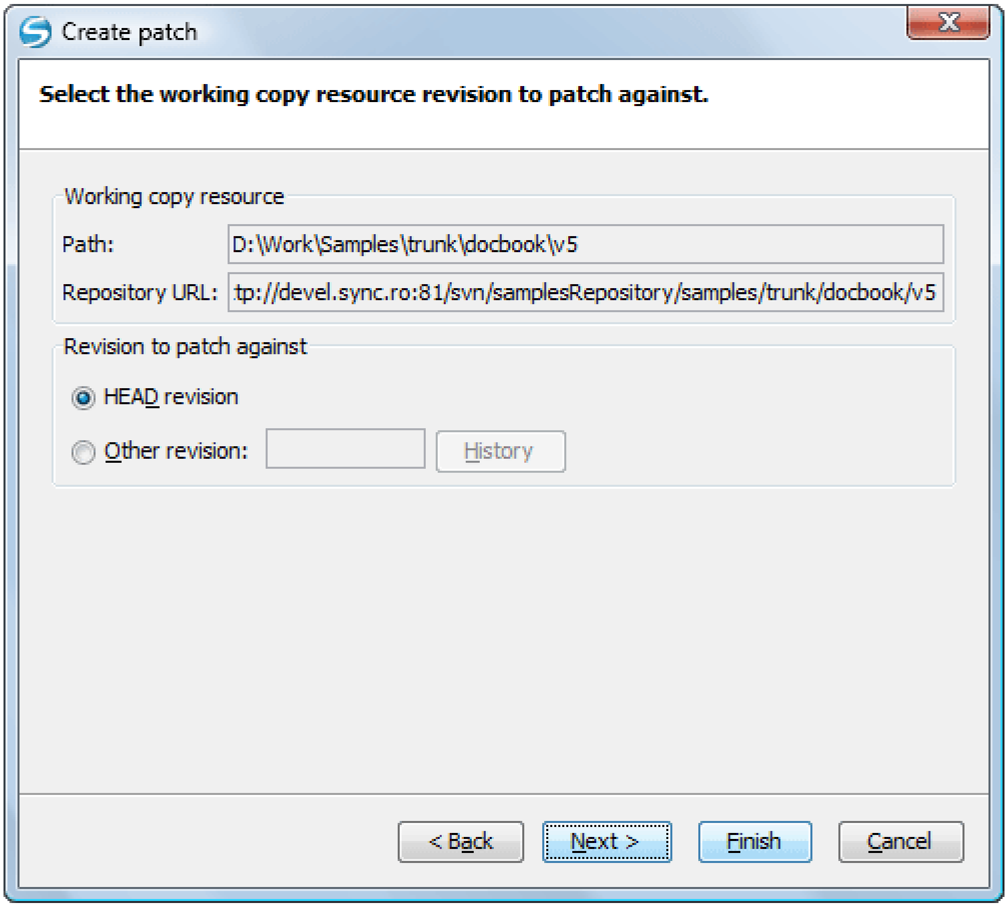

Create patch
Select what type of patch you want to create.
Patch type
Create patch between working copy and repository resource
Select this to create a patch between the selected working copy resource and a repository resource.
Create patch between two repository resources
Select this to create a patch between two repository resources or between two revisions of the same resource.
< Back
Next >
Finish
Cancel

Figure 31: The Create Patch Wizard - Patch Type

2. Select the first option in the dialog.
3. Press the **Next** button.

The second step of the wizard is opened:



Create patch

Select the working copy resource revision to patch against.

Working copy resource

Path: D:\Work\Samples\trunk\docbook\v5

Repository URL: tp://devel.sync.ro:81/svn/samplesRepository/samples/trunk/docbook/v5

Revision to patch against

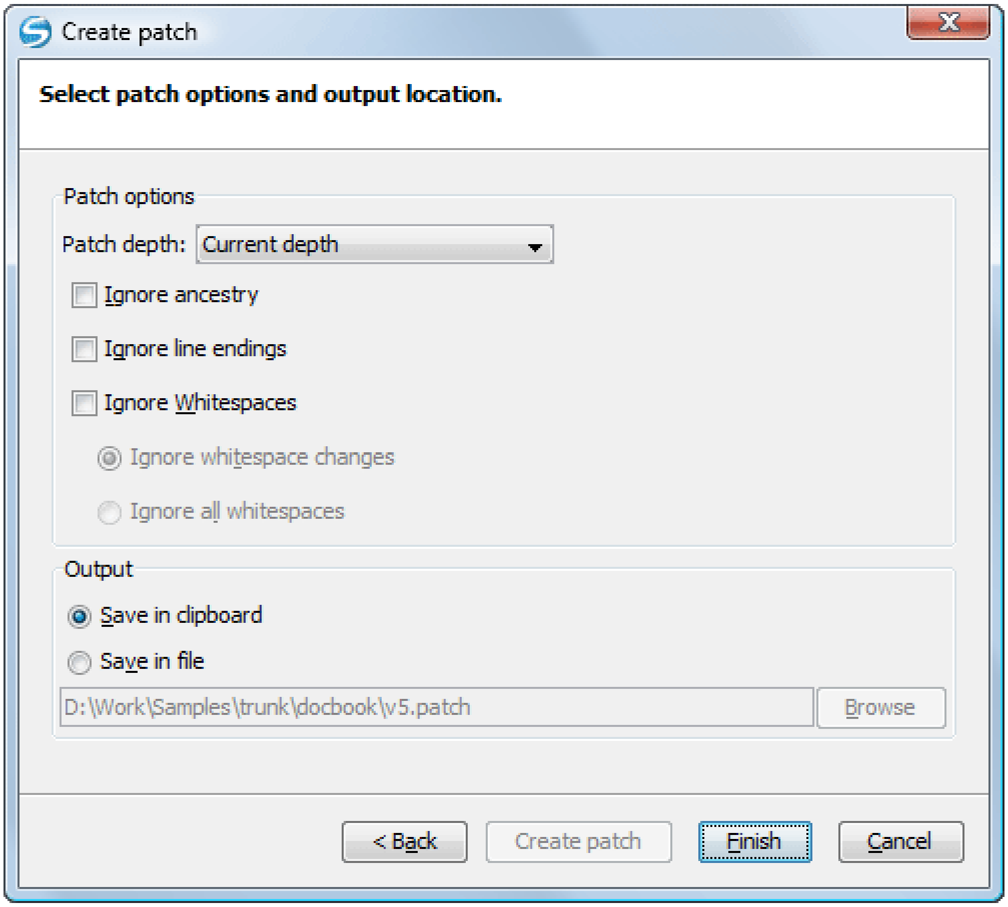
● HEAD revision

○ Other revision:

Figure 32: The Create Patch Wizard - Step 2

4. Specify the working copy resource.
   a) Specify the local file path of the working copy resource.
   b) Specify the repository URL corresponding to the working copy resource.
5. Select the repository revision.

   You have the option of choosing between the HEAD revision and a specific revision number. For the second option you should press *the **History** button* to display a list of the repository revisions.
6. Press the **Next** button.
   The next step of the wizard is displayed:



Create patch

Select patch options and output location.

Patch options

Patch depth: Current depth

☐ Ignore ancestry

☐ Ignore line endings

☐ Ignore Whitespaces

● Ignore whitespace changes

○ Ignore all whitespaces

Output

● Save in clipboard

○ Save in file

D:\Work\Samples\trunk\docbook\v5.patch

Figure 33: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

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

   - **Current depth** - The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
   - **Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.
   - **Immediate children (immediates)** - The patch is created only on the child files and folders without going in subfolders.
   - **File children only (files)** - The patch is created only on the child files.
   - **This folder only (empty)** - The patch is created only on the selected folder (that is no child file or folder is included in the patch).

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

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

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

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

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

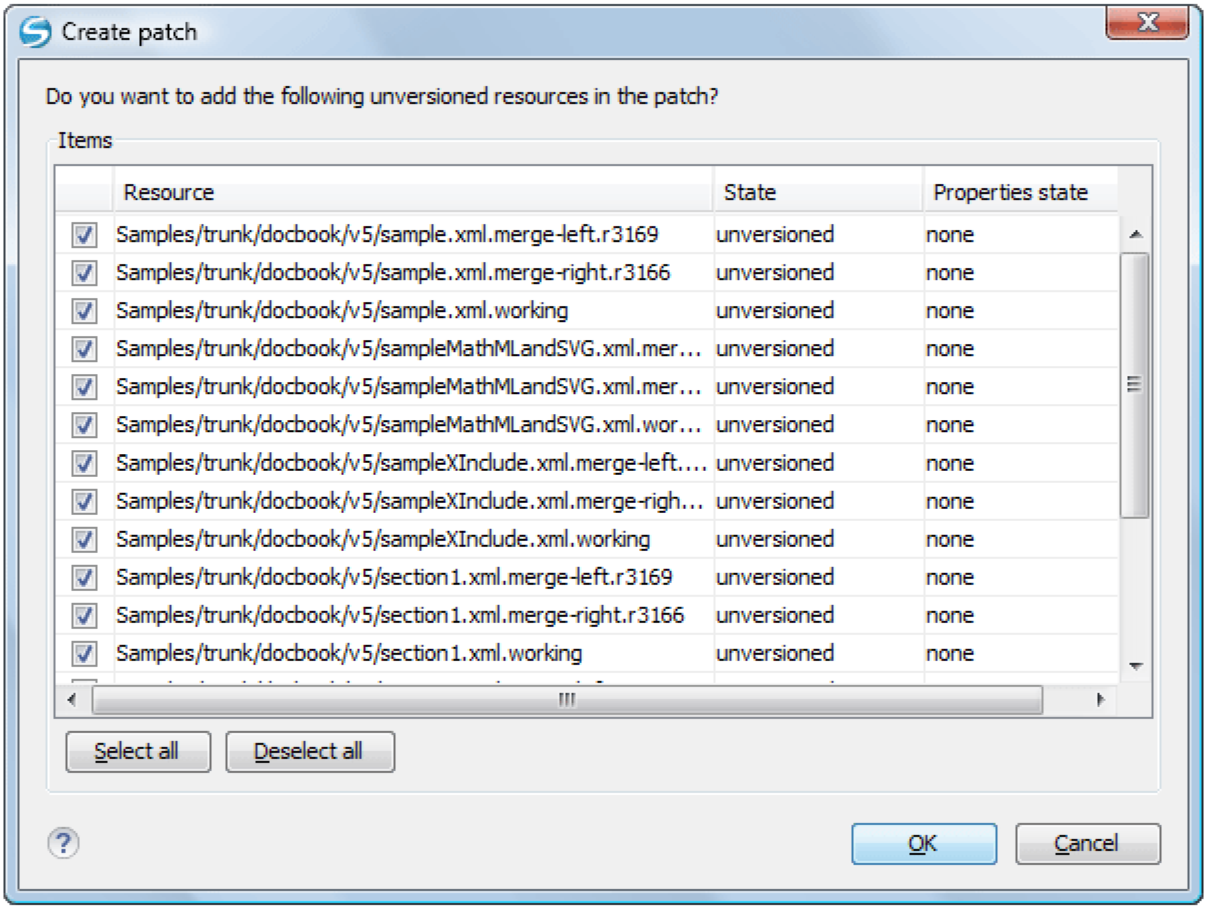
    If checked, the differences in whitespaces are ignored when the patch is created.

11. Select the output location.

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

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

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



Create 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 34: The Patch Wizard - Add Unversioned Resources

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

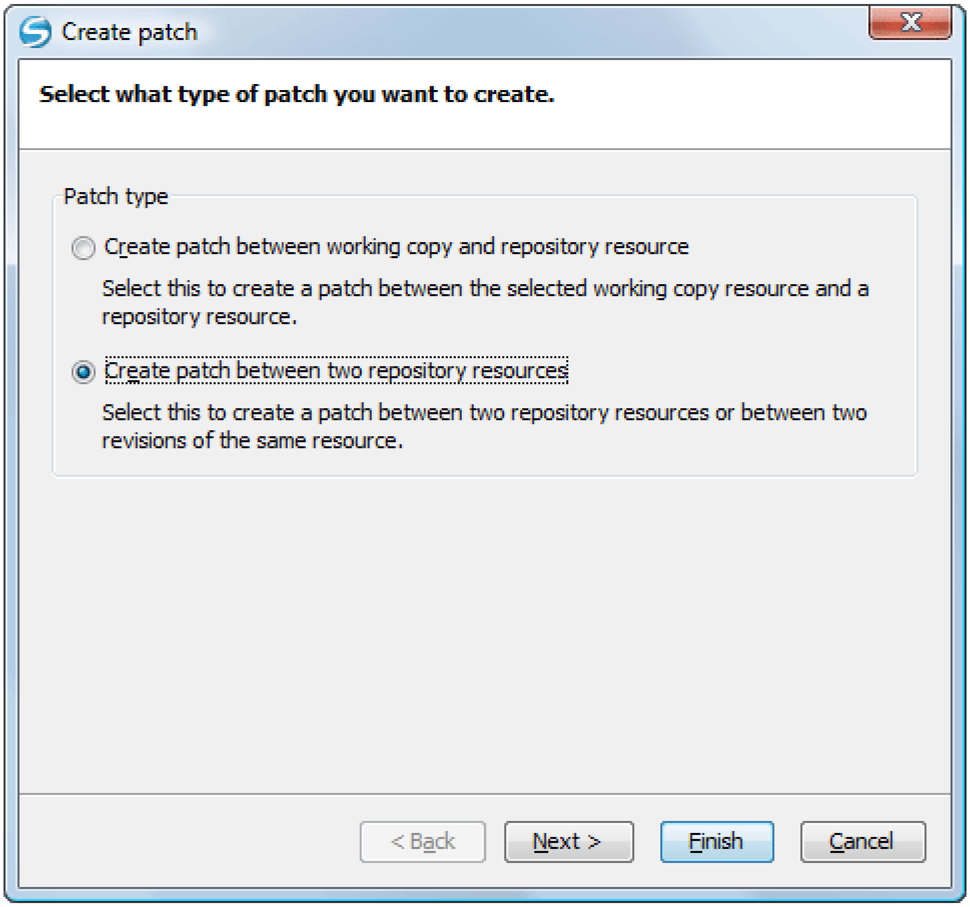
If the patch is applied on a folder of the working copy and that folder contains unversioned files this step of the wizard offers the option of selecting the ones that will be included in the patch.

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

Create a Patch Between Two Repository Revisions

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

1. Go to menu **Tools** > **Create patch** .
This opens the **Create patch** wizard:



Create patch

Select what type of patch you want to create.

Patch type

○ Create patch between working copy and repository resource

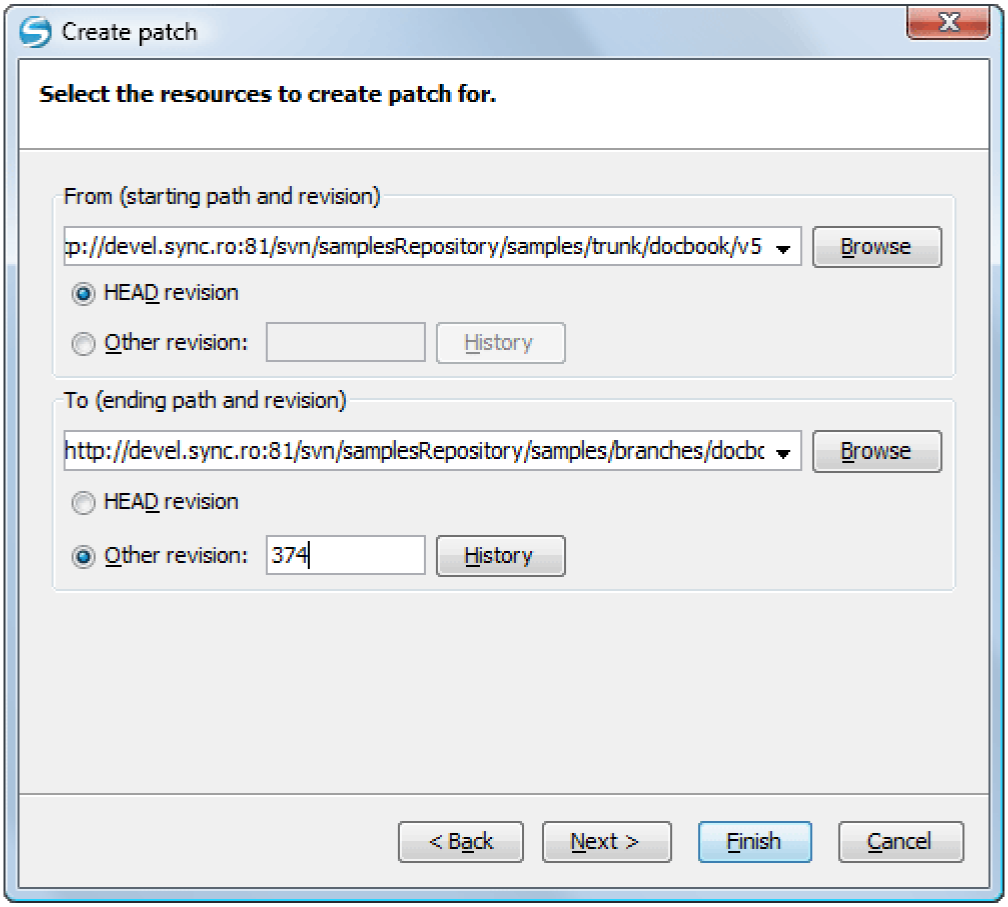
Select this to create a patch between the selected working copy resource and a repository resource.

● Create patch between two repository resources

Select this to create a patch between two repository resources or between two revisions of the same resource.

Figure 35: The Create Patch Wizard - Patch Type

2. Select the second option in the dialog.
3. Press the **Next** button.
   The second step of the wizard is opened:



Create patch

Select the resources to create patch for.

From (starting path and revision)

p://devel.sync.ro:81/svn/samplesRepository/samples/trunk/docbook/v5

● HEAD revision

○ Other revision:

To (ending path and revision)

http://devel.sync.ro:81/svn/samplesRepository/samples/branches/docbc

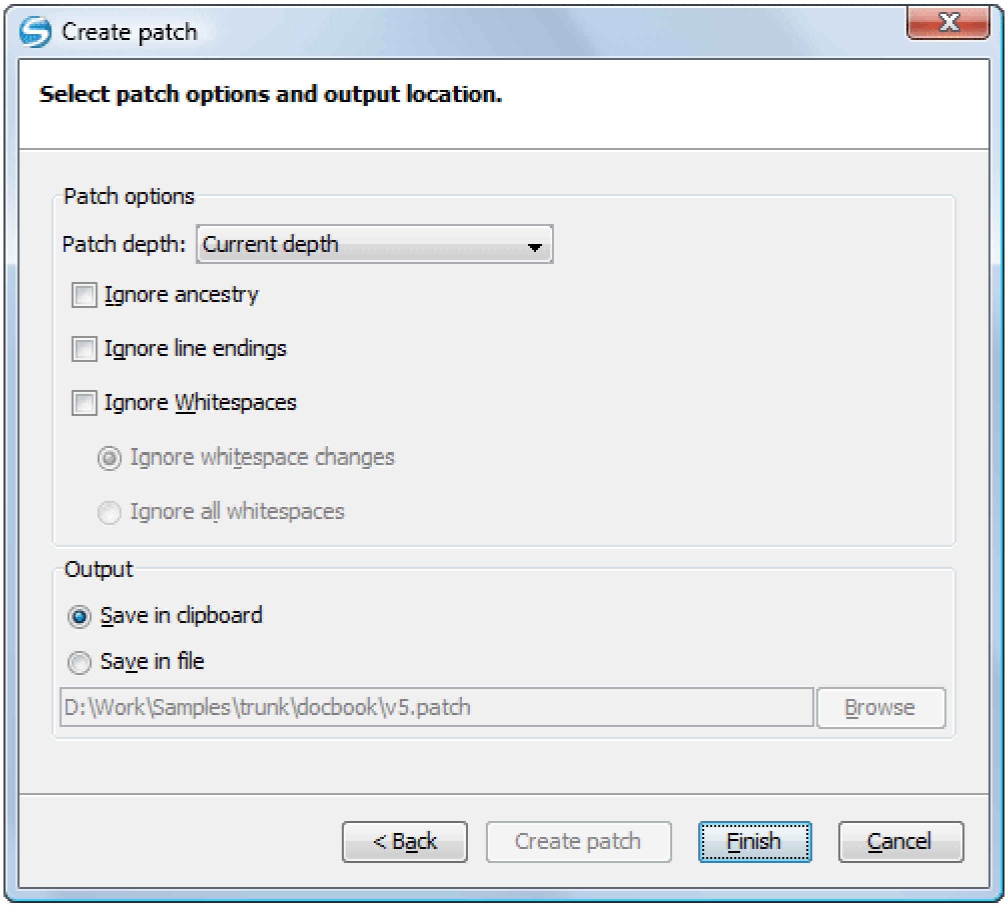
○ HEAD revision

● Other revision: 374

Figure 36: The Create Patch Wizard - Step 2

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

   For both revisions you have the option of choosing between the HEAD revision and a specific revision number. For a specific number you should press *the* ***History*** *button* to display a list of the repository revisions.
6. Press the **Next** button.
   The next step of the wizard is displayed:



Create patch

Select patch options and output location.

Patch options

Patch depth: Current depth

□ Ignore ancestry

□ Ignore line endings

□ Ignore Whitespaces

● Ignore whitespace changes

○ Ignore all whitespaces

Output

● Save in clipboard

○ Save in file

D:\Work\Samples\trunk\docbook\v5.patch

Figure 37: The Create Patch Wizard - Options

7. Select the depth of the patch operation.

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

   - **Current depth** - The depth of going into the folder for creating the patch is the same as the depth of that folder in the working copy.
   - **Recursive (infinity)** - The patch is created on all the files and folders contained in the selected folder.
   - **Immediate children (immediates)** - The patch is created only on the child files and folders without going in subfolders.
   - **File children only (files)** - The patch is created only on the child files.
   - **This folder only (empty)** - The patch is created only on the selected folder (that is no child file or folder is included in the patch).

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

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

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

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

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

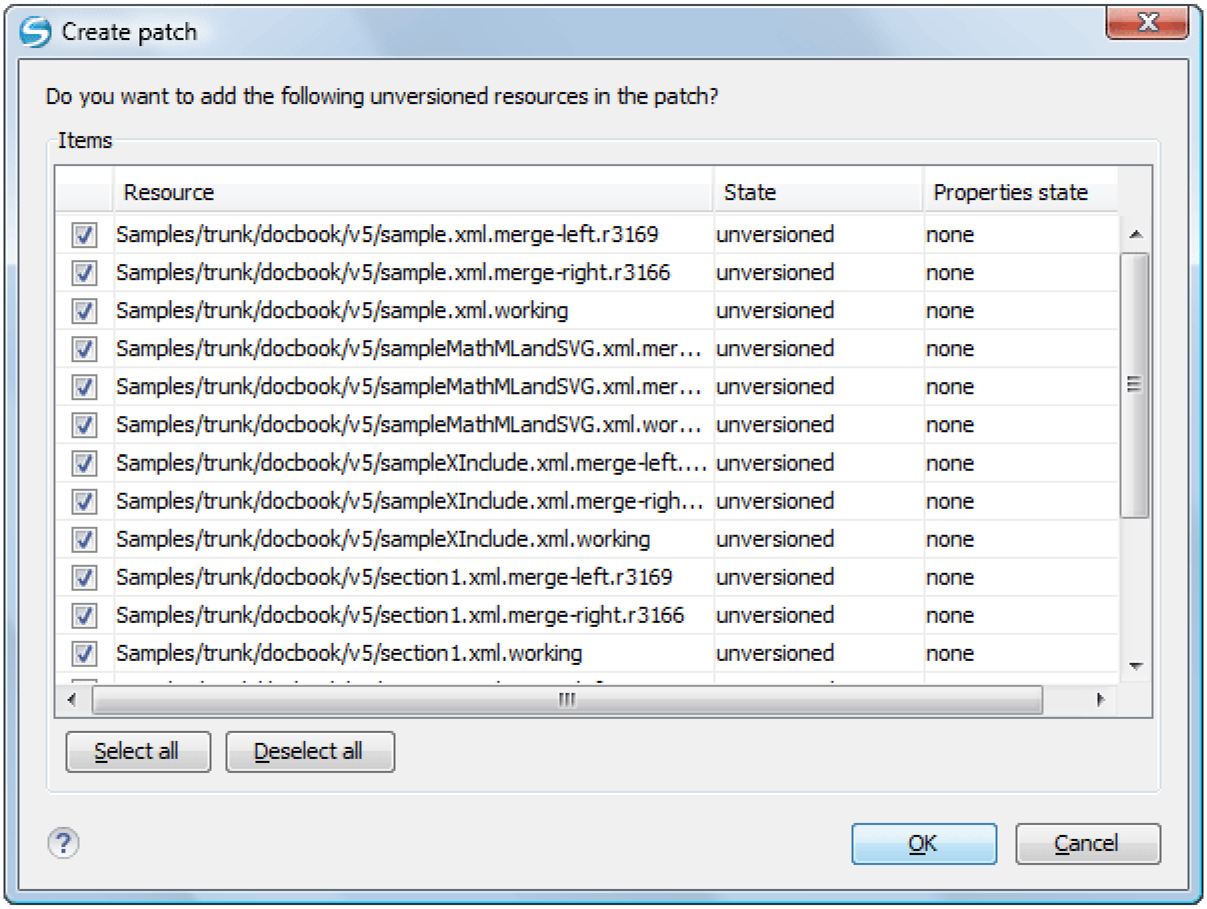
    If checked, the differences in whitespaces are ignored when the patch is created.

11. Select the output location.

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

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

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



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

	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 38: The Patch Wizard - Add Unversioned Resources

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

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

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

Working with Repositories

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

Importing Resources Into a Repository

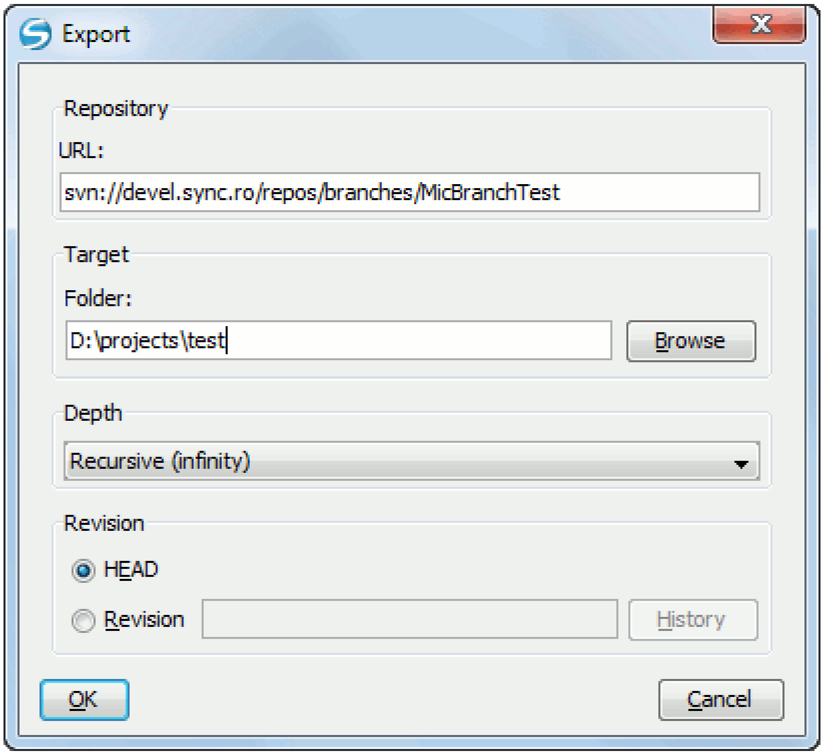
This is the process of taking a project and importing it into a repository so that it can be managed by a Subversion server. If you have already been using Subversion and you have an existing working copy you want to use, then you will likely want to follow the procedure for *using an existing working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

This process is started from menu **Repository** > **Import** > **Import Folder Content** . The same action is available in the **Repositories** view contextual menu. A dialog will ask you to select a directory that will be imported into the selected repository location. The complete directory tree will be imported into the repository including all files. The name of the imported folder will not appear in the repository, but only the contents of the folder will.

Exporting Resources From a Repository

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

The export dialog is very similar to the check out dialog:



Repository

URL: svn://devel.sync.ro/repos/branches/MicBranchTest

Target

Folder: D:\projects\test

Depth: Recursive (infinity)

Revision

● HEAD

○ Revision

Figure 39: Export from Repository

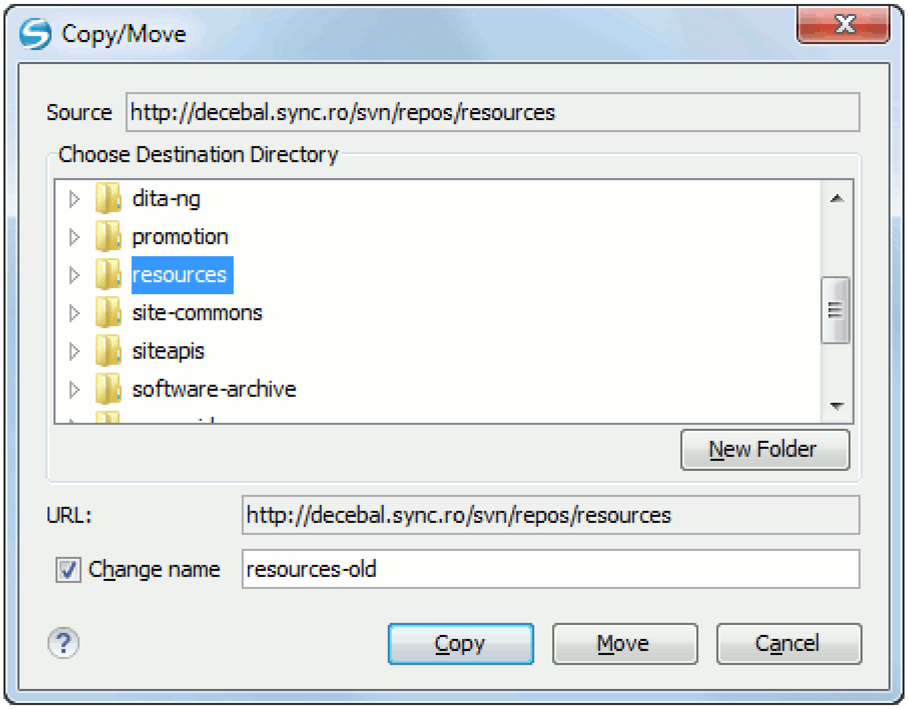
You can choose the target directory from the file system by pressing the **Browse** button. If you need to export a specific revision, you can select the **Revision** radio button and then click on the **History** button and choose a revision from the new dialog. Or you could simply type the revision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

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

Copy / Move / Delete Resources From a Repository

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

- The **Copy/Move** action allows you to copy and move individual or multiple resources. After invoking the action the **Copy/Move** dialog will pop up. The dialog displays the path of the resource that is copied and the tree structure of the repository allowing you to choose the destination directory. The path of this target directory will be presented in the text field **URL**. If you choose to copy a single resource then an additional checkbox called **Change name** and a text field allow you to choose the new name of the copied/moved resource.



Source: http://decebal.sync.ro/svn/repos/resources

Choose Destination Directory

- dita-ng
- promotion
- resources
- site-commons
- siteapis
- software-archive

URL: http://decebal.sync.ro/svn/repos/resources

☑ Change name: resources-old

Figure 40: Copy/Move Resources on a Repository

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

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

Sparse Checkout

Sometimes you need to check out only certain parts of a directory tree. For this you can checkout the top folder (*the action* ***Check Out*** *from the* ***Repositories*** *view*) and then update recursively only the needed directories (*the action* ***Update*** *from the* ***Working Copy*** *view*). Each directory now understands the notion of depth which has four possible values:

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

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

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

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

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

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

Syncro SVN Client Views

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

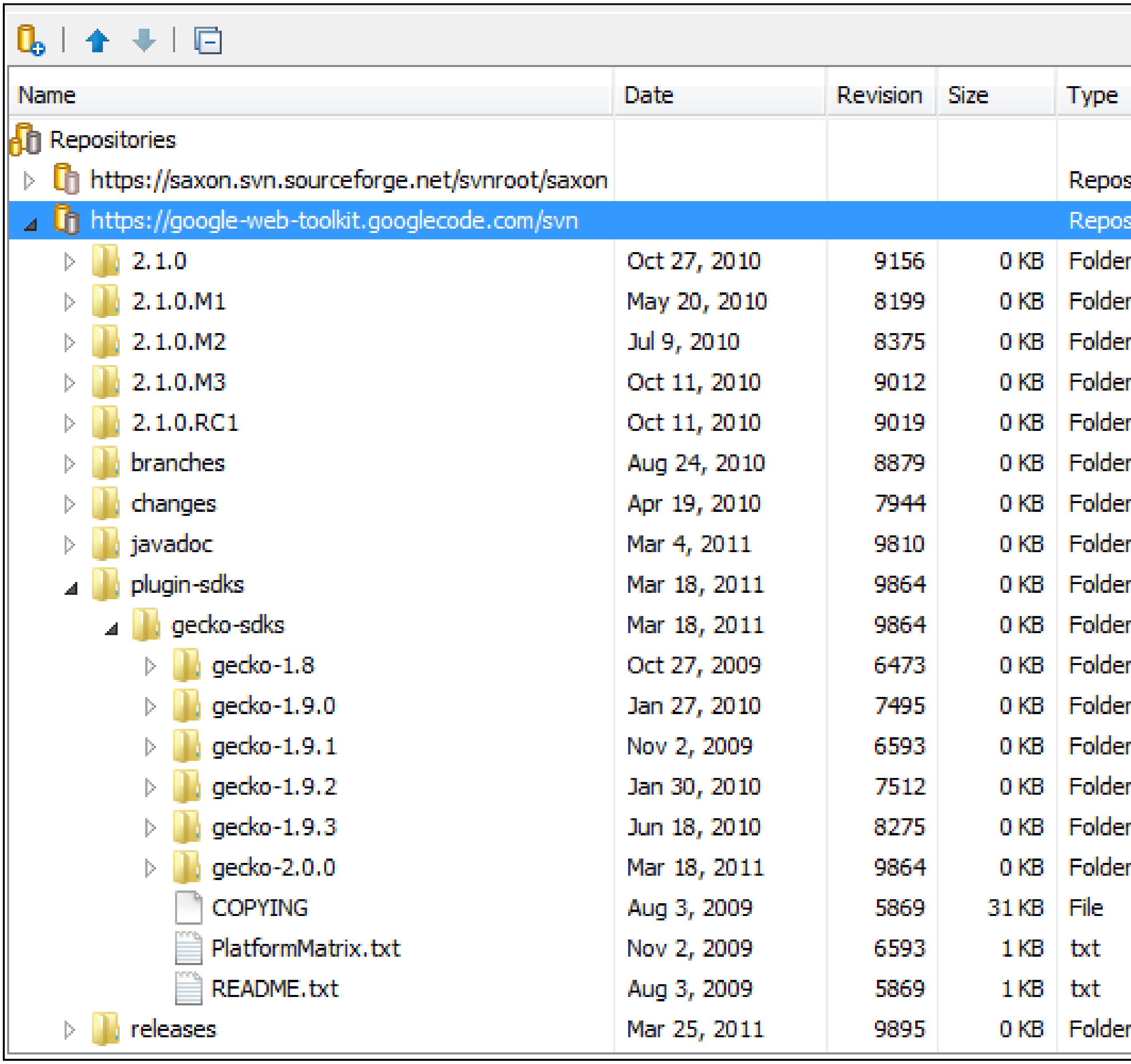
- *Working Copy View*
- *History View*
- *Console View*

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 Subversion repository locations and browse repositories. Repository files and folders are presented in a tree view with the repository locations at the first level, where each location represents a connection to a specific repository. More information about each resource is displayed in a tabular form:

- **Date** - Date when the resource was last modified;
- **Revision** - The revision number at which the resource was last time modified;
- **Author** - Name of the person who made the last modification on the resource;
- **Size** - Resource size on disk;
- **Type** - Contains the resource type or file extension.



Name	Date	Revision	Size	Type
Repositories				
https://saxon.svn.sourceforge.net/svnroot/saxon				Repos
https://google-web-toolkit.googlecode.com/svn				Repos
2.1.0	Oct 27, 2010	9156	0 KB	Folder
2.1.0.M1	May 20, 2010	8199	0 KB	Folder
2.1.0.M2	Jul 9, 2010	8375	0 KB	Folder
2.1.0.M3	Oct 11, 2010	9012	0 KB	Folder
2.1.0.RC1	Oct 11, 2010	9019	0 KB	Folder
branches	Aug 24, 2010	8879	0 KB	Folder
changes	Apr 19, 2010	7944	0 KB	Folder
javadoc	Mar 4, 2011	9810	0 KB	Folder
plugin-sdks	Mar 18, 2011	9864	0 KB	Folder
gecko-sdks	Mar 18, 2011	9864	0 KB	Folder
gecko-1.8	Oct 27, 2009	6473	0 KB	Folder
gecko-1.9.0	Jan 27, 2010	7495	0 KB	Folder
gecko-1.9.1	Nov 2, 2009	6593	0 KB	Folder
gecko-1.9.2	Jan 30, 2010	7512	0 KB	Folder
gecko-1.9.3	Jun 18, 2010	8275	0 KB	Folder
gecko-2.0.0	Mar 18, 2011	9864	0 KB	Folder
COPYING	Aug 3, 2009	5869	31 KB	File
PlatformMatrix.txt	Nov 2, 2009	6593	1 KB	txt
README.txt	Aug 3, 2009	5869	1 KB	txt
releases	Mar 25, 2011	9895	0 KB	Folder

Figure 41: Repositories View

Toolbar

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

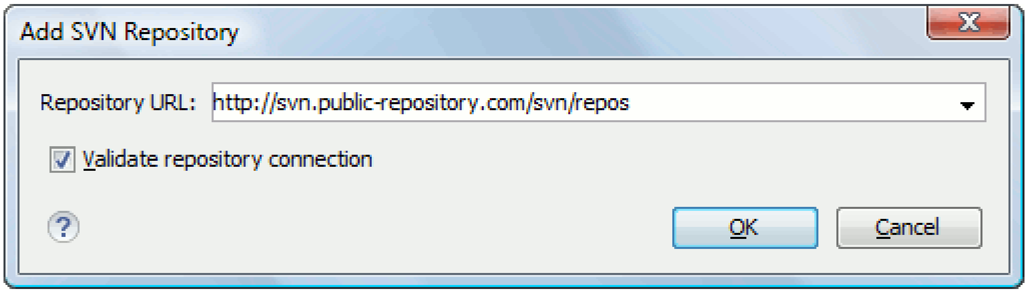
- **New Repository Location** - Allows you to enter a new repository location by means of the **Add SVN Repository** dialog.
- **Move Up** - Move the selected repository up one position in the list of repositories in the **Repositories** view.
- **Move Down** - Move the selected repository down one position 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)** - 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

  Figure 42: Add SVN Repository

  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)** - Context-dependent action that allows you to edit the selected repository location by means of the **Edit SVN Repository** dialog. It is active only when a repository location root is selected.
- **Change the Revision to Browse (Ctrl + Alt + Shift + B)** - Context-dependent action that allows you to change the selected repository revision by means of 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)** - 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.

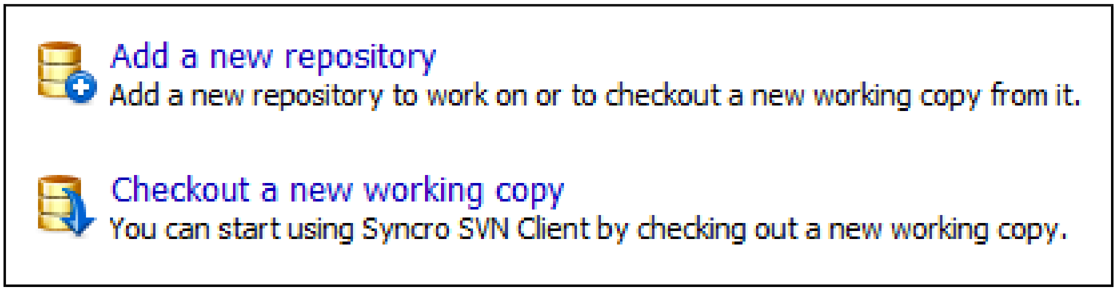
The following action are common to all repository resources:

- **Refresh** - Refreshes the resource selected in the **Repositories** view.
- **Check Out (Ctrl + Alt + Shift + C)** - Allows you to copy resources from a repository into your local file system. To use this operation, you must select a repository root location or a folder from a repository, but never a file. If you don't select anything, you can specify an URL to a folder resource from a repository in the **Check Out** dialog that appears when performing this operation. To read more about this operation, see the section *Check out a working copy*.
- **Export** - Exports a folder from the repository to the local file system.
- **Import** sub-menu:
  - **Import Folder Content (Ctrl + Alt + Shift + M)** - Depending on the selected folder from a repository, allows you to import the contents of a specified folder from the file system into it. To read more about this operation, see the section *Importing resources into a repository*.
  - **Import File(s) (Ctrl + Alt + I)** - Imports the files selected from the files system into the selected folder from the repository.
- **New Folder** - Allows you to create a new folder in the selected repository path.
- **Open** - Opens the selected file in the Editor view in read-only mode.

- **Open with** - Displays the *Open with...* dialog to specify the editor in which the selected file will be opened. In case multiple files are selected, only external applications can be used to open the files.
- **Copy/Move to** **(Ctrl + M)** - Copies or moves to a specified location the selected resource.
- **Rename** **(F2)** - Renames the selected resource.
- **Delete** **(Delete)** - Deletes the selected resource.
- **Copy URL Location** **(Ctrl + Alt + U)** - Copies to clipboard the URL location of the selected resource.
- **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, please see the *Creation and management of Branches/Tags* section.
- **Show History** **(Ctrl + H)** - Displays the history of the selected resource. At the start of the operation you can set filtering options.
- **Show Annotation** **(Ctrl + Shift + A)** - Complex action that does the following operations:
  - opens the selected resource in the **Annotations** editor;
  - displays corresponding annotations list in the **Annotations** view;
  - displays the history of the selected resource.
- **Revision Graph** **(Ctrl + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see the section *Revision Graph*. This operation is enabled for any resource selected into the **Repositories** view or **Working Copy** view.
- **Show SVN Properties** **(Ctrl + Shift + P)** - Brings up the *Properties view* displaying the SVN properties for the selected resource. This view does not allow adding, editing or removing SVN properties of a repository resource. These operations are allowed only for working copy resources.
- **File Information** **(Ctrl + I)** - provides additional information for the selected resource. For more details please see the section *Information view*.

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.

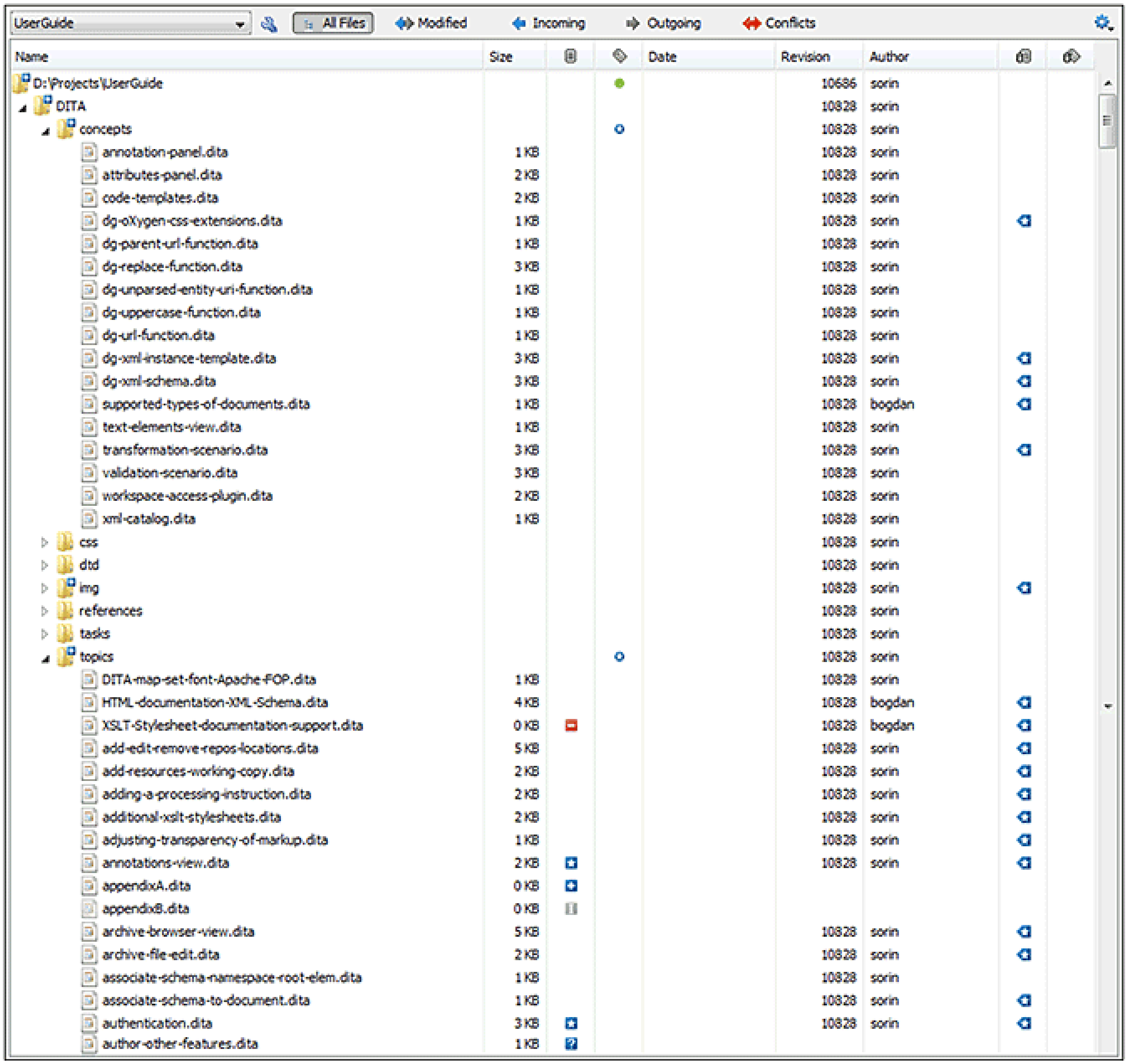
Working Copy View

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), and a **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*.

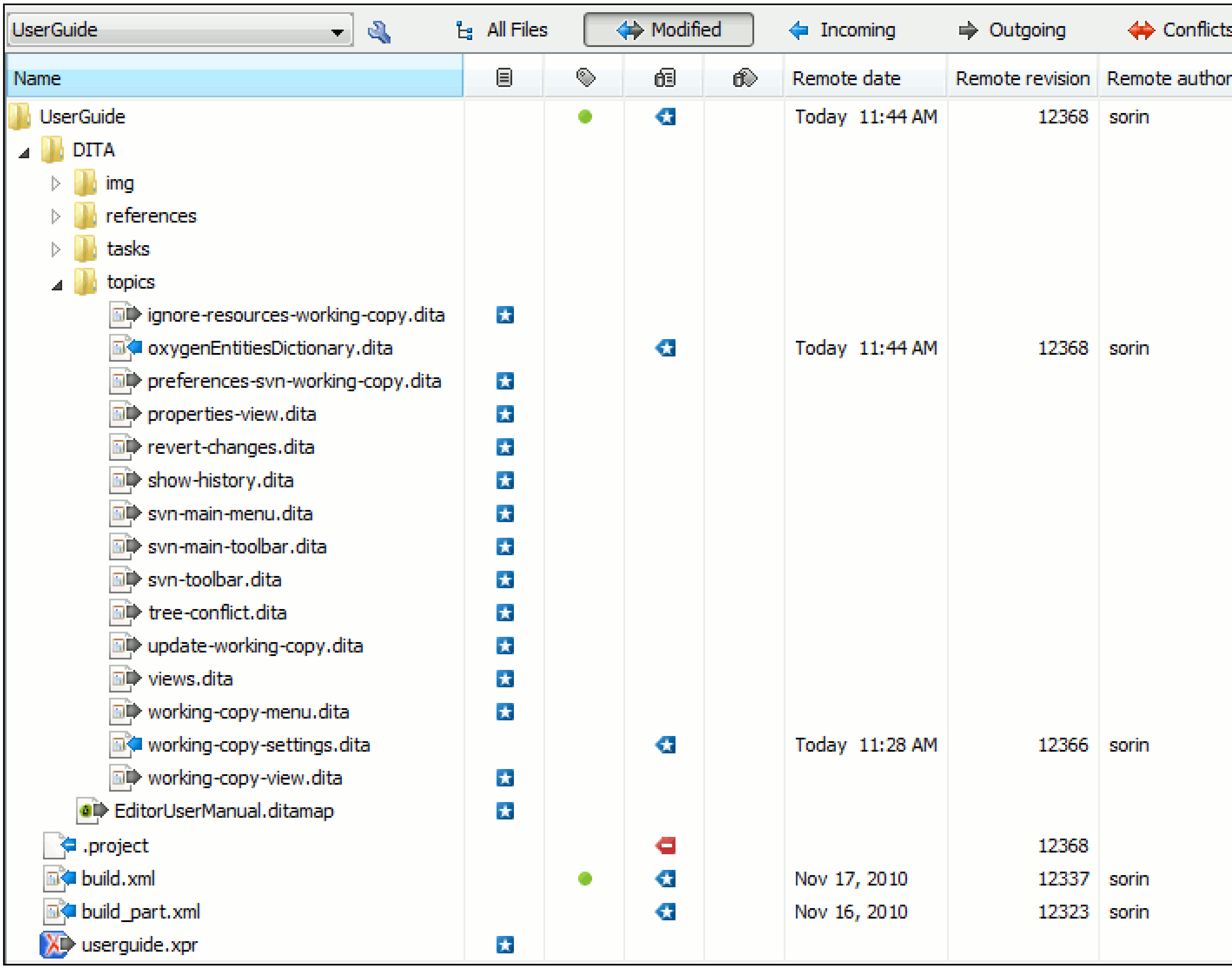


Name	Size			Date	Revision	Author		
D:\Projects\UserGuide			●		10686	sorin		
DITA					10828	sorin		
concepts			○		10828	sorin		
annotation-panel.dita	1 KB				10828	sorin		
attributes-panel.dita	2 KB				10828	sorin		
code-templates.dita	2 KB				10828	sorin		
dg-oXygen-css-extensions.dita	1 KB				10828	sorin	◁	
dg-parent-url-function.dita	1 KB				10828	sorin		
dg-replace-function.dita	3 KB				10828	sorin		
dg-unparsed-entity-uri-function.dita	1 KB				10828	sorin		
dg-uppercase-function.dita	1 KB				10828	sorin		
dg-url-function.dita	1 KB				10828	sorin		
dg-xml-instance-template.dita	3 KB				10828	sorin	◁	
dg-xml-schema.dita	3 KB				10828	sorin	◁	
supported-types-of-documents.dita	1 KB				10828	bogdan	◁	
text-elements-view.dita	1 KB				10828	sorin		
transformation-scenario.dita	3 KB				10828	sorin	◁	
validation-scenario.dita	3 KB				10828	sorin		
workspace-access-plugin.dita	2 KB				10828	sorin		
xml-catalog.dita	1 KB				10828	sorin		
css					10828	sorin		
dtd					10828	sorin		
img					10828	sorin	◁	
references					10828	sorin		
tasks					10828	sorin		
topics			○		10828	sorin		
DITA-map-set-font-Apache-FOP.dita	1 KB				10828	sorin		
HTML-documentation-XML-Schema.dita	4 KB				10828	bogdan	◁	
XSLT-Stylesheet-documentation-support.dita	0 KB	⊟			10828	bogdan	◁	
add-edit-remove-repos-locations.dita	5 KB				10828	sorin	◁	
add-resources-working-copy.dita	2 KB				10828	sorin	◁	
adding-a-processing-instruction.dita	2 KB				10828	sorin	◁	
additional-xslt-stylesheets.dita	2 KB				10828	sorin	◁	
adjusting-transparency-of-markup.dita	1 KB				10828	sorin	◁	
annotations-view.dita	2 KB	⊞			10828	sorin	◁	
appendixA.dita	0 KB	⊞						
appendixB.dita	0 KB	▯						
archive-browser-view.dita	5 KB				10828	sorin	◁	
archive-file-edit.dita	2 KB				10828	sorin	◁	
associate-schema-namespace-root-elem.dita	1 KB				10828	sorin		
associate-schema-to-document.dita	1 KB				10828	sorin	◁	
authentication.dita	3 KB	⊞			10828	sorin	◁	
author-other-features.dita	1 KB	?						

Figure 43: 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 modified remotely;
        - - new file added remotely;
        - - file deleted remotely;
    - - outgoing modification to repository:
        - - file content modified locally;

  - - new file added locally;
  - - file deleted locally;
- - pseudo-conflict state - a resource being locally and remotely modified at the same time;
- - real conflict state - a resource that had both incoming and outgoing changes and not all the differences could be merged automatically by the update operation (manually editing the local file is necessary for resolving the conflict).



Name					Remote date	Remote revision	Remote author
UserGuide		●	←		Today 11:44 AM	12368	sorin
DITA							
img							
references							
tasks							
topics							
ignore-resources-working-copy.dita	★						
oxygenEntitiesDictionary.dita			←		Today 11:44 AM	12368	sorin
preferences-svn-working-copy.dita	★						
properties-view.dita	★						
revert-changes.dita	★						
show-history.dita	★						
svn-main-menu.dita	★						
svn-main-toolbar.dita	★						
svn-toolbar.dita	★						
tree-conflict.dita	★						
update-working-copy.dita	★						
views.dita	★						
working-copy-menu.dita	★						
working-copy-settings.dita			←		Today 11:28 AM	12366	sorin
working-copy-view.dita	★						
EditorUserManual.ditamap	★						
.project			⊖			12368	
build.xml		●	←		Nov 17, 2010	12337	sorin
build_part.xml			←		Nov 16, 2010	12323	sorin
userguide.xpr	★						

Figure 44: 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.
    - **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 will discard any child resource.

    Note: Any folder unmarked with one of the depth icons, has recursive depth (infinity) set by default (presents all levels of child resources).

- **Size** - Resource size on disk;
- **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.
  - I - Resource is being ignored because it is not under version control and its name matches a file name pattern defined in one of the following places:
    - *global-ignores* section in the SVN client-side *'config' file*;
    - ***Application global ignores** option* of Syncro SVN Client;
    - the value of a *svn:ignore property* set on the parent folder of the resource being ignored.
  - + - Marks a newly created resource, scheduled for addition to the version control system.
  - + - Marks a resource scheduled for addition, created by copying a resource already under version control and inheriting all its SVN history.
  - ★ - The content of the resource has been modified.
  - R - 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 scheduled for deletion.
  - ! - The resource is incomplete (as a result of an interrupted *checkout* or *update* operation).
  - ! - The resource is missing because it was moved or deleted without using a SVN aware application.
  - C - The contents of the resource is in *real conflict state*.
  - Tc - 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.
- **Date** - Date when the resource was last time 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.
- **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.
  - R - 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 date** - Date of the resource's latest modification committed on the repository.
- **Remote revision** - Revision number of the resource's latest committed modification.
- **Remote author** - Name of the author who committed the latest modification on the repository.
- **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.

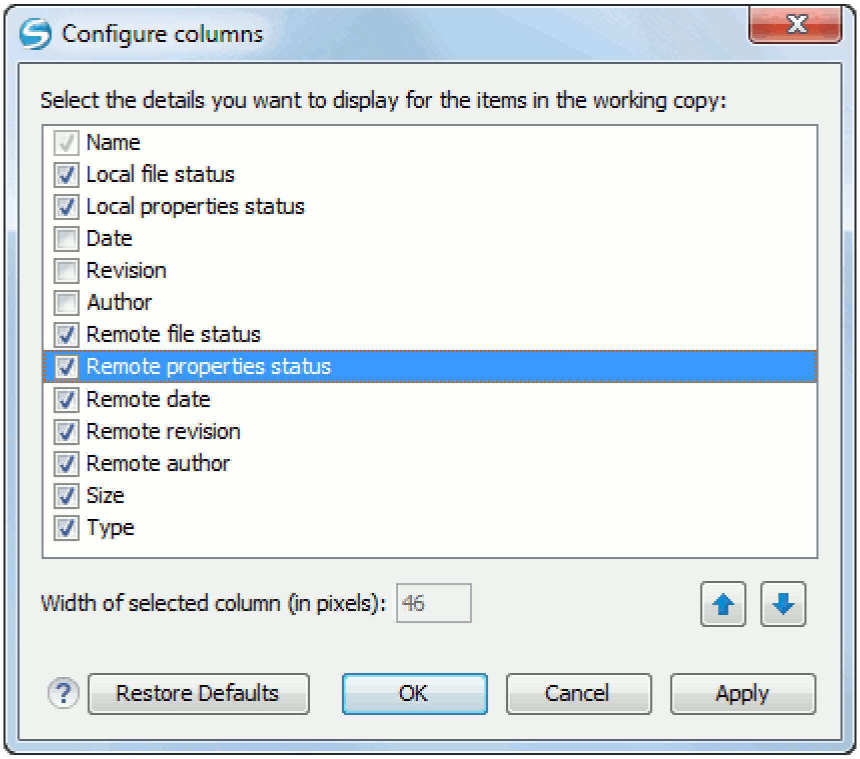
The toolbar allows you to switch between two working copies:

- 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 will be scanned and displayed in the **Working Copy** view.
- **Add/Remove Working Copy** - opens the **Working copies list** dialog which displays the working copies Syncro SVN Client is aware of. In this dialog you can add existing working copies or remove the no longer needed ones. If you try to add a folder which is not a valid Subversion working copy, a warning dialog will inform you that the selected directory is not under version control. Please note that removing a working copy from this dialog will NOT remove it from your file system; you will have to do that manually.

Working Copy Settings

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

- **Show ignored files** - When selected, the application displays the ignored resource inside the **All Files** mode.
- **Show deleted files** - Allows displaying or hiding the deleted resource inside the **All Files** mode. All other modes always display deleted resources, disregarding this action.
- **Tree** / **Compressed** / **Flat** - Affect the way the information is displayed inside the **Modified**, **Incoming**, **Outgoing**, and **Conflicts** view modes.
- **Configure columns** - Allows changing the order and width of table columns and showing/hiding table columns. 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 45: 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 a SVN working copy is loaded, Syncro SVN Client first checks the working copy's format. If it is a SVN 1.6 format, the admin data of that working copy is loaded and displayed in a tree like form in the view using the icons specific to the status of each resource: normal, unversioned, modified, etc. If it is the old format (SVN 1.5, SVN 1.4 or SVN 1.3), a confirmation dialog is displayed allowing the automatic conversion of the working copy to the new format, that is the SVN 1.6 one.

If you select the **Never ask me again** checkbox before pressing the **Yes** button, then the *Automatically upgrade working copies to the client's version* option is automatically checked. From now on, all other older version working copies will be automatically updated to the latest version.

The format of the working copy can be downgraded or upgraded at any time with the **Upgrade** and **Downgrade** actions available in the **Tools** menu. These actions allow switching between SVN 1.4, SVN 1.5 and SVN 1.6 working copy formats.

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 + 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 + Alt + C)** - Displays changes made in the currently selected file.
- **Open (Ctrl + 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 a default file system browsing application (e.g. Windows Explorer on Windows, Finder on Mac OS X, etc).
- **Open with (Ctrl + 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.
- **Expand all (Ctrl + Alt + X)** - Displays all descendants of the selected folder. You can obtain a similar behavior by double-clicking on a collapsed folder.
- **Refresh (F5)** - Re-scans the selected resources recursively and refreshes their status in the working copy view.
- **Synchronize (Ctrl + 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 + U)** - Updates the selected resources to the *HEAD* revision (latest modifications) from the repository. If the selection contains a directory, it will be updated depending on its *depth*.
- **Update to revision/depth** - Allows you to update the selected resources from the working copy to an earlier revision from the repository. You can also select the update *depth* for the current folder. You can find out more about the *depth* term in the *sparse checkouts* section.
- **Commit** - Collects the outgoing changes from the selected resources in the working copy and allows you to choose exactly what to commit by selecting or not resources. A directory will always be committed recursively. The unversioned resources will be deselected by default. In the commit dialog you can also enter a commit comment before sending your changes to the repository.
- **Revert (Ctrl + Shift + V)** - Undoes all local changes for the selected resources. It does not contact the repository, the files are obtained from Subversion's pristine copy. It is enabled only for modified resources. See *Revert your changes* for more information.
- **Override and Update** - Drops any outgoing change and replaces the local resource with the HEAD revision. Action available on resources with outgoing changes, including the conflicting ones. See the *Revert your changes* section.
- **Override and Commit** - Drops any incoming changes and sends your local version of the resource to the repository. Action available on conflicting resources. See also the section *Drop incoming modifications*.
- **Mark Resolved (Ctrl + Shift + R)** - Instructs the Subversion system that you resolved a conflicting resource. For more information, see *Merge conflicts*.
- **Mark as Merged (Ctrl + 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 + 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 + 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 + Alt + C)** - Compares the working copy file with the BASE revision file (the so-called *pristine copy*).

  - **Revision (Ctrl + Alt + R)** - Shows the **History view** containing the log history of that resource.
  - **Branch/Tag** - Compares the working copy file with a revision of the file from a branch or tag. The revision is specified by URL (selected with a repository browser dialog) and revision number (selected with a revision browser dialog).
  - **Each other** - Compares two selected files with each other.

  These *compare* actions are enabled only if the selected resource is a file.
- **Replace with**:
  - **Latest from HEAD** - Replaces the selected resources with their versions from the *HEAD* revision of the repository.
  - **BASE revision** - Replace the selected resources with their versions from the pristine copy (the BASE revision).

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

    - for **Replace with BASE revision** action, the resources being unversioned or added have no *BASE* revision, so they cannot be replaced. However, they will be deleted if the action is invoked on a parent folder. The action will never work for missing folders or for obstructing files (folders being obstructed by a file), because you cannot recover a tree of folders;
    - for **Replace with latest from HEAD** action, you must be aware that there are cases when resources will be completely deleted or reverted to BASE revision and after that updated to HEAD revision, in order to avoid conflicts. These cases are:
      - the resource is *unversioned*, *added*, *obstructed* or *modified*;
      - the resource is affected by a *svn:ignore* or *svn:externals* property which is locally added on the parent folder and not yet committed to the repository.

- **Show History (Ctrl + 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)** - It will display the **Annotations view** where all the users that modified the selected resource will be presented together with the specific lines and revision numbers modified by each user. For more details about resource annotations see *Annotations View*.
- **Revision Graph (Ctrl + Shift + G)** - This action allows you to see the graphical representation of a resource's history. For more details about a resource's revision graph see *Revision Graph*.
- **Copy URL Location (Ctrl + Alt + U)** - Copies the encoded URL of the selected resource from the Working Copy to the clipboard.
- **Copy/Move to (Ctrl + M)** - Copies or moves to a specified location the currently selected resource.
- **Rename (F2)** - You can only rename one resource at a time. As for the move command, a copy of the original resource will be made with the new name and the original will be marked as deleted.
- **Delete (Delete)** - Allows you to delete resources from the Subversion working copy. If *unversioned*, *added* or *modified* resources will be encountered, a dialog will prompt you to confirm their deletion, because their content cannot be recovered. The action is not enabled when the selection contains *missing* resources.
- **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 + 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 + Shift + W)** - Creates a new folder inside the selected folder, having the contents of a target folder from the current working copy's repository. The application does this by setting a *svn:externals* property on the selected folder and updating the folder in order to bring all the newly pointed resources inside your current working copy. The operation shows a dialog which allows you to specify the new folder's name and easily select the target folder by browsing the repository's contents.

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

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

- **Add to version control** **(Ctrl + Alt + V)** - Adds the selected resources to version control. A directory will be added recursively to version control. It is not mandatory to explicitly add resources to version control but it is recommended. At commit time unversioned resources will have to be manually selected in the commit dialog. This action is only active on unversioned resources.
- **Add to "svn:ignore"** **(Ctrl + Alt + I)** - Allows you to keep inside your working copy files that should not participate to the version control operations. This action can only be performed on resources not under version control. It actually modifies the value of the *svn:ignore* property of the resource's parent directory. Read more about this in the *Ignore Resources Not Under Version Control* section.
- **Cleanup** **(Ctrl + Shift + C)** - Performs a maintenance cleanup operation to the selected resources from the working copy. This operation removes the Subversion maintenance locks that were left behind. Useful when you already know where the problem originated and want to fix it as quickly as possible. Only active for resources under version control.
- **Locking**:
  - **Scan for locks** **(Ctrl + L)** - Contacts the repository and recursively obtains the list of locks for the selected resources. A dialog containing the locked files and the lock description will be displayed. Only active for resources under version control. For more details see *Scanning for locks*.
  - **Lock** **(Ctrl + K)** - Allows you to lock certain files for which you need exclusive access. You can write a comment describing the reason for the lock and you can also force(*steal*) the lock. The action is active only on files under version control. For more details on the use of this action see *Locking a file*.
  - **Unlock** **(Ctrl + Alt + K)** - Releases(unlocks) 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)** - Brings up the *Properties view* and displays the SVN properties for the selected resource.
- **File Information** **(Ctrl + I)** - Provides additional information for the selected resource from the working copy. For more details please see *Obtain information for a resource*.

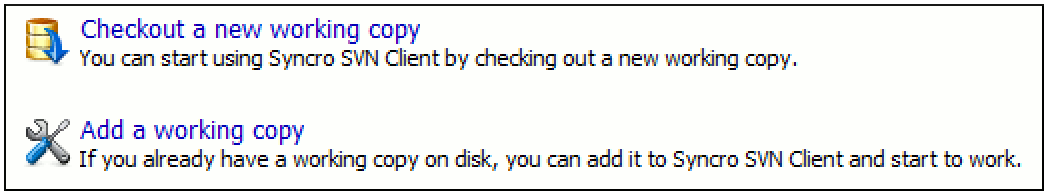
Drag and Drop Operations

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

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:



Checkout 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 46: 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.

Icons

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

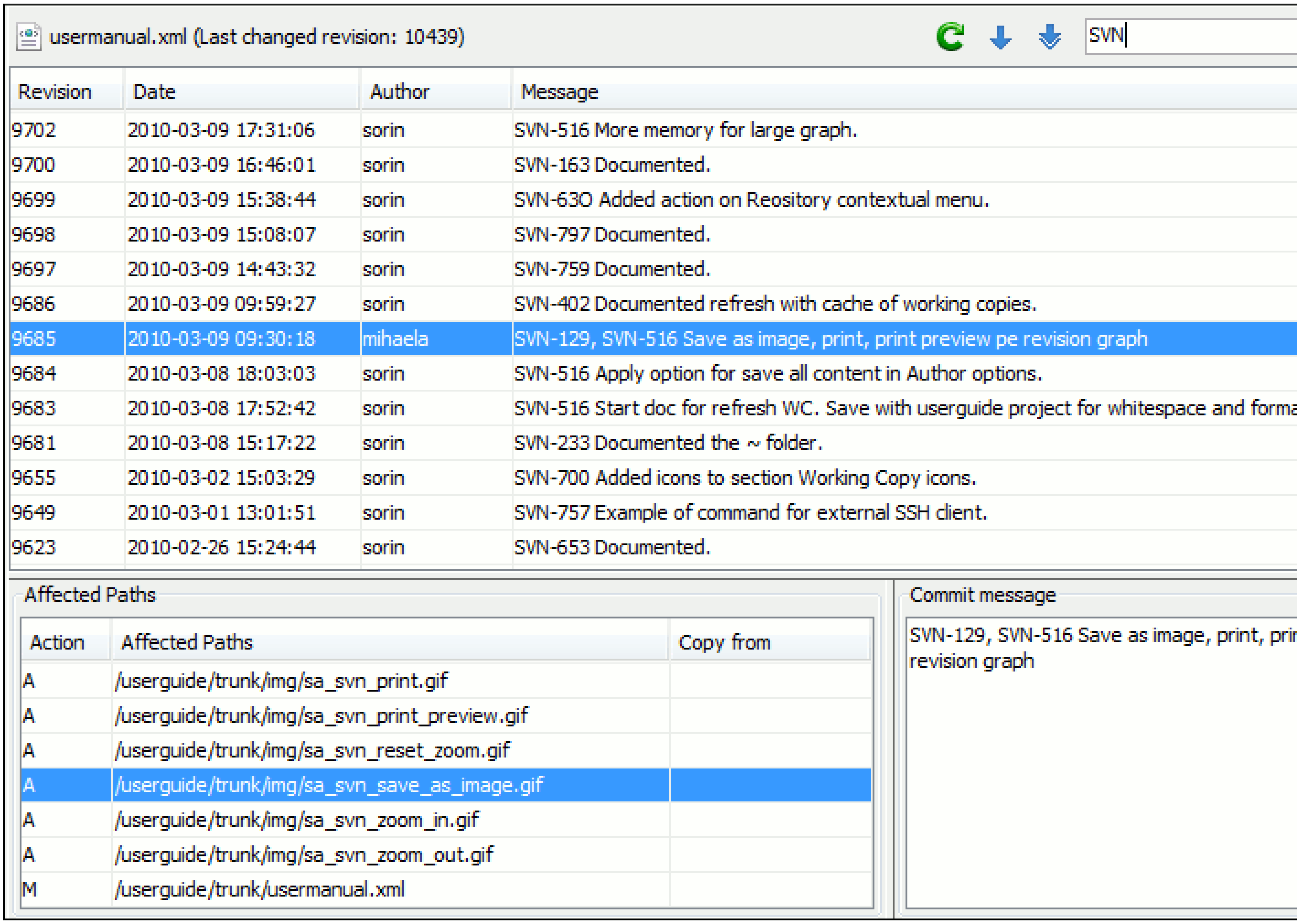
- **Modified** - The resource has been locally modified since the last update. This is obtained after editing a file and making changes.
- **External** - This indicates a mapping of a local directory to the URL of a versioned resource. It is declared with a *svn:externals* property in the parent folder.
- *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.
- **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.
- **Immediate children** (*sparse checkout*) - The directory is marked with a purple bubble in the top left corner.
- **File children only** (*sparse checkout*) - The directory is marked with a blue bubble in the top left corner.
- **This folder only - empty** (*sparse checkout*)- The directory is marked with a gray bubble in the top left corner.

History View

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

The view consists of three distinct areas:

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



usermanual.xml (Last changed revision: 10439)

SVN

Revision	Date	Author	Message
9702	2010-03-09 17:31:06	sorin	SVN-516 More memory for large graph.
9700	2010-03-09 16:46:01	sorin	SVN-163 Documented.
9699	2010-03-09 15:38:44	sorin	SVN-63O Added action on Reository contextual menu.
9698	2010-03-09 15:08:07	sorin	SVN-797 Documented.
9697	2010-03-09 14:43:32	sorin	SVN-759 Documented.
9686	2010-03-09 09:59:27	sorin	SVN-402 Documented refresh with cache of working copies.
9685	2010-03-09 09:30:18	mihaela	SVN-129, SVN-516 Save as image, print, print preview pe revision graph
9684	2010-03-08 18:03:03	sorin	SVN-516 Apply option for save all content in Author options.
9683	2010-03-08 17:52:42	sorin	SVN-516 Start doc for refresh WC. Save with userguide project for whitespace and forma
9681	2010-03-08 15:17:22	sorin	SVN-233 Documented the ~ folder.
9655	2010-03-02 15:03:29	sorin	SVN-700 Added icons to section Working Copy icons.
9649	2010-03-01 13:01:51	sorin	SVN-757 Example of command for external SSH client.
9623	2010-02-26 15:24:44	sorin	SVN-653 Documented.

Affected Paths

Action	Affected Paths	Copy from
A	/userguide/trunk/img/sa_svn_print.gif	
A	/userguide/trunk/img/sa_svn_print_preview.gif	
A	/userguide/trunk/img/sa_svn_reset_zoom.gif	
A	/userguide/trunk/img/sa_svn_save_as_image.gif	
A	/userguide/trunk/img/sa_svn_zoom_in.gif	
A	/userguide/trunk/img/sa_svn_zoom_out.gif	
M	/userguide/trunk/usermanual.xml	

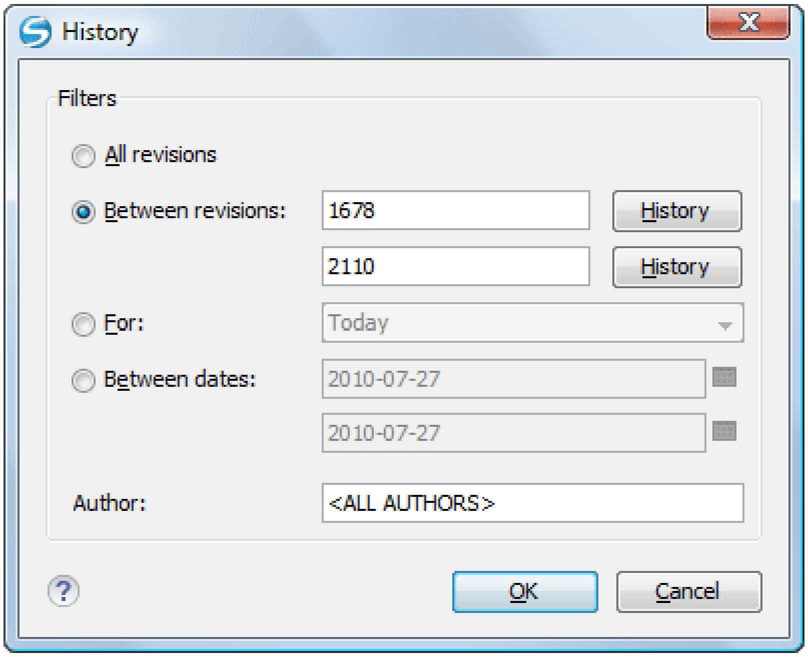
Commit message

SVN-129, SVN-516 Save as image, print, pri
revision graph

Figure 47: History View

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.



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

Figure 48: History Filters Dialog

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

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

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

The History Filter Field

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

Features

Single selection actions:

- **Compare with working copy** - Compares the selected revision with your working copy file. It is enabled only when you select a file.
- **Open** - opens the selected revision of the file into the Editor. This is enabled only for files.
- **Open with ...** - Displays the *Open with...* dialog to specify the editor in which the selected file will be opened.
- **Get Contents** - Replaces the current version from the working copy with the contents of the selected revision from the history of the file. The *BASE* version of the file is not changed in the working copy so that after this action the file will appear as modified in a synchronization operation, that is newer than the *BASE* version, even if the contents is from an older version from history.

- **Save revision to** - Saves the selected revision to a file so you have an older version of that file. This option is only available when you access the history of a file, and it saves a version of that file only.
- **Revert changes from this revision** - Reverts changes made in the selected revision.
- **Update to revision** - Updates your working copy resource to the selected revision.
- **Check out from revision** - Gets the content of the selected revision for the resource into local file system.
- **Show Annotation** - Computes the latest revision number and author name that modified each line of the file up to the selected revision, that is no modification later than the selected revision is taken into account.
- **Change Author** - Changes the name of the SVN user that committed the selected revision.
- **Change Message** - Changes the commit message of the selected revision.

Double selection actions:

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

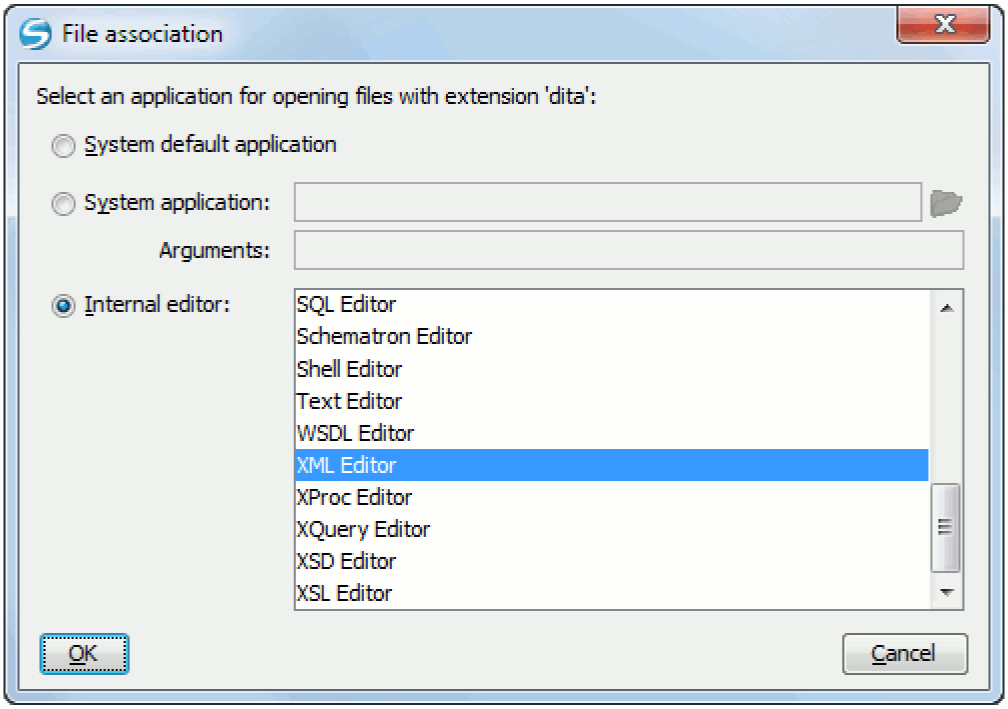
For more information about the **History view** and its features please read the sections *Request history for a resource* and *Using the resource history view*

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:



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 49: 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.

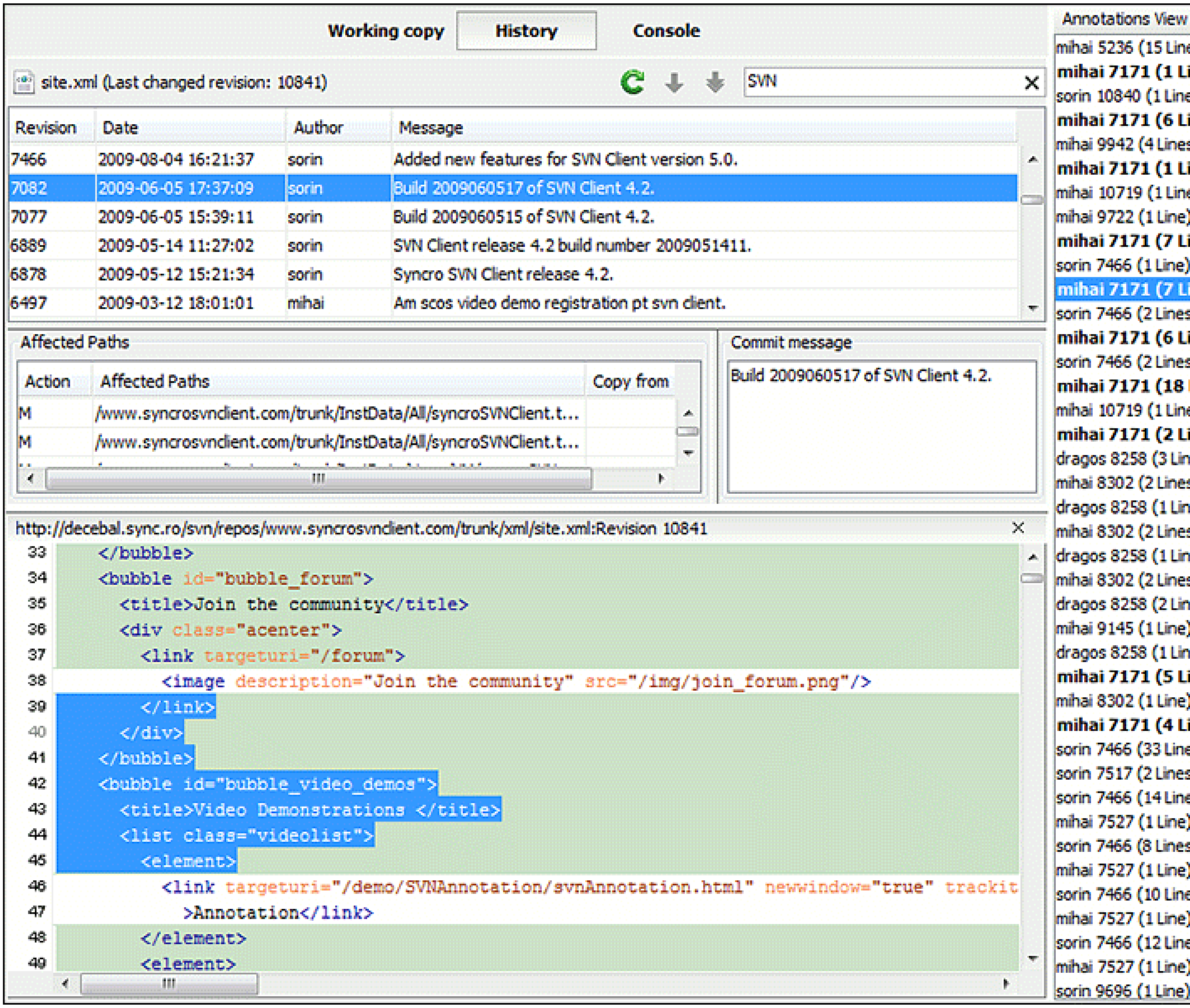
Only one file at a time can be edited in an internal editor. If you try to open another file it will be opened in the same editor window. The editor provides syntax highlighting for known file types. This means that a different color will be used for each recognized token type found in the file. If the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

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

Annotations View

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



site.xml (Last changed revision: 10841)

SVN

Revision	Date	Author	Message
7466	2009-08-04 16:21:37	sorin	Added new features for SVN Client version 5.0.
7082	2009-06-05 17:37:09	sorin	Build 2009060517 of SVN Client 4.2.
7077	2009-06-05 15:39:11	sorin	Build 2009060515 of SVN Client 4.2.
6889	2009-05-14 11:27:02	sorin	SVN Client release 4.2 build number 2009051411.
6878	2009-05-12 15:21:34	sorin	Syncro SVN Client release 4.2.
6497	2009-03-12 18:01:01	mihai	Am scos video demo registration pt svn client.

Affected Paths

Action	Affected Paths	Copy from
M	/www.syncrosvnclient.com/trunk/InstData/All/syncroSVNClient.t...	
M	/www.syncrosvnclient.com/trunk/InstData/All/syncroSVNClient.t...	

Commit message

Build 2009060517 of SVN Client 4.2.

http://decebal.sync.ro/svn/repos/www.syncrosvnclient.com/trunk/xml/site.xml:Revision 10841

```
    </bubble>
    <bubble id="bubble_forum">
      <title>Join the community</title>
      <div class="acenter">
        <link targeturi="/forum">
          <image description="Join the community" src="/img/join_forum.png"/>
        </link>
      </div>
    </bubble>
    <bubble id="bubble_video_demos">
      <title>Video Demonstrations </title>
      <list class="videolist">
        <element>
          <link targeturi="/demo/SVNAnnotation/svnAnnotation.html" newwindow="true" trackit
            >Annotation</link>
        </element>
        <element>
```

```
mihai 5236 (15 Line
mihai 7171 (1 Li
sorin 10840 (1 Line
mihai 7171 (6 Li
mihai 9942 (4 Lines
mihai 7171 (1 Li
mihai 10719 (1 Line
mihai 9722 (1 Line)
mihai 7171 (7 Li
sorin 7466 (1 Line)
mihai 7171 (7 Li
sorin 7466 (2 Lines
mihai 7171 (6 Li
sorin 7466 (2 Lines
mihai 7171 (18
mihai 10719 (1 Line
mihai 7171 (2 Li
dragos 8258 (3 Lin
mihai 8302 (2 Lines
dragos 8258 (1 Lin
mihai 8302 (2 Lines
dragos 8258 (1 Lin
mihai 8302 (2 Lines
dragos 8258 (2 Lin
mihai 9145 (1 Line)
dragos 8258 (1 Lin
mihai 7171 (5 Li
mihai 8302 (1 Line)
mihai 7171 (4 Li
sorin 7466 (33 Line
sorin 7517 (2 Lines
sorin 7466 (14 Line
mihai 7527 (1 Line)
sorin 7466 (8 Lines
mihai 7527 (1 Line)
sorin 7466 (10 Line
mihai 7527 (1 Line)
sorin 7466 (12 Line
mihai 7527 (1 Line)
sorin 9696 (1 Line)
```

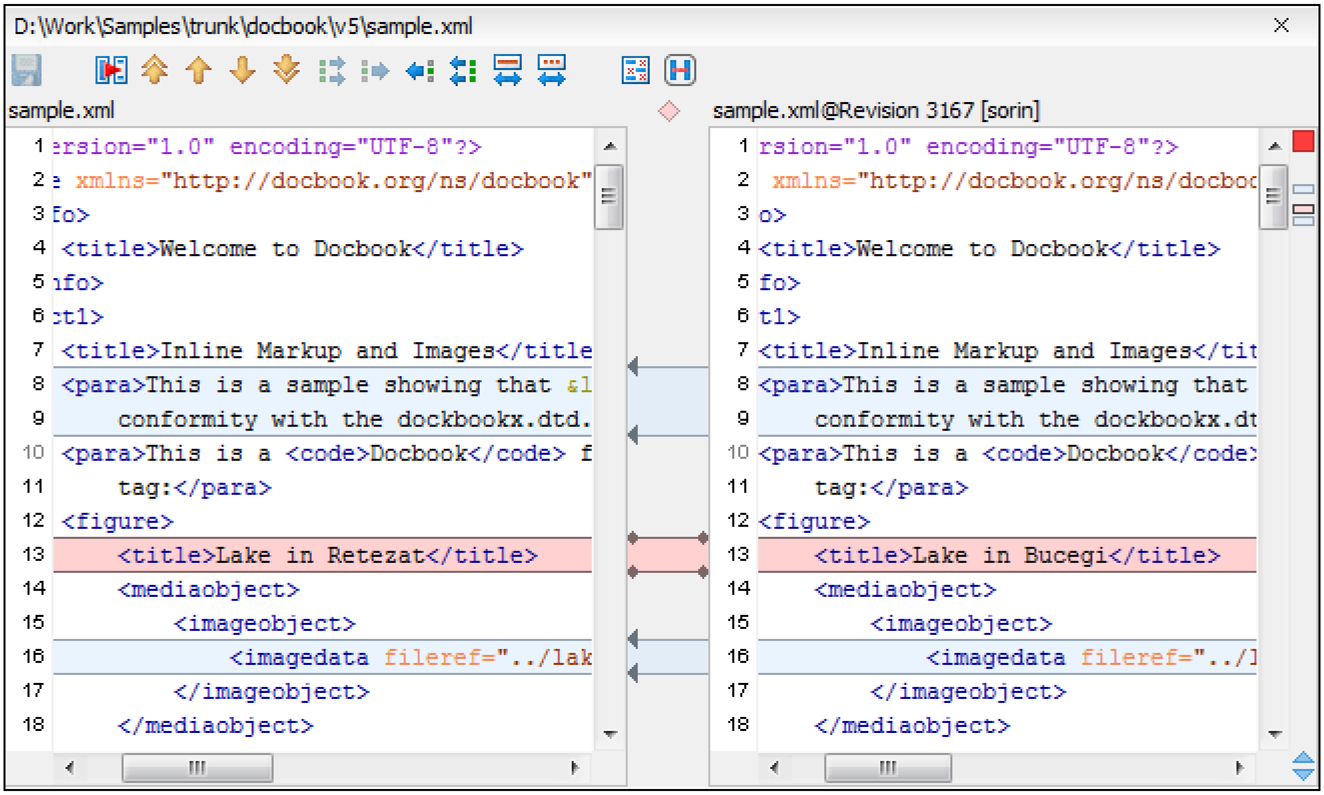
Figure 50: The Annotations View

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

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

Compare View

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



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

sample.xml

```
ersion="1.0" encoding="UTF-8"?>
e xmlns="http://docbook.org/ns/docbook"
fo>
 <title>Welcome to Docbook</title>
ıfo>
ct1>
 <title>Inline Markup and Images</title
 <para>This is a sample showing that &l
     conformity with the dockbookx.dtd.
 <para>This is a <code>Docbook</code> f
     tag:</para>
 <figure>
     <title>Lake in Retezat</title>
     <mediaobject>
         <imageobject>
             <imagedata fileref="../lak
         </imageobject>
     </mediaobject>
```

sample.xml@Revision 3167 [sorin]

```
rsion="1.0" encoding="UTF-8"?>
 xmlns="http://docbook.org/ns/docboc
o>
<title>Welcome to Docbook</title>
fo>
t1>
<title>Inline Markup and Images</tit
<para>This is a sample showing that
    conformity with the dockbookx.dt
<para>This is a <code>Docbook</code>
    tag:</para>
<figure>
    <title>Lake in Bucegi</title>
    <mediaobject>
        <imageobject>
            <imagedata fileref="../l
        </imageobject>
    </mediaobject>
```

Figure 51: Compare View

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

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

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

The Compare view contains two editors. Edits are allowed only in the left editor and only when it contains the working copy file. To learn more about how the view can be used in the day by day work see *View differences*.

Toolbar

The list of actions available in the toolbar consists of:

- **Save action** - Saves the content of the left editor when it can be edited.
- **Perform files differencing** - Performs files differencing on request.
- **Go to first modification** - Navigates to the first found difference.
- **Go to previous modification** - Navigates to the previous found difference.
- **Go to next modification** - Navigates to the next found difference.
- **Go to last modification** - Navigates to the last found difference.

- **Copy change from right to left** - Copies the selected change from the right editor to the left editor.
- **Copy all non-conflicting changes from right to left** - Copies all non-conflicting changes from the right editor to the left editor. A non-conflicting change from the right editor is a change that does not overlap with a left editor change.
- **Show modification details at word level** - Compares the currently selected change at word level to provide more details about that change.
- **Show modification details at character level** - Compares the currently selected change at character level to provide more details about that change.
- **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** - Scrolls both the views of the **Compare** view at the same time so that the two sides of a difference are visible at the same time.

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

Image Preview

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

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

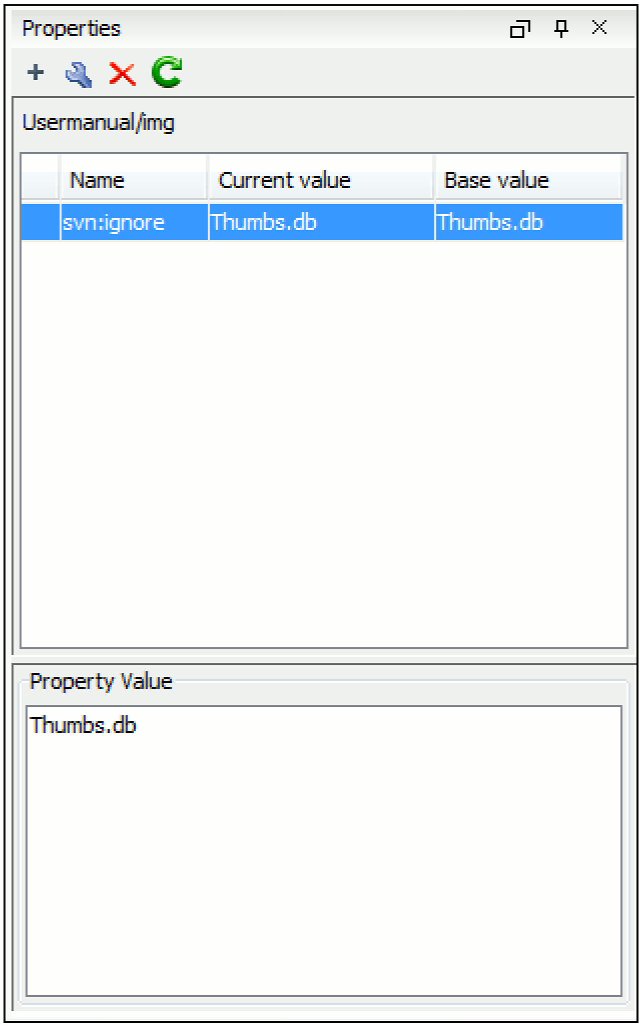
Compare Images View

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

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

Properties View

The properties view presents 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

Thumbs.db

Figure 52: The Properties View

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

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

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

The `svn:externals` Property

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

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

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

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

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

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

⚠ **Attention:**

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

Toolbar / Contextual Menu

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

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

Console View

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 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 referring to the currently selected view. As you change the focused view, you are able to read a short description of it and its functionality.

The Revision Graph of a SVN Resource

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

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

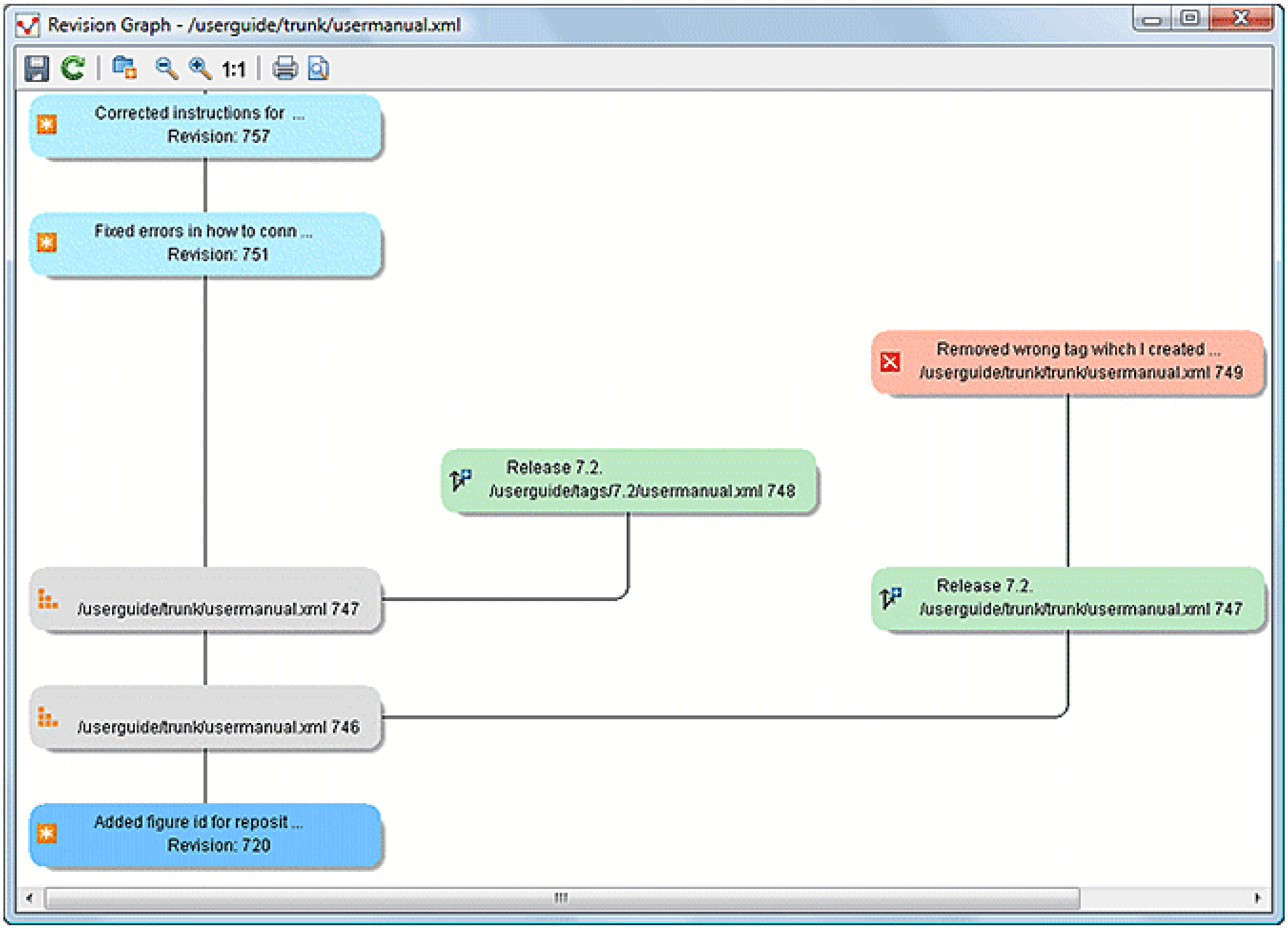

Revision Graph - /userguide/trunk/usermanual.xml
1:1
Corrected instructions for ...
Revision: 757
Fixed errors in how to conn ...
Revision: 751
Removed wrong tag wihch I created ...
/userguide/trunk/trunk/usermanual.xml 749
Release 7.2.
/userguide/tags/7.2/usermanual.xml 748
/userguide/trunk/usermanual.xml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

Figure 53: 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.

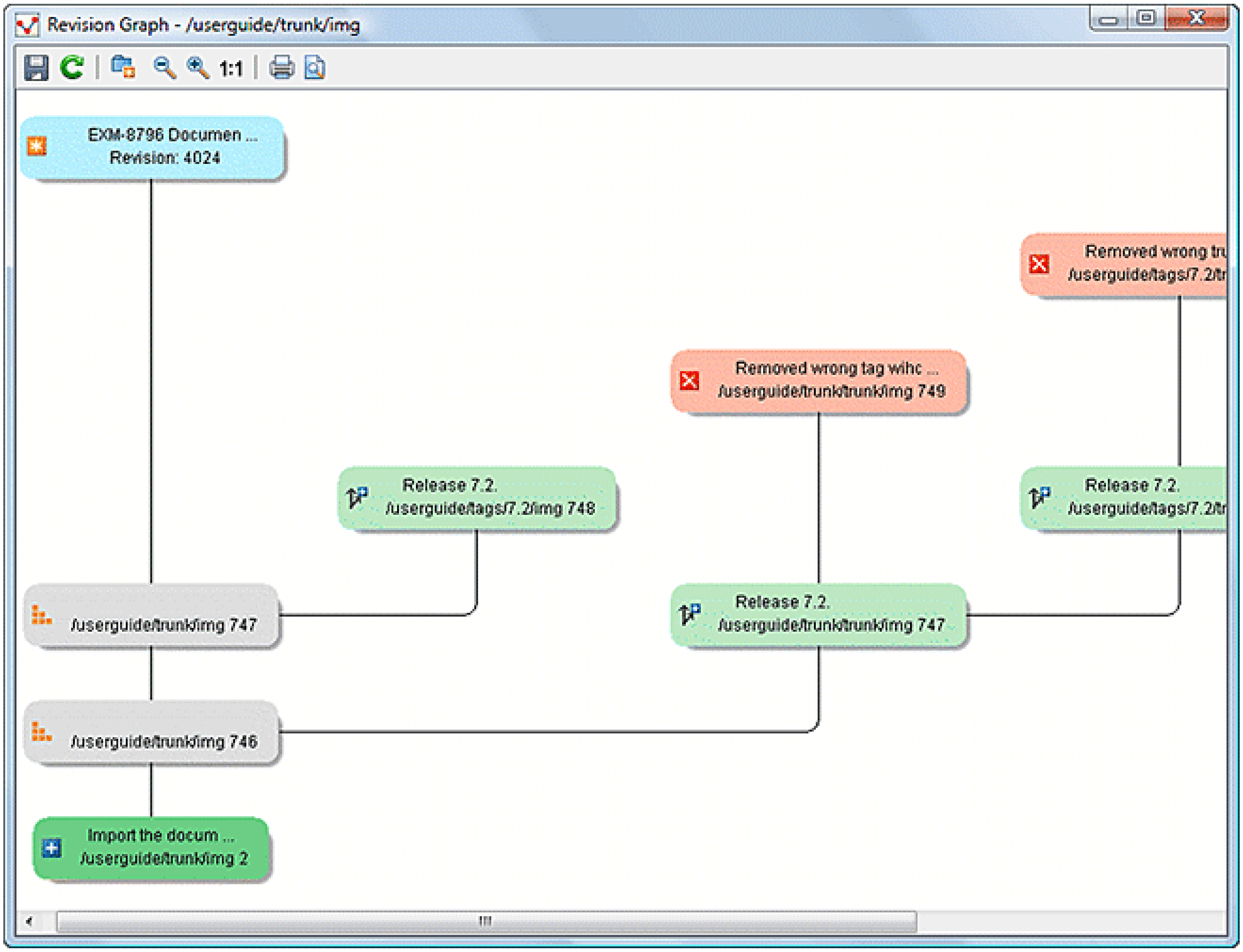


Figure 54: The Revision Graph of a Directory (Direct Changes)

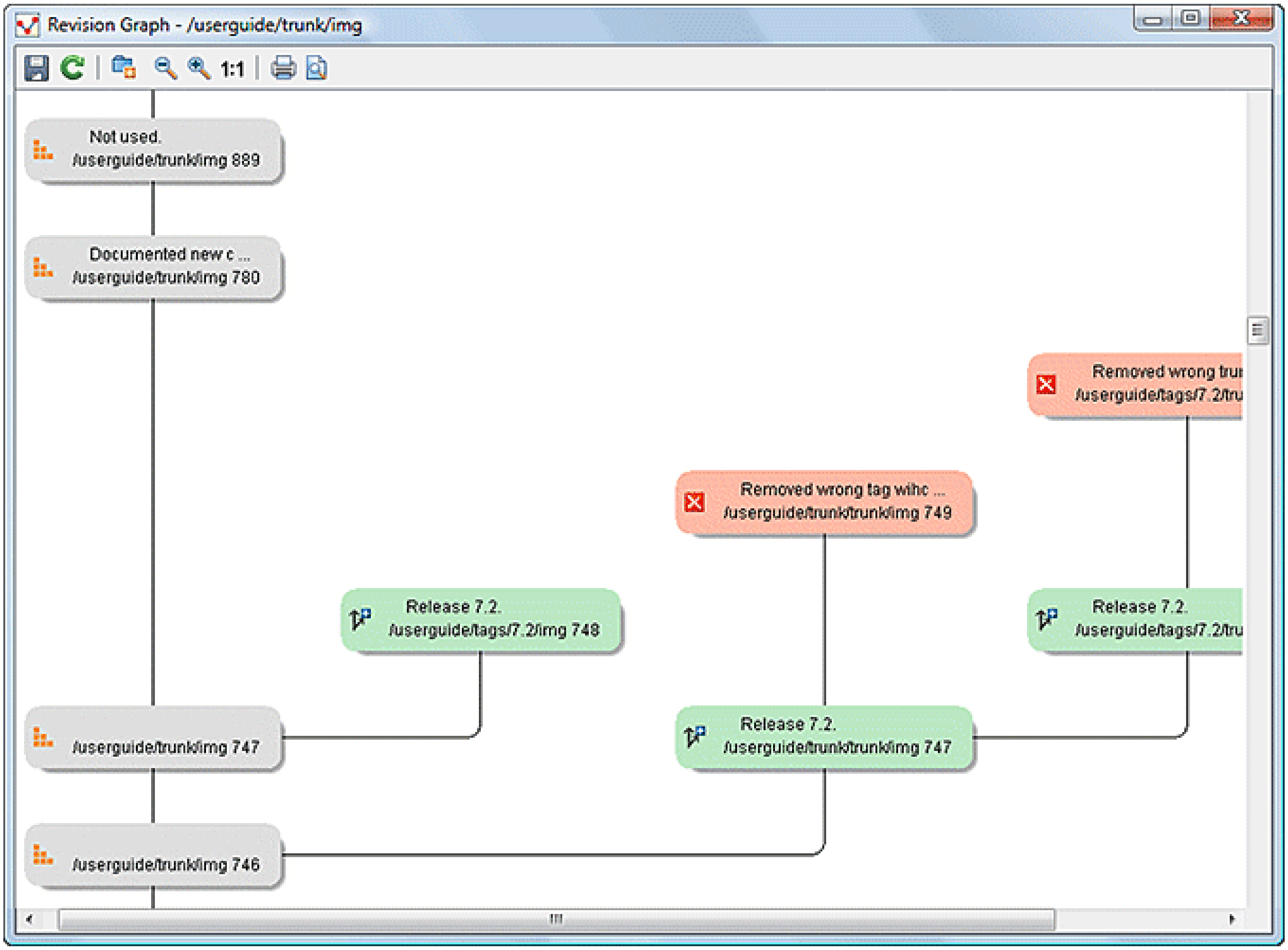


Figure 55: The Revision Graph of a Directory (Also Indirect Changes)

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

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

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

- **Open** - Opens the selected revision in the editor panel. Available only for files.
- **Open with...** - Opens the selected revision in the editor panel. Available only for files.
- **Compare with HEAD** - Compares the selected revision with the HEAD revision and displays the result in the diff panel. Available only for files.

- **Show History** - Displays the history of the resource in *the History view*. Available for both files and directories.
- **Check Out** - *Checks out* the selected revision of the directory. Available only for directories.

When two nodes are selected in the revision graph of a file the right click menu of this selection contains only the **Compare** for comparing the two revisions corresponding to the selected nodes. If the resource for which the revision graph was built is a folder then the right click menu displayed for a two nodes selection also contains the **Compare** action but it computes the differences between the two selected revisions as a set of directory changes. The result is displayed in the *Directory Change Set* view in the same way as for *the compare action invoked from the History view* on two revisions of a folder.

Attention:

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

Command Line Reference

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

Checkout Command

Used to pull a SVN tree from the server to the local file system. The syntax of checkout command is the following.

```
svn checkout --revision rev  URL  PATH
```

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

Update Command

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

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

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

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

Commit Command

Sends changes from your working copy to the repository.

```
svn commit -m "log message"--no-unlock   PATH
```

-m "log message"	Specifies the commit comment.	
--no-unlock		Specifies that the resource should keep locks after commit if this is the case.

PATH		Location on the file system of the resource to commit. Can be more than one.

Diff Command

Displays the differences found between two revisions.

```
svn diff --revision rev1:rev2 PATH
```

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

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

Show History

Display commit log messages.

```
svn log --revision rev1:rev2 --limit N --verbose PATH
```

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

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

Refresh

Print the status of working copy files and directories.

```
svn status --verbose PATH
```

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

Synchronize

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

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

Import

Commits an unversioned file or tree into the repository.

```
svn import -m "log message" PATH URL
```

-m "log message"	Specifies the commit log message.

PATH		Local path to the resource on the file system.
URL	URL on the repository where the resource will be imported.	

Export

Exports a directory tree.

```
svn export --revision rev URL PATH
```

rev	Specifies the desired revision(if necessary).	
URL		Repository URL you want to export from.
PATH		Location on the file system where to export.

Information

Displays information about a local or remote item.

```
svn info --revision rev PATH|URL
```

rev	Specifies the revision number for which the information will be requested.	
PATH		Local file system path to the resource.
URL		Repository URL for the resource.

Add

Add files, directories, or symbolic links.

```
svn add PATH...
```

PATH	Local file system path of the unversioned resources to be added to version control. More than one can be specified.

Add to svn:ignore

```
svn propset svn:ignore PATH PARENTPATH
```

svn:ignore	Predefined property name for ignoring resources.	
PATH		Relative path from the working copy root for the resource to be ignored.
PARENTPATH		Path to the parent of the resource to be ignored.

Delete

Deletes resources from a working copy or from a Subversion repository.

```
svn delete --recursive PATH|URL
```

--recursive	Specifies that the operation should be performed recursively.	
PATH		Local file system path of the resource to delete.
PARENTPATH		Repository URL of the resource to delete.

Copy

Copy a file or directory in a working copy or in the repository.

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

SRCPATH	Working copy path of the resource to be copied.	
DSTPATH		Working copy path where the resource will be copied to.
SRCURL	Repository path of the resource to be copied.	
DSTURL	Repository path where the resource will be copied to.	

Move / Rename

Move a file or directory.

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

SRCPATH	Working copy path of the resource to be moved.	
DSTPATH		Working copy path where the resource will be moved to.
SRCURL	Repository path of the resource to be moved.	
DSTURL	Repository path where the resource will be moved to.	

Mark resolved

`svn resolved` --recursive *PATH*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Path to the resource in the local working copy.

Revert

Undo all local edits.

`svn revert` [--recursive] *PATH*

--recursive	Specifies that the operation should be performed recursively.	
PATH		Local working copy path to revert to.

Cleanup

Recursively cleans up the working copy.

`svn cleanup` *PATH*

PATH	Local working copy path to clean up.

Show / Refresh Properties

`svn proplist` *PATH*

`svn propget` *PROPNAME PATH*

PATH	Local path of the resource.
PROPNAME	Property name.

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

Branch / Tag

`svn copy` -m "log message" *URL1 URL2*

`svn copy` -m "log message" *URL1@rev1 URL2*

`svn copy` -m "log message" *PATH URL*

-m "log message"	Commit message.	
URL1		Source repository URL.
rev1	Revision of the source.	
URL2	Destination repository URL.	
PATH	Source working copy path.	
URL	Destination repository URL.	

Merge

Apply the differences between two sources to a working copy path.

`svn merge` [--dry-run] *rev1:rev2 URL PATH*

`svn merge` [--dry-run] *URL1@rev1 URL2@rev2 PATH*

--dry-run	Specifies that the operation will be simulated without making any modifications.	
URL		Repository URL for the resource to merge.
URL1	Repository URL for the start branch to merge.	
rev1	Start revision for the resource to merge.	
URL2	Repository URL for the end branch to merge.	
rev2	End revision for the resource to merge.	
PATH	Destination path in the working copy for the result of the merge.	

Scan for locks

Obtains the repository status for all the resources in the path.

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

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

Lock

Lock working copy paths or URLs in the repository so that no other user can commit changes to them.

```
svn lock [--force] [-m "log message"] PATH
```

--force	Forces(steals) the lock.	
-m "log message"		Lock message.
PATH		Path to the file to be locked.

Unlock

Unlock working copy paths or URLs.

```
svn unlock [--force] PATH
```

--force	Forces(breaks) the lock.	
PATH		Path to the file from the working copy..

Mark as merged

```
rename FILE FILE.TMP
svn update FILE
rename FILE.TMP FILE
```

FILE	File to be marked as merged.	
FILE.TMP		Temporary filename.

Override and update

```
svn revert PATH
svn update PATH
```

PATH	Path of the resource to be overridden.

Override and Commit

If the resource is in conflict first you should perform a *Mark Resolved action*. If the resource has incoming changes you should perform a *Mark as Merged action* followed by a *Commit action*.

Add / Edit property

`svn propset` [--recursive] *PROPNAME* *PROPVALUE* *PATH*

--recursive	Specifies that the property should be set recursively.	
PROPNAME		Property name.
PROPVALUE	Property value.	
PATH		Resource's path.

Remove property

Removes a property from an item.

`svn propdel` [--recursive] *PROPNAME* *PATH*

--recursive	Specifies that the property should be deleted recursively.	
PROPNAME		Property name.
PATH		Resource's path.

Revert changes from this revision

`svn merge` rev:rev-1 *URL*

rev	Revision whose changes must be reverted.	
URL		The SVN URL corresponding to the resource.

Revert changes from these revisions

Short reference description.

`svn merge` rev1:rev2 *URL*

rev1	First revision number.	
rev2	Second revision number.	
URL		The SVN URL corresponding to the resource.

Chapter 4

Text Editor Specific Actions

Topics:

The Text mode of the editor panel provides the usual actions specific for a plain text editor: undo / redo, copy / paste, find / replace, etc. These actions are executed from the menu bar or toolbar and also by invoking their usual keyboard shortcuts.

Undoing and Redoing User Actions

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

Copying and Pasting Text

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

Finding and Replacing Text in the Current File

This section explains how to use the find and replace features of the application.

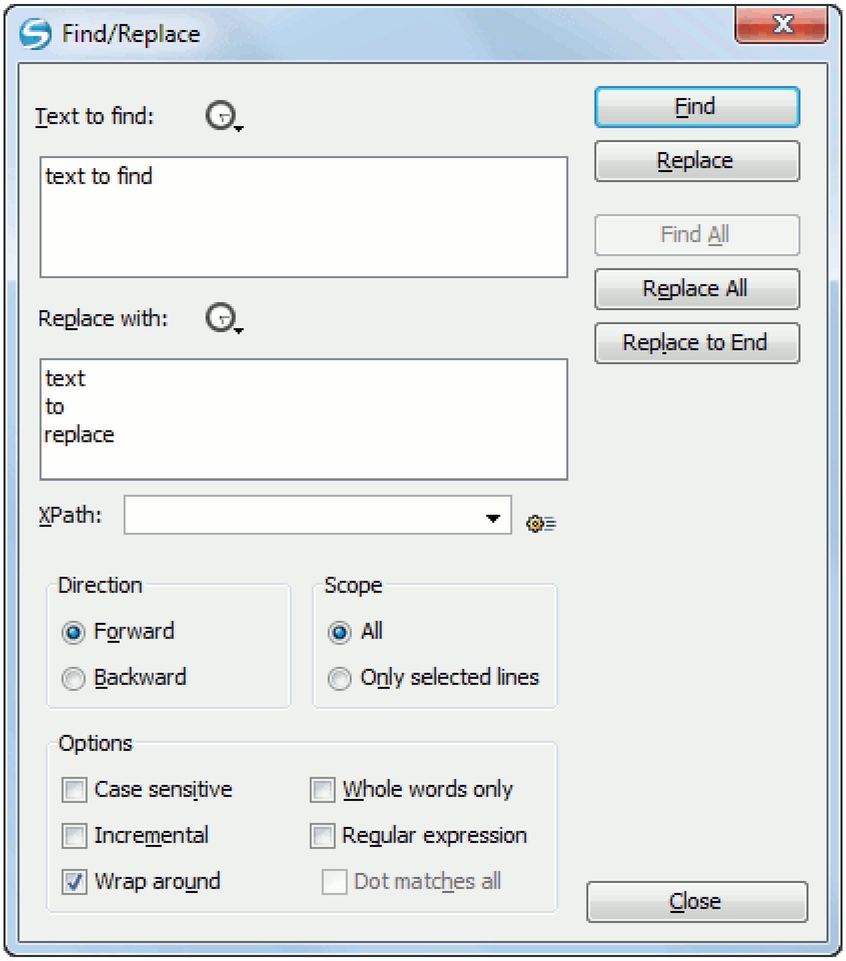
The Find / Replace Dialog

The **Find / Replace** dialog is opened from menu **Edit** > **Find / Replace...** **(Ctrl+F)** . It enables you to define search and replace operations on the current document. The find works on multiple lines, which means a find match can cover characters on more than one line. Special characters like newline and tab can be inserted using the contextual menu.

To insert a new line in the find or replace text area, press **(CTRL + Enter)** instead of **(Enter)**. The replace operation can bind Perl 5 regular expression group variables ($1, $2, etc.) from the find match. For example to replace the tag with attributes called `tag-name` with the tag `tag-name1` use *<tag-name(\s+)(.*)>* in the **Text to find** area and *<tag-name1$1$2>* in the **Replace with** area.

The following actions can be executed in the **Find / Replace** dialog:

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



Text to find:

text to find

Replace with:

text
to
replace

XPath:

Direction
- ● Forward
- ○ Backward

Scope
- ● All
- ○ Only selected lines

Options
- □ Case sensitive
- □ Whole words only
- □ Incremental
- □ Regular expression
- ☑ Wrap around
- □ Dot matches all

Figure 56: Find / Replace Dialog

The dialog contains the following options:

- **Text to find** - The target character string to search for. You can search for Unicode characters specified in the `\uNNNN` format. Also, hexadecimal notation (`\xNNNN`) and octal notation (`\0NNNN`) can be used. In this case you have to check the **Regular expression** checkbox. For example, to search for a space character you can use the `\u0020` code.
- **Replace with** - The character string with which to replace the target. The string for replace can be on a line or on multiple lines. Special characters like newline and tab can be inserted using the contextual menu. It may contain Perl 5 regular expression group markers, only if the search expression is a regular expression and the **Regular expression** checkbox is checked.

  Note: If the execution of the regular expression does not end in about five seconds, the application displays a dialog that allows you to interrupt the operation.

  Unicode characters can also be used in the **Replace with** area.
- The history buttons - The last find and replace operations history is available using the history buttons from the top of the find and replace text areas.
- **Direction** - Specifies if the search direction is from current position to end of file (**Forward** direction) or to start of file (**Backward** direction).
- **Scope** - Specifies if the search is executed on all file or only on the lines that were selected when the dialog was invoked. If the selection was on a single line the search is executed on the whole file (by default the **All** option is selected).

- **Find** - Executes a find operation for the next occurrence of the target. It stops after highlighting the find match in the editor panel.
- **Replace** - Executes a replace operation for the target followed by a find operation for the next occurrence.
- **Replace All** - Executes a replace operation in the entire scope of the document.
- **Replace to End** - Executes a replace operation starting from current target until the end of the document, in the direction specified by the current selection of the **Direction** switch (**Forward** or **Backward**).
- **Case sensitive** - When checked, operations are case-sensitive.
- **Whole words only** - When checked, only whole occurrences of a word will be included in the operation.
- **Incremental** - When checked, the search operation is started every time you type or delete a letter in the **Text to find** text box.
- **Regular expression** - When checked, it allows you to use regular expressions in Perl 5 syntax. A content completion assistant window is available to help you edit regular expressions. It is activated every time you type \(backslash key) or on-demand if you press **Ctrl-Space**.
- **Dot matches all** - A dot used in a regular expression matches also end of line characters.
- **Wrap around** - Continues the find from the start (end) of the document after reaching the end (start) when the search is in forward (backward) direction.

Keyboard Shortcuts for Finding the Next and Previous Match

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

- **Find** > **Find Next** **(F3)** - Performs another search in forward direction using the last search configuration.
- **Find** > **Find Previous** **(Shift+F3)** - Performs another search in backward direction using the last search configuration.

Chapter 5

Configuring the Application

Topics:

This chapter presents all the user preferences that allow you to configure the application .

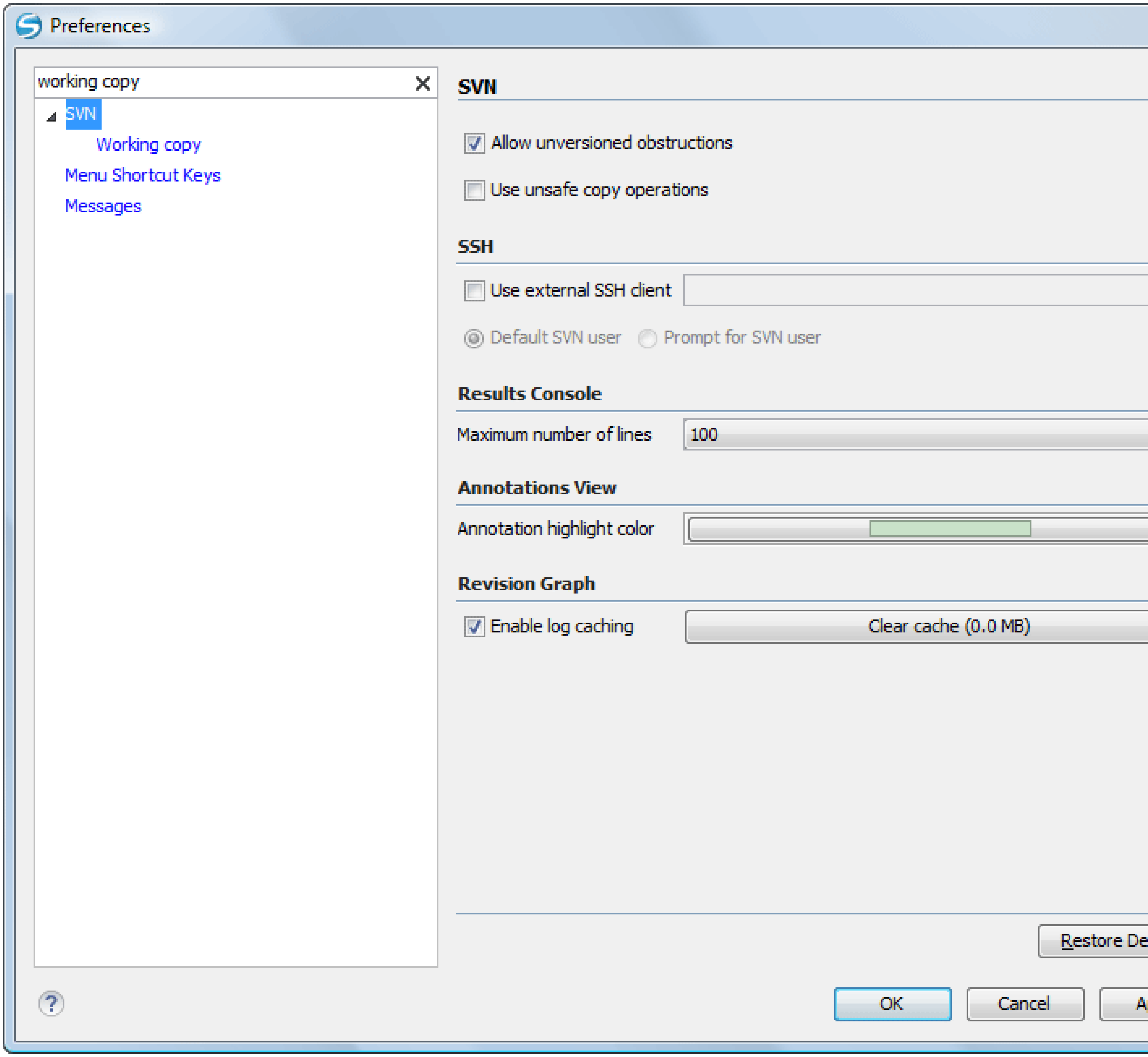
Importing / Exporting Global Options

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

Preferences

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

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



Preferences

working copy

- SVN
  - Working copy
- Menu Shortcut Keys
- Messages

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

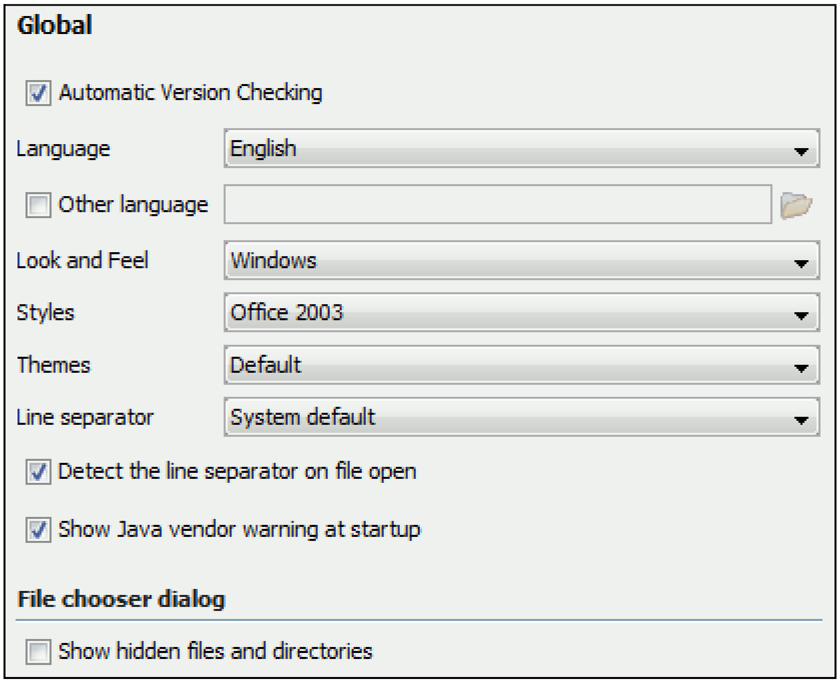
Figure 57: The Search field from the Preferences dialog

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

If you don't know how to use a specific preference that is available in any **Preferences** panel or what effect it will have you can open a help page about the current panel at any time pressing the help button ? located in the left bottom corner of the dialog or pressing the F1 key.

Global

The **Global** preferences panel is opened from menu **Options** > **Preferences** > **Global** .



Global

☑ Automatic Version Checking

Language: English

□ Other language

Look and Feel: Windows

Styles: Office 2003

Themes: Default

Line separator: System default

☑ Detect the line separator on file open

☑ Show Java vendor warning at startup

File chooser dialog

□ Show hidden files and directories

Figure 58: The Global preferences panel

The following user preferences are av available in this panel:

- **Automatic Version Checking** - When enabled, checks the availability of new Syncro SVN Client versions at *http://www.syncrosvnClient.com/*.
- **Language** - The application supports a number of languages for localization of the GUI. Go to menu **Options** > **Preferences** > **Global** and select the **Language** drop-down list to display the language choices.

  Note: After restarting the application, if some GUI labels are not rendered correctly (for example Chinese or Korean characters) you will need to install the corresponding language pack from your OS installation kit (for example the East-Asian language pack).

- **Other language** - To change the user interface language of Syncro SVN Client you must set here the properties file with all the user interface messages and labels translated to your preferred language. After setting the file you have to restart Syncro SVN Client in order to change the user interface language to your preferred language.
- **Look and Feel** - Use this option to change graphic style (look and feel) of the GUI.
- **Styles** - On Windows there are available the following styles:
  - Office 2003
  - Vsnet
  - Eclipse
  - Xerto
  - Default

  Note: After changing the style you have to restart the application in order for the modification to take effect.

  On Linux there are available the following styles:

  - Eclipse
  - Default

  Note: After changing the style you have to restart the application in order for the modification to take effect.

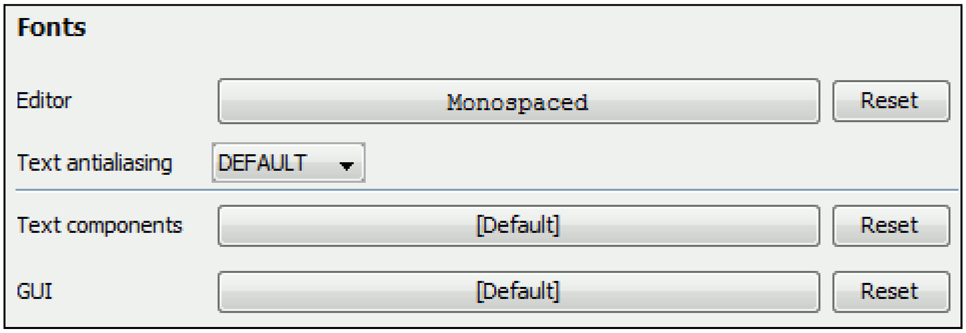
  On Mac OS X this option is not available.

- **Themes** - On Windows this option is enabled only for the **Office 2003** and **Default** styles. In these cases, the following themes are available:
  - Normal Color
  - Home Stead
  - Metallic
  - Default
  - Gray

  On Linux and Mac OS X this option is not available.
- **Line separator** - This option defines the line separator. The **System Default** choice sets the line separator of the platform.
- **Detect the line separator on file open** - When this option is checked the editor will detect the line separator when the edited file is loaded and it will use it when the file is saved. The new files are saved using the line separator defined by the **Line separator** option.
- **Show Java vendor warning at startup** - Sun Microsystems Java VM (on Windows and Linux) or Apple Computer Java VM (on Mac OS X) is recommended for running Syncro SVN Client . If a different VM is used, then a warning is displayed. This option allows the user to choose whether the warning dialog is shown or not.
- **Show hidden files and directories** - Shows system hidden files and folders in the file browser dialog and the folder browser dialog. This setting is not available on Mac OS X.

Fonts

The **Fonts** preferences panel is opened from menu **Options** > **Preferences** > **Fonts** .



Fonts

Editor: Monospaced

Text antialiasing: DEFAULT

Text components: [Default]

GUI: [Default]

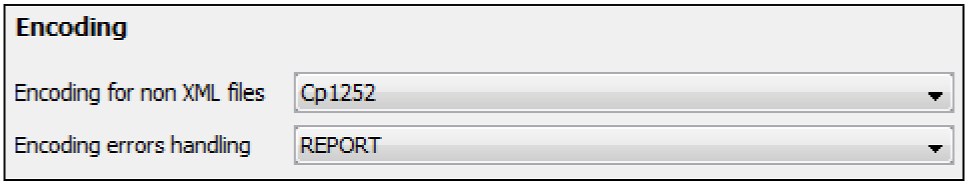
Figure 59: The Fonts preferences panel

The fonts that can be configured in Syncro SVN Client are the following:

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

Encoding

The **Encoding** preferences panel is opened from menu **Options** > **Preferences** > **Encoding**



Encoding

Encoding for non XML files: Cp1252

Encoding errors handling: REPORT

Figure 60: The Encoding preferences panel

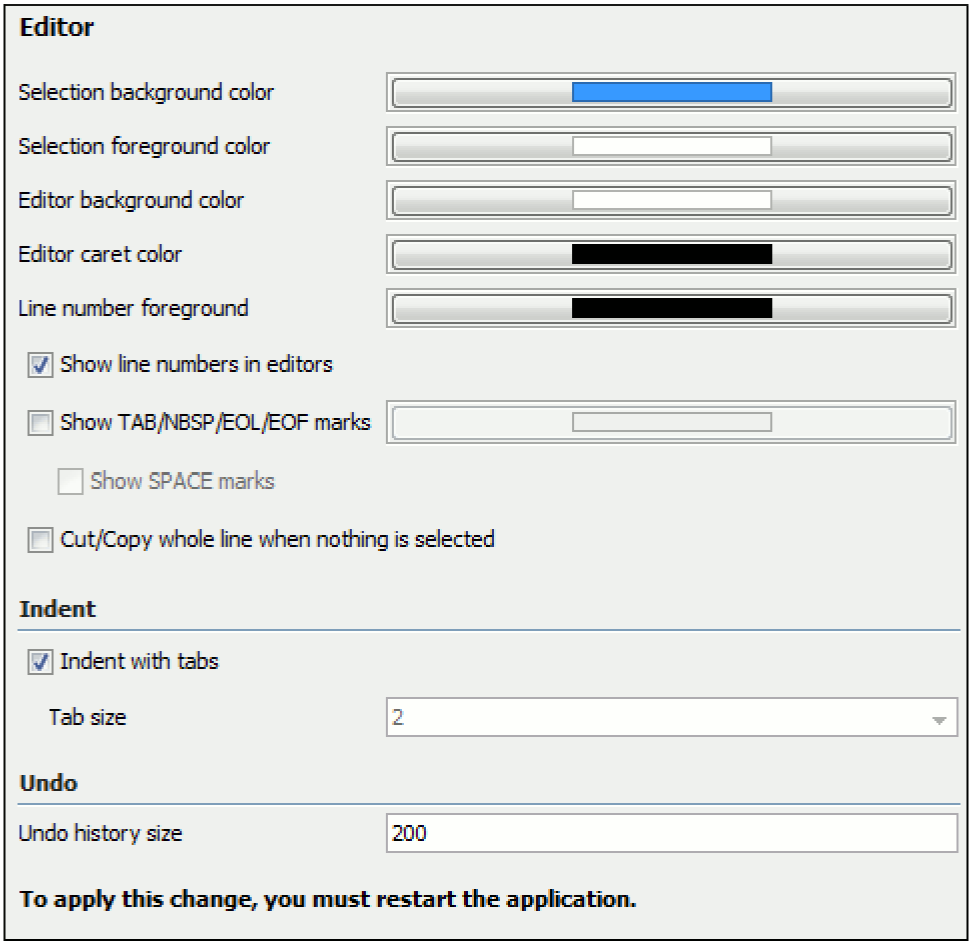
The encoding preferences are the following:

- **Encoding for non XML files** - The default encoding to be used when opening non XML documents. This is necessary because non XML files have a large variety of formats and there is no standard mechanism for declaring the encoding that should be used for opening and saving the file. In case of XML files the encoding is usually declared at the beginning of the file in a special declaration or it assumes the default value UTF-8.
- **Encoding errors handling** - This option defines how to handle characters that cannot be represented in the specified encoding of the document when the document is opened. The available options are:
  - **REPORT** - Shows an error dialog with the character that cannot be represented in the specified encoding and allows the user to decide how to continue (ignore that character, replace it with a standard replacement character). This is the default option.
  - **IGNORE** - The character is ignored and it will not be included in the document displayed in the editor panel.
  - **REPLACE** - Replace the character with a standard replacement character. For example if the encoding is UTF-8 the replacement character has the Unicode code `FFFD`, and if the encoding is ASCII the character code is 63.

Editor

The **Editor** preferences panel is opened from menu **Options** > **Preferences** > **Editor** .

Use these options to configure the visual aspect of the text editor.



Editor

Selection background color 🟦

Selection foreground color ⬜

Editor background color ⬜

Editor caret color ⬛

Line number foreground ⬛

☑ Show line numbers in editors

☐ Show TAB/NBSP/EOL/EOF marks

☐ Show SPACE marks

☐ Cut/Copy whole line when nothing is selected

Indent

☑ Indent with tabs

Tab size: 2

Undo

Undo history size: 200

To apply this change, you must restart the application.

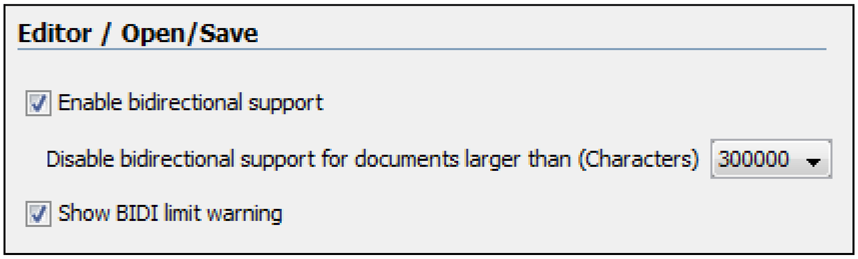
Figure 61: The Editor Preferences Panel

The following options are available in this panel:

- **Selection background color** - Background color of selected text.
- **Selection foreground color** - Text color of selected text.
- **Editor background color** - Background color of the editor and also of the Diff Files' editors.
- **Editor caret color** - Customize the caret color.
- **Line number foreground** - Foreground color for the line numbers displayed at the right of editor panel.
- **Show line numbers in editor** - Enables the line numbers column located in the left part of the editing space. When unchecked, line numbers option is disabled.
- **Show TAB/NBSP/EOL/EOF marks** - Marks the TAB/NBSP/EOL/EOF using small icons, for a better visualization of the document. Also set the marks color.
- **Show SPACE marks** - Marks the SPACE characters with a dot.
- **Indent with tabs** - When checked set the indent to a tab unit. When unchecked, the indent measures as many spaces as needed to go to the next tab stop position. The maximum number of space characters is defined by the **Tab size** option.
- **Tab size** - Sets the number of spaces or the tab size that equals a single indent. The *Indent* can be spaces or a tab, select the preference using the **Indent With Tabs** option. If set to 4, one tab will equal 4 white spaces or 1 tab with size of 4 characters depending on which option was set in the **Indent With Tabs** option.

Open / Save

The **Open / Save** preferences panel is opened from menu **Options** > **Preferences** > **Editor** > **Open / Save** .



Editor / Open/Save

☑ Enable bidirectional support

Disable bidirectional support for documents larger than (Characters) 300000

☑ Show BIDI limit warning

Figure 62: The Open / Save Preferences Panel

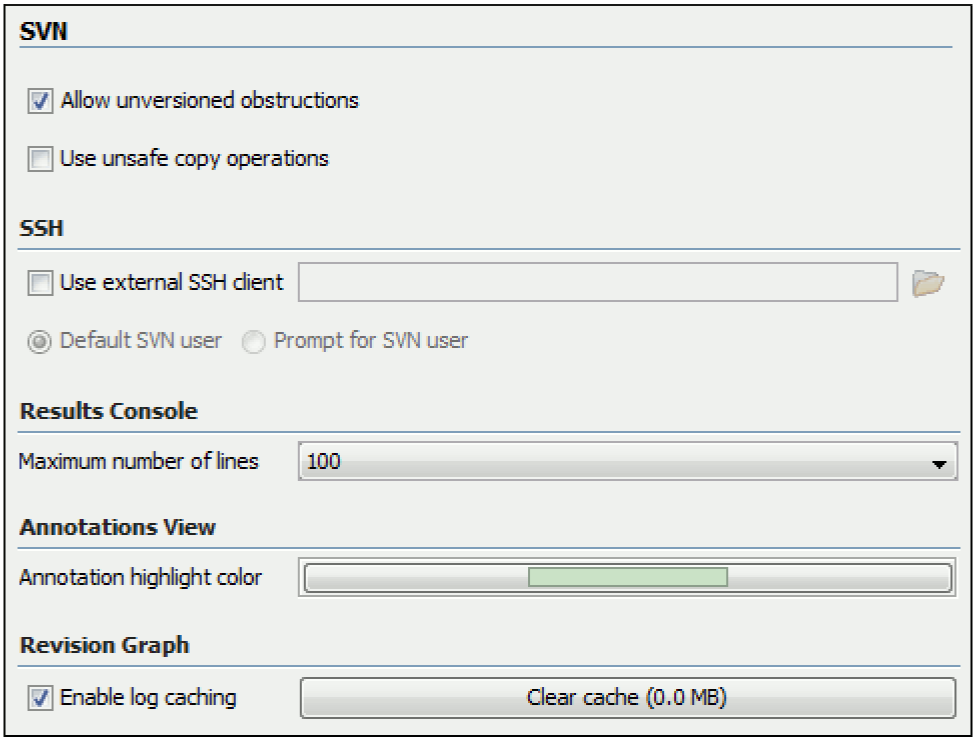
The preferences related with opening and saving documents are the following:

- **Characters limit for bidirectional text documents** - Specifies the characters limit for bidirectional text documents. If the total number of characters found in a document exceeds this limit, the bidirectional support is disabled.
- **Show BIDI limit warning** - If checked, a warning dialog is shown when the opened file which contains bidirectional characters is too large and bidirectional support is disabled to improve application performance.
- **Consider application bundles to be directories when browsing** - This option is available only on the Mac OS X platform. When checked the file browser dialog allows browsing inside an application bundle as in a regular folder. When unchecked the file browser dialog does not allow browsing inside an application bundle, as the Finder application does on Mac OS X. The same effect can be obtained by setting the property *apple.awt.use-file-dialog-packages* to true or false in the `Info.plist` descriptor file of the Syncro SVN Client application by adding two lines in this descriptor file:

```
<key>apple.awt.use-file-dialog-packages</key>
<string>false</string>
```

SVN

The **SVN** preferences panel is opened from menu **Options** > **Preferences** > **SVN** and it is the place where the user preferences for the embedded SVN client tool are configured. 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 63: The SVN Preferences Panel

The SVN preferences are the following:

- **Enable symbolic link support** (*available only on Mac OS X and Linux*) - Subversion has the ability to put a symbolic link under version control, via the usual SVN `add` command. The Subversion repository has no internal concept of a symbolic link, it stores a versioned symbolic link as an ordinary file with a `svn:special` property attached. The SVN client (on Unix) sees the property and translates the file into a symbolic link in the working copy.

  Note: Windows file systems have no symbolic links, so a Windows client won't do any such translation: the object appears as a normal file.

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

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

- **Allow unversioned obstructions** - This option 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.
- **SSH** - Here you can specify the command line for an external SSH client which will be used when connecting to a SVN+SSH repository. Absolute paths are recommended for the SSH client executable and the file paths given as arguments (if any). Depending on the SSH client used and your SSH server configuration you may need to specify

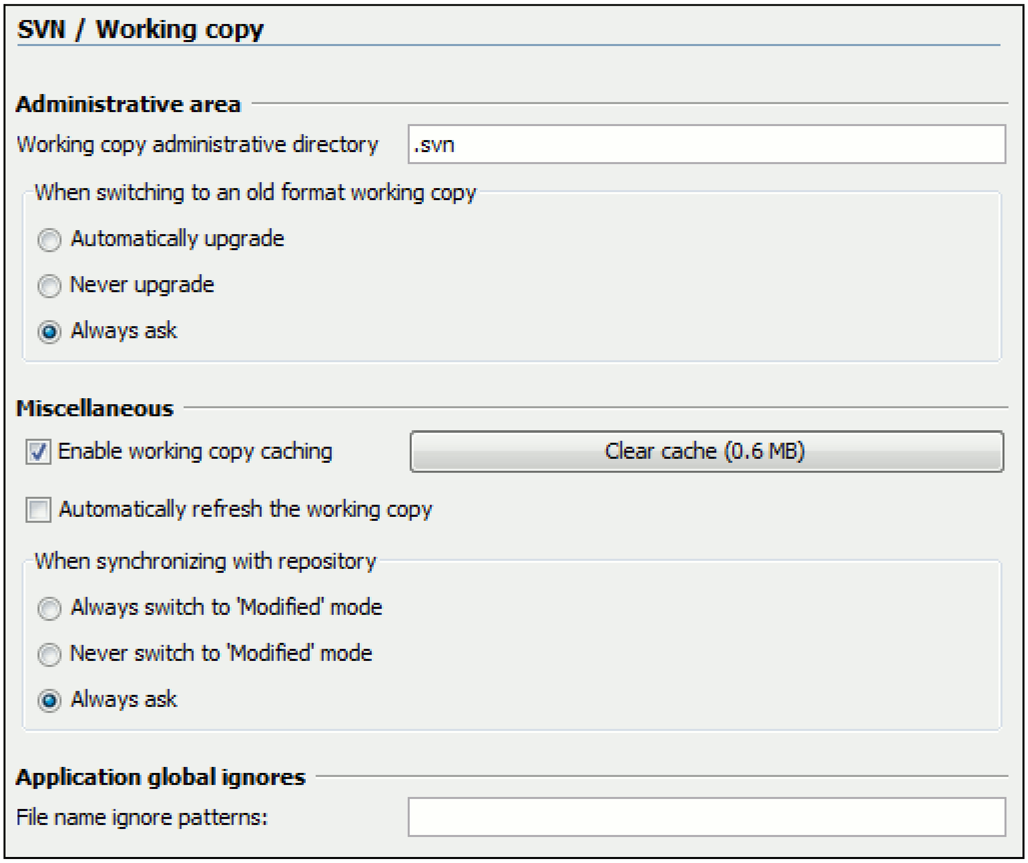
in the command line the user name and / or private key / passphrase. Here you can also choose if the default user name (the same user name as the SSH client user) will be used for SVN repository operations or you should be prompted for a SVN user name whenever SVN authentication is required. For example 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** - Here you can specify the maximum number of lines displayed in the **Console** view.
- **Annotations View** - Here you can set the color used for highlighting in the editor panel all the changes contributed to a resource by the revision selected in *the* ***Annotations*** *view*.
- **Revision Graph** - Here you can enable caching for the action of computing a revision graph. When a new revision graph is requested one of the caches from the previous actions may be used which will avoid running the whole query again on the SVN server. If a cache is used it will finish the action much faster.

Working Copy

The **Working Copy** panel is open from menu **Options** > **Preferences** > **SVN** > **Working Copy** and it contains options that are specific to SVN working copies.



SVN / Working copy

Administrative area

Working copy administrative directory: .svn

When switching to an old format working copy

○ Automatically upgrade
○ Never upgrade
● Always ask

Miscellaneous

☑ Enable working copy caching

☐ Automatically refresh the working copy

When synchronizing with repository

○ Always switch to 'Modified' mode
○ Never switch to 'Modified' mode
● Always ask

Application global ignores

File name ignore patterns:

Figure 64: The Working Copy Panel

These options are the following:

- **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 switching to an old format working copy** - You can instruct Syncro SVN Client to do one of the following:
  - **Automatically upgrade** - Older format working copies are upgraded to the newest known format.
  - **Never upgrade** - Older format working copies are left untouched. No attempt to upgrade the format is made.

  - **Always ask** - You are notified when such a working copy is used and you are allowed to choose what action to be taken - to upgrade or not the format of the current working copy.

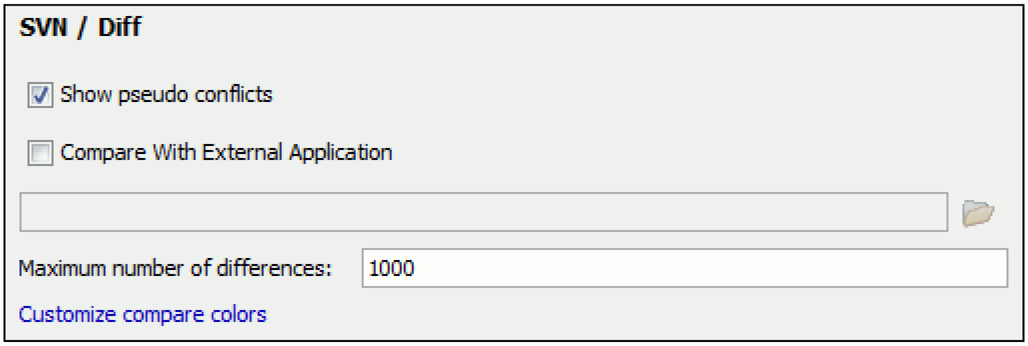
- **Enable working copy caching** - If checked, the content of the working copies is cached for refresh operations.
- **Automatically refresh the working copy** - If checked, the working copy is refreshed from cache. Only the new changes (modifications with a date/time that follows the last refresh operation) are refreshed from disk. Enabled by default.
- **When synchronizing with repository** - The action that will be executed automatically after the **Synchronize** action. The possible actions are:

  - **Always switch to 'Modified' mode** - The **Synchronize** action is followed automatically by a switch to **Modified** mode of **Working Copy** view, if **All Files** mode is currently selected.
  - **Never switch to 'Modified' mode** - Keeps the currently selected view mode unchanged.
  - **Always ask** - The user is always asked if he wants to switch to **Modified** mode.

- **Application global ignores** - Allows setting file patterns that may include the * and ? wildcards for unversioned files and folders that must be ignored when displaying the working copy resources in *the **Working Copy** view*.

Diff

The **Diff** preferences panel is opened from menu **Options** > **Preferences** > **Diff** and it allows you to set the compare options for SVN client.



SVN / Diff

☑ Show pseudo conflicts

☐ Compare With External Application

Maximum number of differences: 1000

Customize compare colors

Figure 65: The SVN Diff Preferences Panel

The SVN diff preferences are the following:

- **Show pseudo conflicts** - It allows you to specify if you want to see pseudo-conflicts in *the **Compare** view*. A pseudo conflict occurs when two developers make the same change, for example when both add or remove the same line of code.
- **Compare With External Application** - You can specify an external application to be launched for compare operations in the following cases:

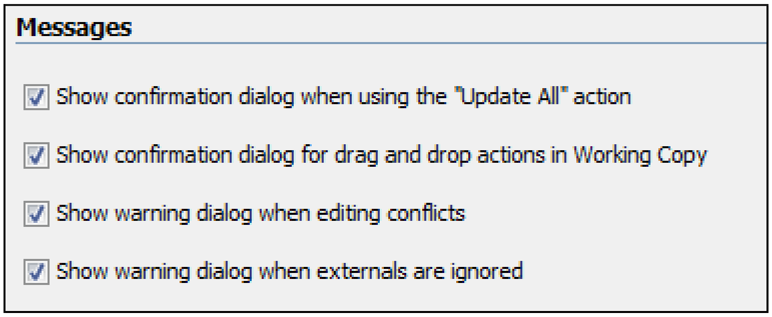
  - when two history revisions are compared
  - when the working copy file is compared with a history revision
  - when *a conflict is edited*

  The parameters ${firstFile} and ${secondFile} specify the positions of the two compared files in the command line for the external diff application. The parameter ${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison: the working copy version of a file is compared with the repository version, with the BASE revision (the latest revision read from the repository by an Update or Synchronize operation) being the common ancestor of these two compared versions.

- **Maximum number of differences** - You can change the maximum number of differences allowed in the view.

Messages

The **Messages** preferences panel is opened from menu **Options** > **Preferences** > **Messages** and allows disabling 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

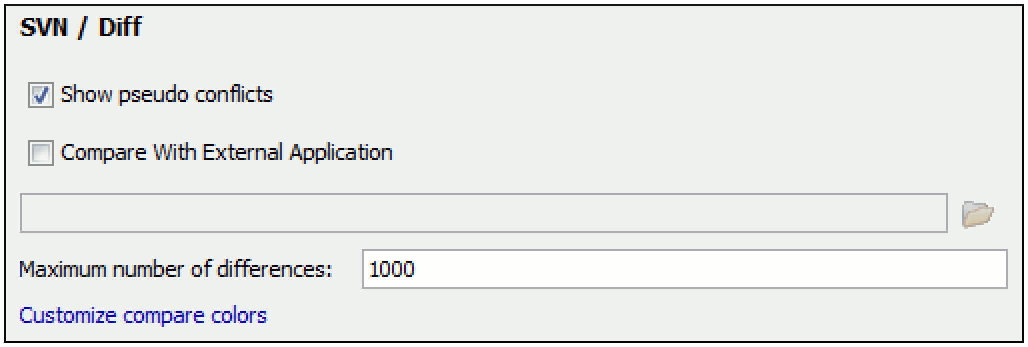
☑ Show warning dialog when externals are ignored

Figure 66: 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*).

Diff

The **Diff** preferences panel is opened from menu **Options** > **Preferences** > **Diff** and it allows you to set the compare options for SVN client.



SVN / Diff

☑ Show pseudo conflicts

□ Compare With External Application

Maximum number of differences: 1000

Customize compare colors

Figure 67: The SVN Diff Preferences Panel

The SVN diff preferences are the following:

- **Show pseudo conflicts** - It allows you to specify if you want to see pseudo-conflicts in *the **Compare** view*. A pseudo conflict occurs when two developers make the same change, for example when both add or remove the same line of code.
- **Compare With External Application** - You can specify an external application to be launched for compare operations in the following cases:
  - when two history revisions are compared

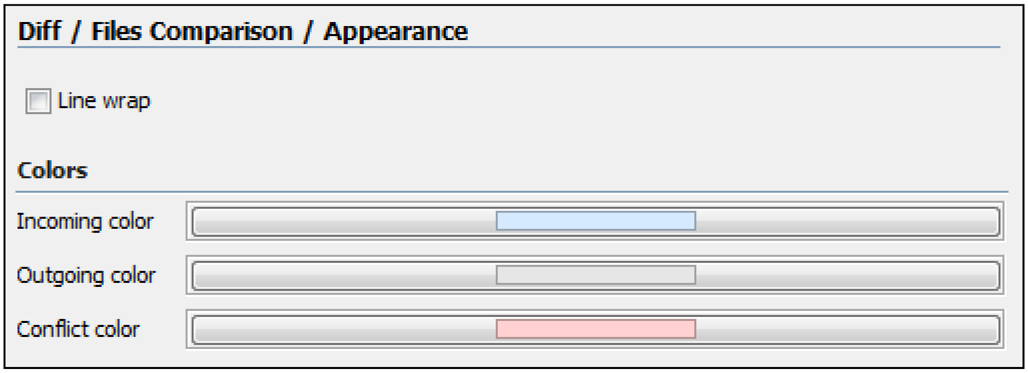
  - when the working copy file is compared with a history revision
  - when *a conflict is edited*

  The parameters ${firstFile} and ${secondFile} specify the positions of the two compared files in the command line for the external diff application. The parameter ${ancestorFile} specifies the common ancestor (that is, the BASE revision of a file) in a three-way comparison: the working copy version of a file is compared with the repository version, with the BASE revision (the latest revision read from the repository by an Update or Synchronize operation) being the common ancestor of these two compared versions.

- **Maximum number of differences** - You can change the maximum number of differences allowed in the view.

Appearance

The **Files Comparison / Appearance** preferences panel is opened from menu **Options** > **Preferences** > **Diff** > **Files Comparison** > **Appearance** and offers the following options:



Diff / Files Comparison / Appearance

□ Line wrap

Colors

Incoming color

Outgoing color

Conflict color

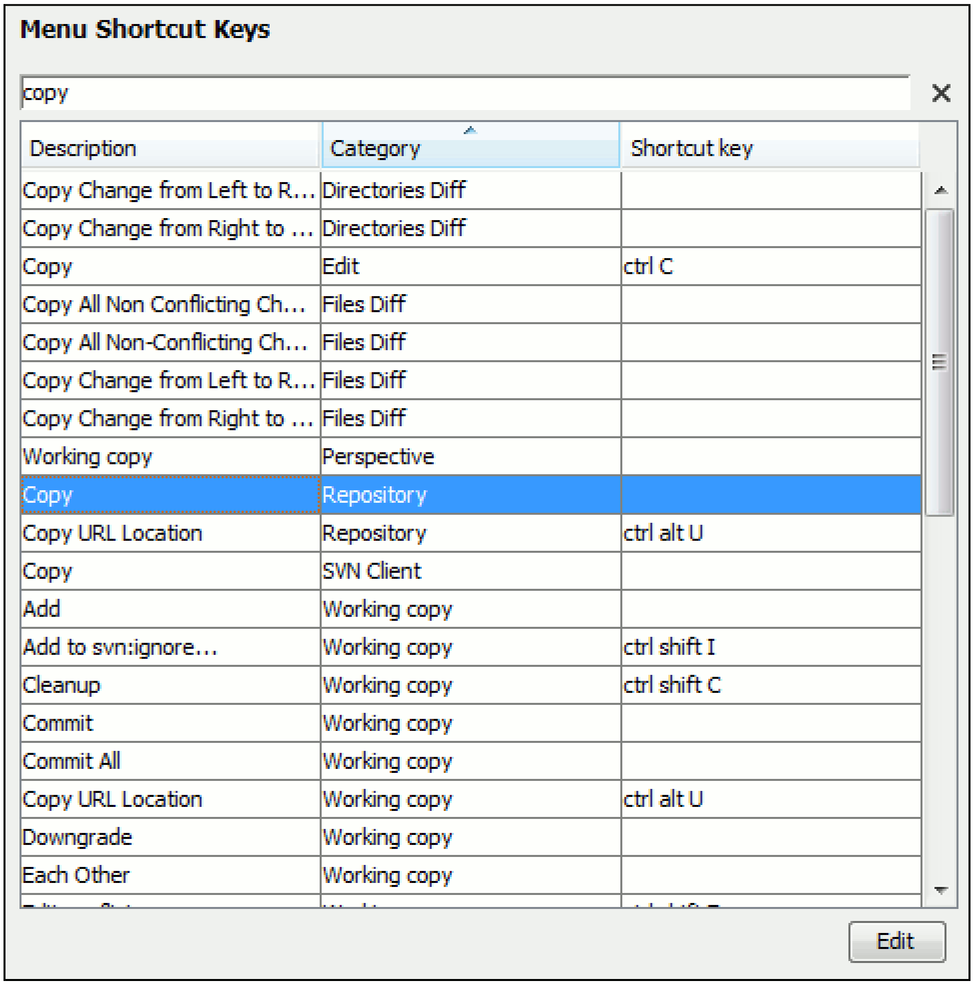
Figure 68: Files Comparison Appearance Preferences Panel

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

Menu Shortcut Keys

The **Menu Shortcut Keys** preferences panel is opened from menu **Options** > **Preferences** > **Menu Shortcut Keys** . It allows configuring in one place the keyboard shortcuts available for the menu items on the menus of Syncro SVN Client . The current shortcuts assigned to menu items are displayed in the following table.

You can find an operation in the table using the filter field that can search by the operation's description, category or shortcut key:



Menu Shortcut Keys

copy

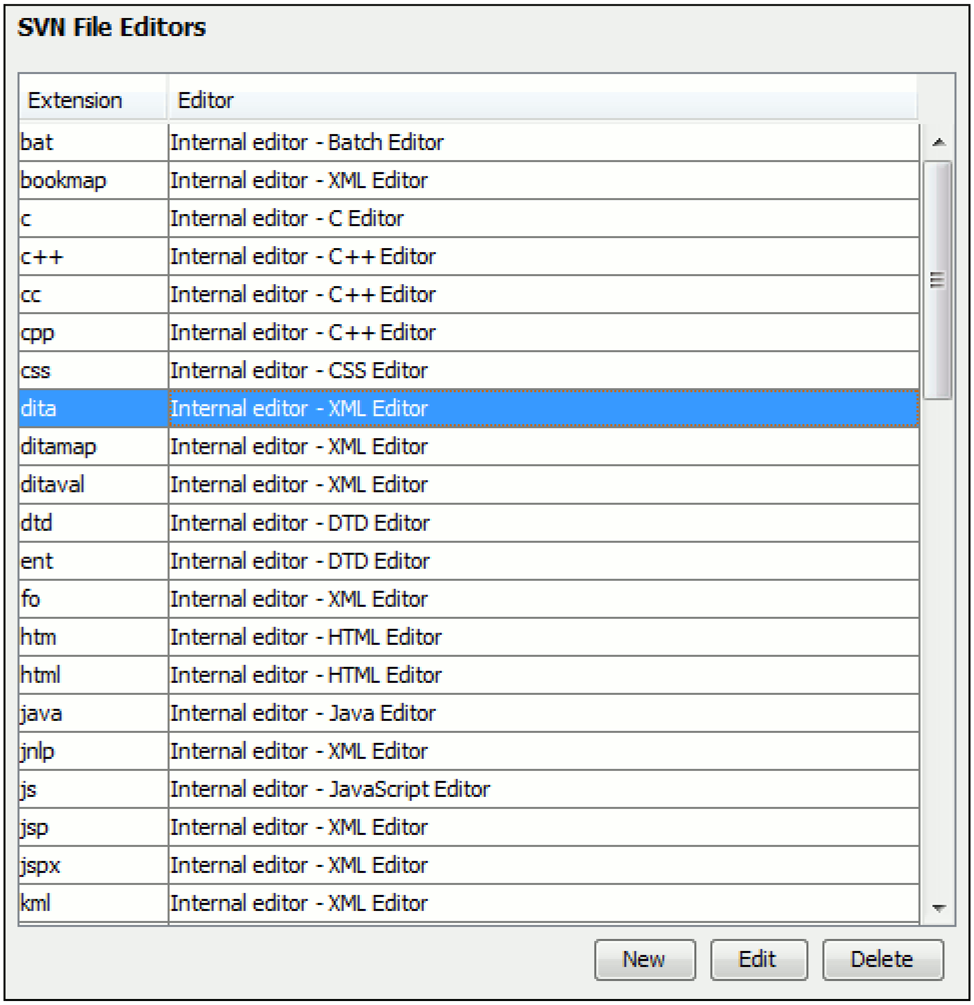
Description	Category	Shortcut key
Copy Change from Left to R...	Directories Diff	
Copy Change from Right to ...	Directories Diff	
Copy	Edit	ctrl C
Copy All Non Conflicting Ch...	Files Diff	
Copy All Non-Conflicting Ch...	Files Diff	
Copy Change from Left to R...	Files Diff	
Copy Change from Right to ...	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 69: The Menu Shortcut Keys Preferences Panel

- **Description** - A short description of the menu item operation.
- **Category** - The shortcuts are classified in categories for easier management. For example the **Cut** operation for the source view is distinguished from the tree view one by assigning it to a separate category.
- **Shortcut key** - The keyboard shortcut that launches the operation. Double-clicking on a table row or pressing the **Edit** button allows the user to register a new shortcut for the operation displayed on that row.

SVN File Editors

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

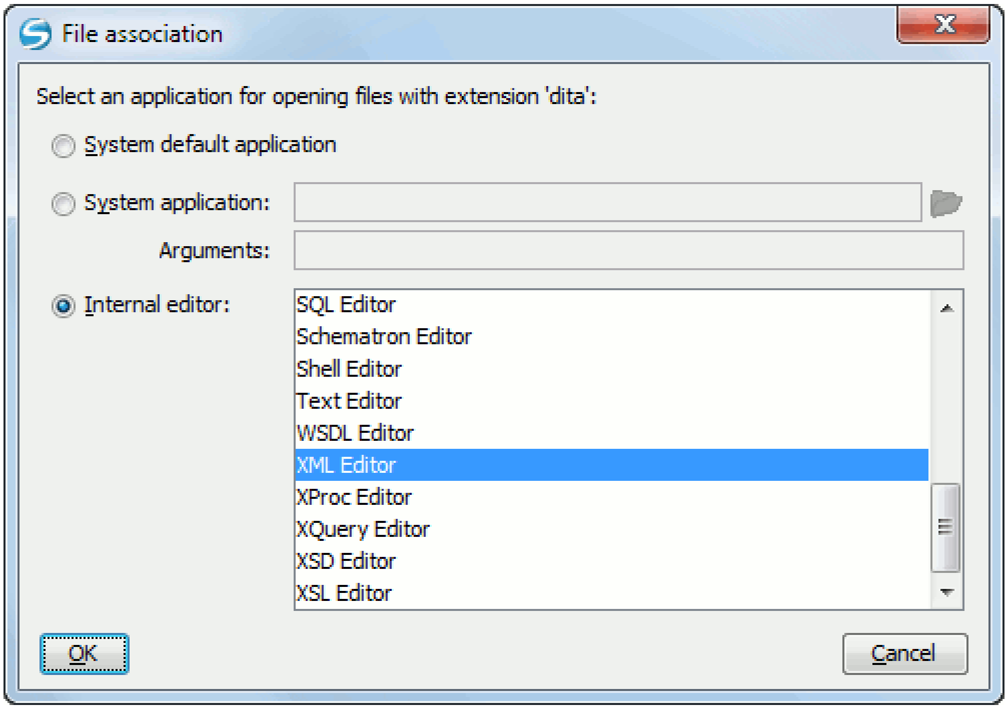


SVN File Editors

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 70: 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 71: 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.

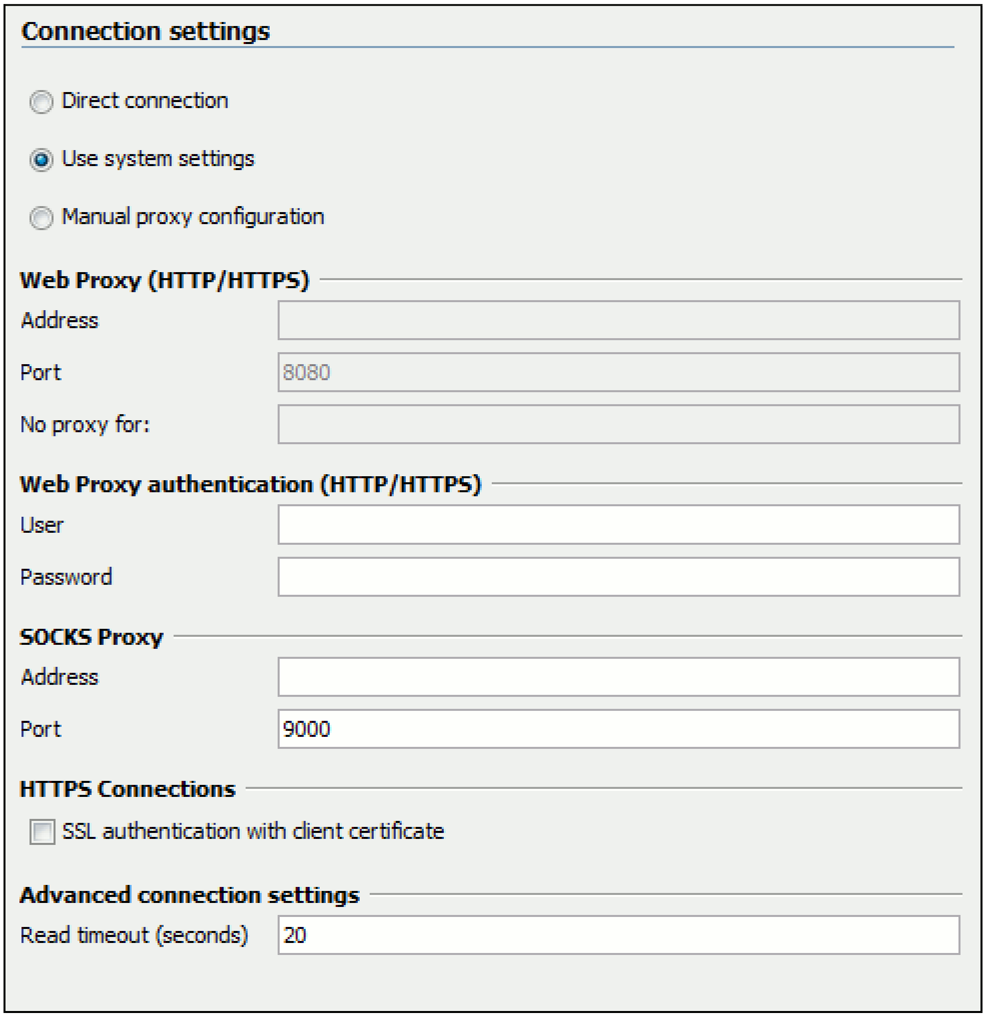
If a file type is associated with an internal editor other than an XML editor type then the encoding set in *the preference* ***Encoding for non XML files*** is used for opening and saving a file of that type. This is necessary because in case of XML files the encoding is usually declared at the beginning 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.

HTTP / Connection settings

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 don't know the proxy parameters, please consult your network administrator.

You can open the Connection settings panel from menu **Options** > **Preferences** > **Connection settings** .

The Connection settings Preferences Panel



Connection settings

○ Direct connection

● Use system settings

○ Manual proxy configuration

Web Proxy (HTTP/HTTPS)

Address

Port 8080

No proxy for:

Web Proxy authentication (HTTP/HTTPS)

User

Password

SOCKS Proxy

Address

Port 9000

HTTPS Connections

□ SSL authentication with client certificate

Advanced connection settings

Read timeout (seconds) 20

Complete the dialog as follows:

- **Direct connection** - If checked, the HTTP(S) connections go directly to the target host without going through a proxy server.
- **Use system settings** - If checked, the HTTP(S) connections go through the proxy server set in the operating system. For example on Windows the proxy settings are the ones used by Internet Explorer.

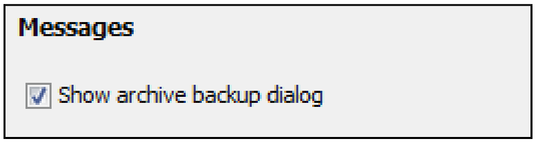
  ⚠ **Attention:** The system settings for the proxy cannot be read correctly from the operating system on some Linux systems. The system settings option should work properly on Gnome based Linux systems but it does not work on KDE based ones as *the Java virtual machine does not offer the necessary support yet*.

- **Manual proxy configuration** - If checked, the HTTP(S) connections go through the proxy server specified in the fields **Address** and **Port** of the section **Web Proxy (HTTP / HTTPS)**. Also this section specifies the hosts to which the connections must not go through a proxy server in the field **No proxy for**.
- **Web Proxy authentication (HTTP / HTTPS)** - In this section you set the user and password necessary for authentication with the proxy server. The user and password set here will be used both in case of manual proxy configuration and in case of system settings selected above.
- **SOCKS Proxy** - In this section you set host and port of a SOCKS proxy through which all the connections must pass. If the **Address** field is empty the connections will use no SOCKS proxy.
- **SSL authentication with client certificate** - If checked and the SVN server accessed through the HTTPS protocol requires a digital certificate, the user is asked to specify a file path containing the digital certificate in the PKCS format for accessing that server.
- **Read Timeout (s)** - The period in seconds after which the application will consider a HTTP server is unreachable if it does not receive any response to a request sent to that server.

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

Messages

The **Messages** preferences panel is opened from menu **Options** > **Preferences** > **Messages** and allows disabling the following warning messages which may appear in the application:



Messages

☑ Show archive backup dialog

Figure 72: The Messages Preferences Panel

- **Show archive backup dialog** - If checked, a dialog will be shown allowing the user different backup options before modifying an archive's content.

Reset Global Options

To reset all global preferences to their default values you have to go to menu **Options** > **Reset Global Options** > **Reset Global Options** .

Chapter 6

Common Problems

Topics:

This chapter presents common problems that may appear when running the application and the solutions for these problems.

Special Characters Are Replaced With a Square in Editor

My file was created with other application and it contains special characters like é, ©, ®, etc. Why does Syncro SVN Client display a square for these characters when I open the file in Syncro SVN Client ?

You must set a font able to render the special characters in the ***Font** preferences*. If it is a text file you must set also *the encoding used for non XML files.* If you want to set a font which is installed on your computer but that font is not accessible in the **Font** preferences that means the Java virtual machine is not able to load the system fonts, probably because it is not a True Type font. It is a problem of the Java virtual machine and a possible solution is to copy the font file in the `[JVM-home-folder]/lib/fonts` folder. `[JVM-home-folder]` is the value of the property *java.home* which is available in the **System properties** tab of the **About** dialog that is opened from menu **Help** > **About** .

The Scroll Function of my Notebook's Trackpad is Not Working

I got a new notebook (Lenovo Thinkpad™ with Windows) and noticed that the scroll function of my trackpad is not working in Syncro SVN Client .

It is a problem of the Synaptics™ trackpads which can be fixed by adding the following lines to the `C:\Program Files\Synaptics\SynTP\TP4table.dat` file:

```
*,*,oxygen.exe,*,*,*,WheelStd,1,9
*,*,author.exe,*,*,*,WheelStd,1,9
*,*,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 1.5.0 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.