VisualSVN

Right thing. Done right.

Subversion Integration for Visual Studio

VisualSVN Team

VisualSVN: Subversion Integration for Visual Studio

VisualSVN Team

Introduction

VisualSVN is a transparent integration of the Subversion version control system to the Visual Studio development environment. VisualSVN allows you to take full control on any changes in the project that are made by you or your colleagues. With VisualSVN you can easily see the full history of modifications and restore previous versions of your project. So, you can treat it as a “smart time machine”.

VisualSVN is built on the base of the Subversion open source version control system that is a de-facto standard storage system for software projects. There are a lot of tools and services available for Subversion such as code review systems and hosting providers. With VisualSVN you can use these tools and services without limitations.

What is Subversion?

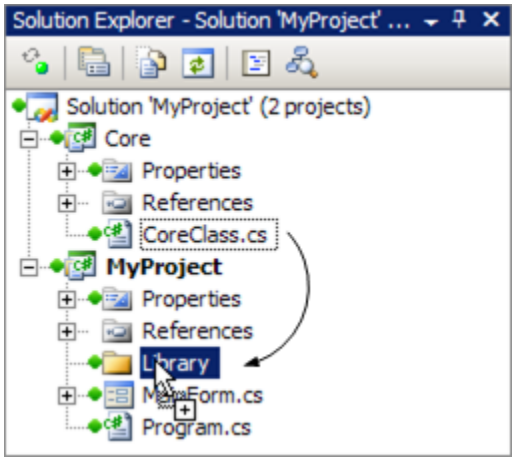
Subversion is a centralized multiuser version control system. Project files and full history of modifications are stored in a *repository*. The repository can be located both on a server, and on a local computer.

Each developer has his own copy of the project on the local computer, which is called the *working copy*. You can freely make any changes in your working copy, such as modifying files, adding new files, renaming files and folders and so on. And you don't even need a permanent connection to the repository. For example, you can work on your laptop on a plane. When a logical bundle of modifications is ready, you can upload it to the repository as a single atomic *commit*. Subversion automatically prevents overwrites of modifications that are made by another developer. Moreover, in most cases Subversion automatically merges modifications.

What is VisualSVN?

Subversion is a perfect version control system. VisualSVN makes Subversion easy to use for Visual Studio developers. By integrating Subversion to Visual Studio VisualSVN improves yours productivity and reduces the probability of routine mistakes.

VisualSVN is conceptually based on the principle of “transparent Subversion”. VisualSVN automatically reflects all actions to Subversion from Visual Studio that happen in the daily development cycles. For example, VisualSVN automatically marks all newly added files and folders as *added*. This transparent behavior allows you to concentrate on development, while VisualSVN takes care off versioning.



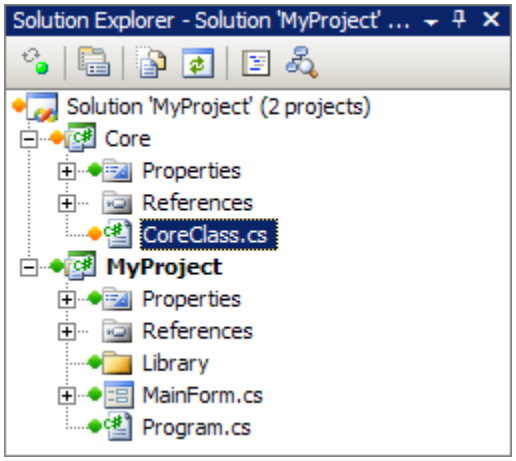
Moreover, VisualSVN allows you to manage files in the Solution Explorer in a transparent fashion. For example, you can drag-and-drop files between projects in the Solution Explorer and VisualSVN transparently reflects this operation to Subversion, with full history preservation. VisualSVN supports the

complete list of file management operations including addition, deletion, copying, renaming and drag-and-drop. Transparent file management allows you to refactor without pain.

Intuitive status

VisualSVN displays *traffic lights* status for almost all items in the Solution Explorer. There are only three statuses:

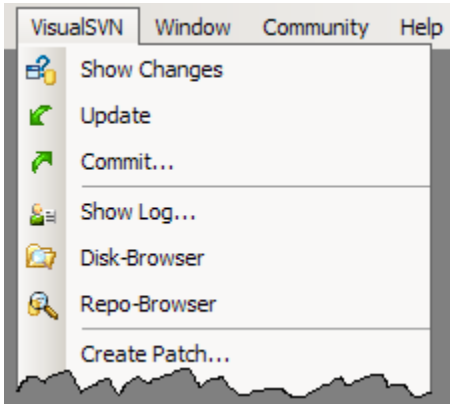
- green - there are no changes,
- yellow - there are some changes,
- red - there are conflicts or errors.



For each compound item such as folder or project VisualSVN displays the cumulative status. For example, there are some changes within a project's folder if the status of the corresponding item in the Solution Explorer is yellow. For the topmost solution item in the Solution Explorer VisualSVN displays the cumulative status of the entire working copy. For convenience, VisualSVN repeats the cumulative status of the entire working copy in the right corner of Visual Studio status bar.

Unlimited Subversion

In addition to such unique capabilities as transparent file management and intuitive status display, VisualSVN provides convenient access to all Subversion's commands using the mature and stable graphical user interface of TortoiseSVN, the de-facto standard Subversion client for the Windows platform. Thanks to this fact, you get unified access to Subversion both in Visual Studio, and Windows Explorer.



VisualSVN is a professional Subversion integration to the Visual Studio development environment. The functionality of VisualSVN completely covers the needs of the software development process including initial adding of a project under Subversion, refactoring, branching and merging.

Getting Started

Version control with VisualSVN is user friendly and straightforward. You don't need to have prior Subversion experience.

To get started with VisualSVN you should have the following software installed:

- Visual Studio 2003, 2005 or 2008 (all editions except Express are supported),
- TortoiseSVN (version 1.4.8 or higher is recommended),
- VisualSVN (version 1.4.0 or higher is recommended).

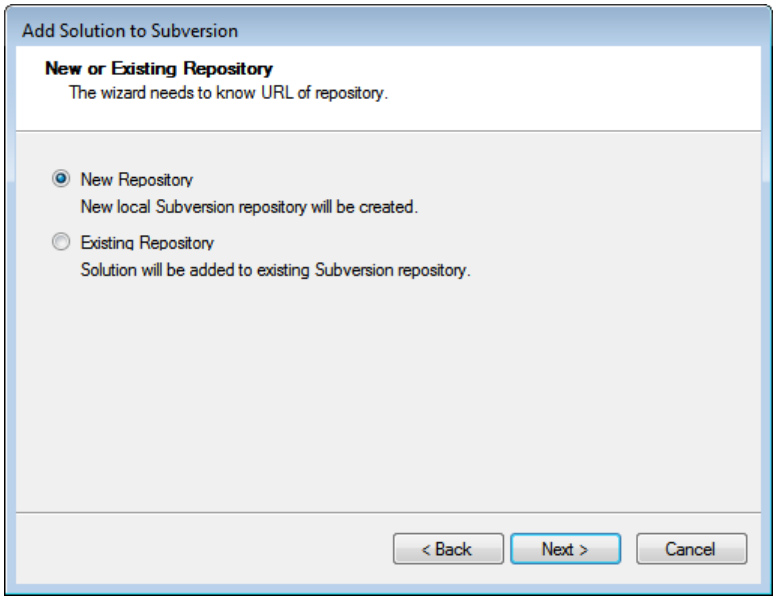
Add Solution to Subversion

First you need to place your solution under Subversion control. There are only two main steps to do this: execute the “Add Solution to Subversion” command and *commit* your solution to the repository.

Executing the “Add Solution to Subversion” command

Open or create new solution in Visual Studio. Then choose Add Solution to Subversion from the VisualSVN menu command and the wizard will open.

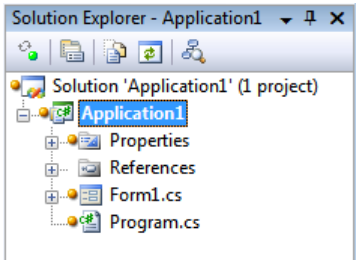
Now you should choose the repository where your code history will be stored. VisualSVN can add your code to a new or an existing repository. When VisualSVN creates new repository it creates the default structure with **branches**, **tags** and **trunk** folders. Initially your code will be stored in the **trunk** folder.



Now you choose the directory where your code will be stored. VisualSVN can add your code to a new or an existing repository. When VisualSVN creates new repository it creates the default repository structure with **branches**, **tags** and **trunk** folders. Initially your code will be stored in the **trunk** folder.

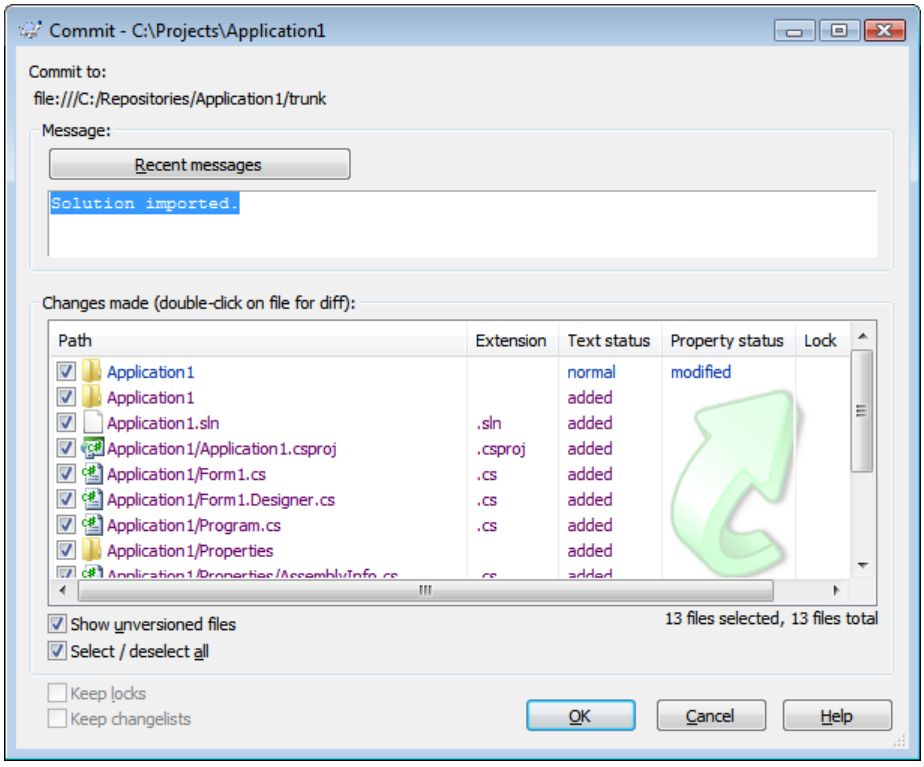
Please note that when adding your code to an existing repository you can choose any remote repository. For example, you can easily add your solution to a hosted repository provided by services like Google Code [http://code.google.com/]. Just enter the repository URL in the corresponding field in the dialog.

After this operation is finished all source files and folders in your solution will be marked as added to Subversion. Added files will be indicated by yellow icons in Solution Explorer. But all changes are still local and have not been sent to the target repository yet. This allows you to review the results of adding your solution to Subversion before the operation is finished.

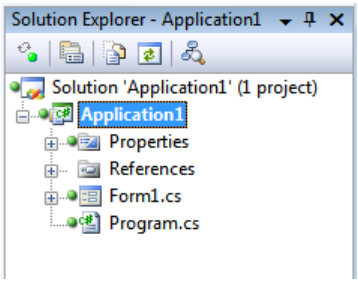


Commit Solution to Subversion

To commit your solution to Subversion, select VisualSVN ⇒ Commit menu item, enter a comment and click OK. The first versions of your source files will be sent to the target Subversion repository.



After you committed your solution to Subversion you should see green icons in Solutions Explorer indicating that your code is controlled by Subversion and you have no local changes. You are ready to work!


Solution Explorer - Application1
Solution 'Application1' (1 project)
Application1
Properties
References
Form1.cs
Program.cs

Understanding VisualSVN

Why Visual Studio integration?

There is a powerful cross-platform command line interface to all Subversion commands. It's very suitable for tasks such as writing build scripts. Moreover, there is a mature and stable graphical user interface for most of Subversion operations provided by TortoiseSVN. TortoiseSVN provides a very usable context dependent access to Subversion directly from the Windows Explorer.

However, most software developers work in an integrated development environment such as Visual Studio forcing them to work in two loosely coupled contexts: Visual Studio and Windows Explorer, for example. In addition to losing focus by frequent context switching, you can be faced with the following problems:

- **Daily-cycle routine errors**

  For example: The developer adds a new file to the project but forgets to mark the file as an “added file” in the working copy. During the commit this new file will not be sent to the repository, which causes failed builds.

- **Complicated file management**

  In modern software development cycles you often need to change the project layout and move a file from one project to another. To do this without losing the history you need to perform several operations in the Visual Studio and in the Windows Explorer. In addition to significant loss of time, the process is also error prone.

- **Dirty commits**

  It's easy to forget to check the status of the working copy before the starting a new task. This often causes *dirty commits*, when a single physical commit contains changes from several logical tasks.

VisualSVN provides transparent and comfortable access to Subversion directly from the Visual Studio and you get all the advantages of Subversion without facing the problems mentioned above.

Design principles of VisualSVN

The main goal of the VisualSVN is to transparently integrate Subversion into the Visual Studio. All Subversion functionality should be integrated smoothly and without significant limitations.

The main design principles of the VisualSVN are as follows:

- **Transparent source control**

  VisualSVN automatically and transparently reflects to Subversion all operations from Visual Studio that occur during daily software development cycles.

- **Genuine Subversion**

  VisualSVN doesn't introduce a new version control system, but makes the standard Subversion easy to access for Visual Studio developers.

- **Intuitive status display**

VisualSVN displays the status of all versioned items to allow developers to instantly determine the necessity of performing Subversion operations.

Basic Work Cycle

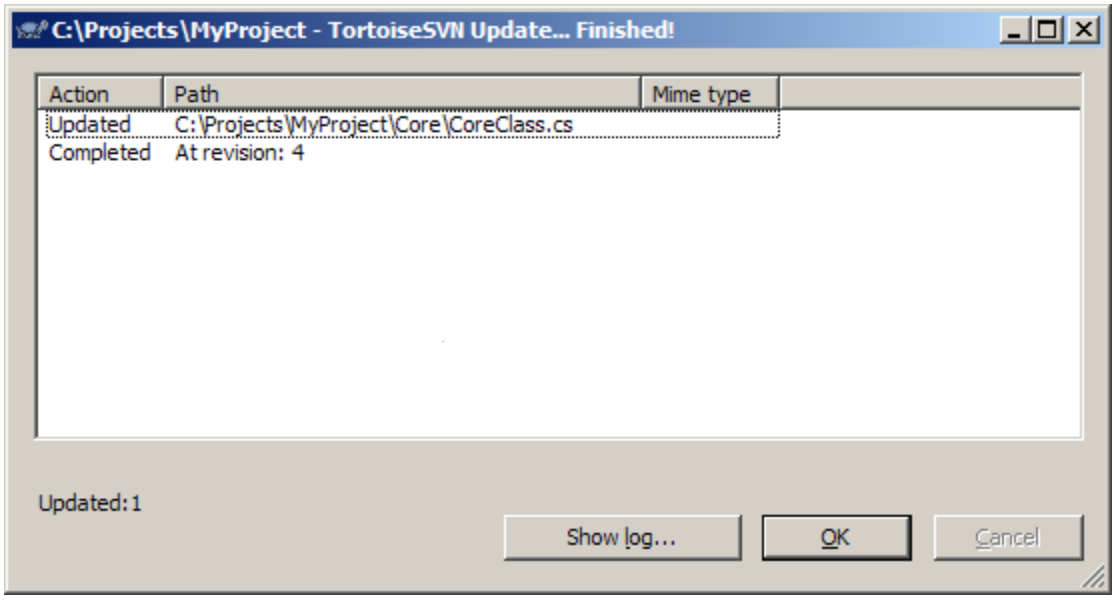
VisualSVN has numerous features, but on a day-to-day basis you will use only a few of them. In this section we'll describe the most common things that you might find yourself doing with VisualSVN in the course of a day's work.

The typical work cycle looks like this:

- Update your working copy
- Make changes
- Examine your changes
- Possibly undo some changes
- Merge others' changes
- Commit your changes

Update your working copy

When working on a project with a team, you'll want to update your working copy to receive any changes made by other developers on the project since your last update. Use the VisualSVN ⇒ Update menu item to bring your working copy in sync with the latest revision in the repository.



While updating you'll see TortoiseSVN's Update window that will show you which files of your solution are need to be updated.

Make changes

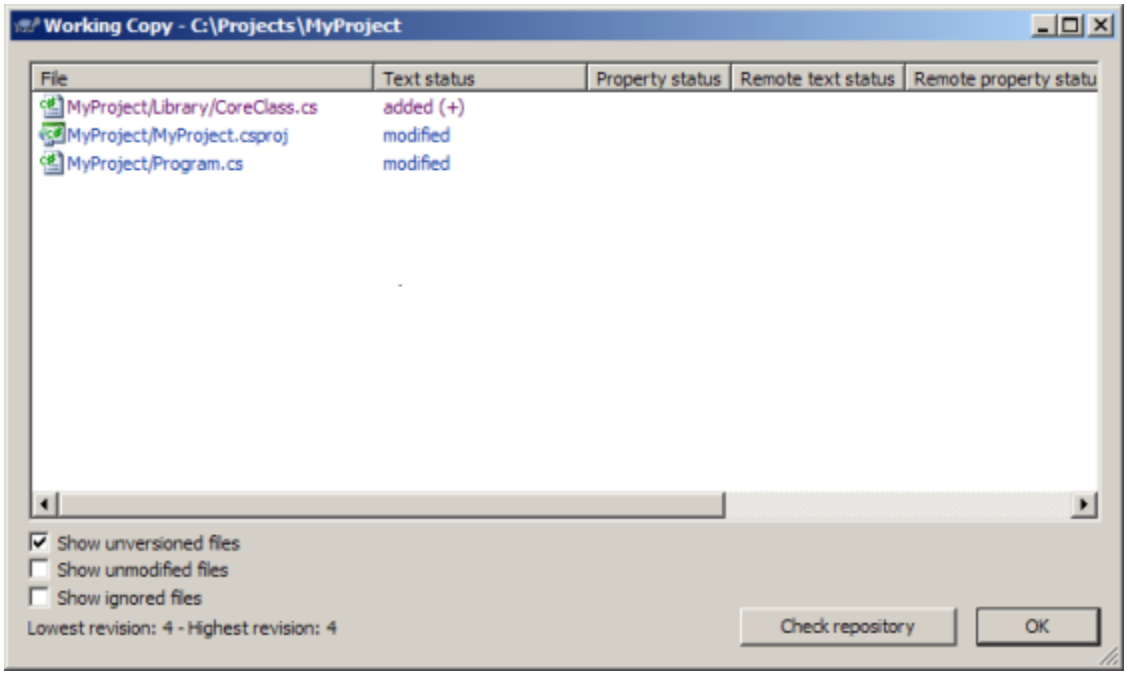
Now you can get to work and make changes in your working copy. It's usually most convenient to decide on a discrete change (or set of changes) to make, such as writing a new feature, fixing a bug, etc.

You don't need to tell VisualSVN that you intend to make a change. Just make your changes using a text editor or Solution Explorer and VisualSVN automatically detects which changes have been made.

Please note that all your changes are local to your own working copy until you commit them to the repository.

Examine Your Changes

Once you've finished making changes, you need to commit them to the repository, but before you do so, it's usually a good idea to take a look at exactly what you've changed. By examining your changes before you commit, you can write a more accurate log message. You may also discover that you've inadvertently changed a file, and this gives you a chance to revert those changes before committing them. Additionally, this is a good opportunity to review and scrutinize changes before publishing them.



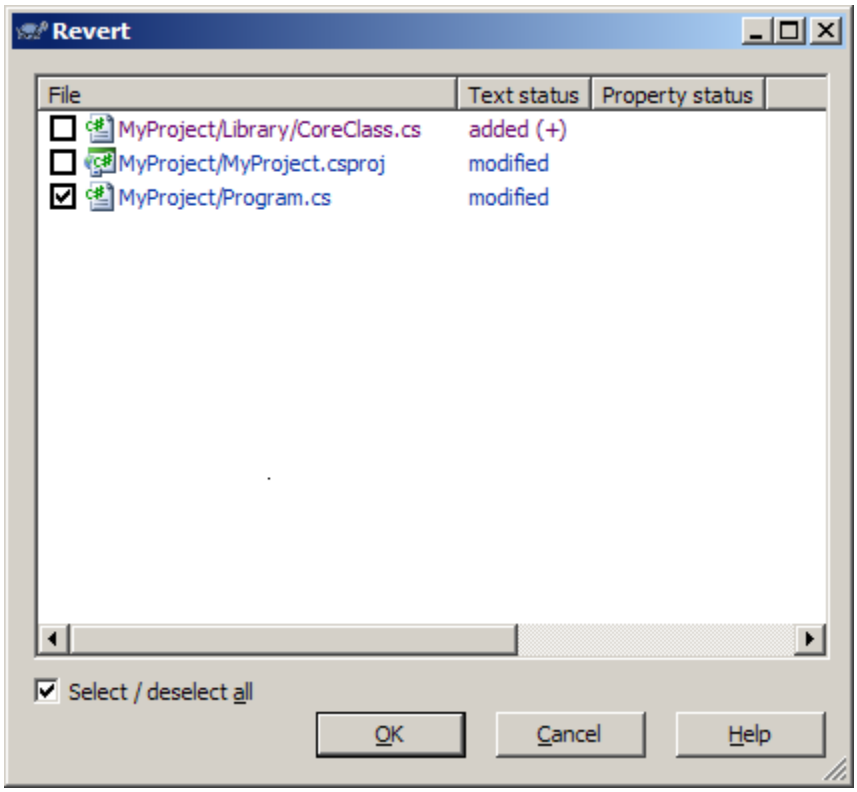
You can see an overview of the changes you've made by using TortoiseSVN's Show Changes window. To open the window choose the VisualSVN ⇒ Show Changes command on the main menu.

Please note that you can examine your changes without network access. This makes it easy to manage your changes-in-progress even when you are somewhere without a network connection.

Possibly undo some changes

Suppose while examining your changes you determine that all the changes you made to a particular file are mistakes. Maybe you shouldn't have changed the file at all, or perhaps it would be easier to make different changes starting from scratch.

This is a perfect opportunity to use the VisualSVN ⇒ Revert main menu command. You'll see TortoiseSVN's Revert window where you can examine and undo some of your changes.

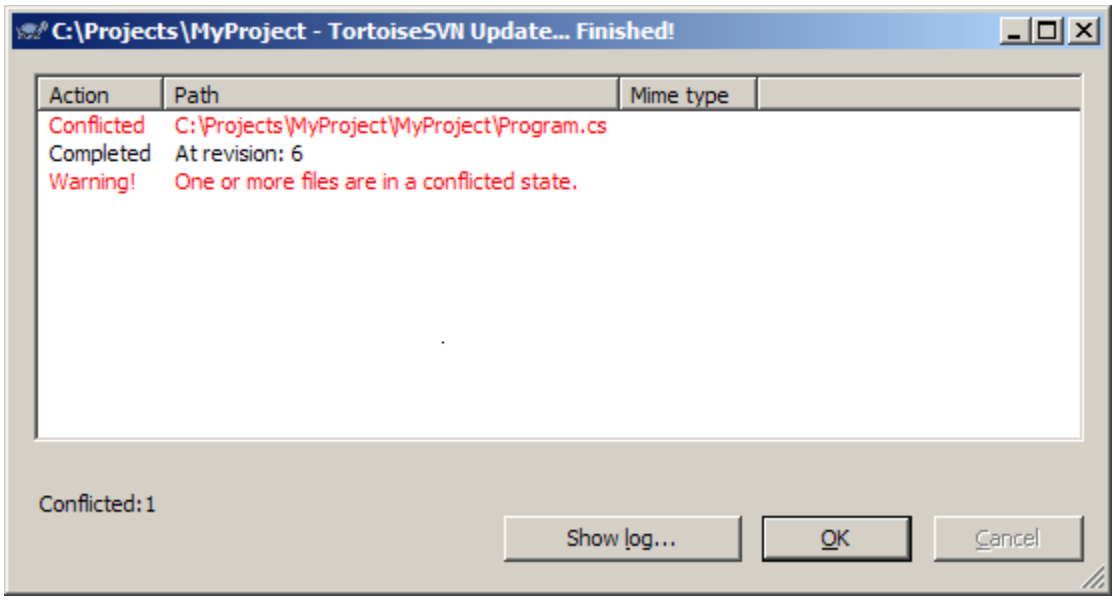


Subversion reverts the file to its pre-modified state by overwriting it with the cached “pristine” copy from the `.svn` area. You can also undo any directory modifications. For example, you might decide that you don't want to rename a file after all.

Merge others' changes

It often happens that while you've been working on your changes, somebody else already finished and committed their own changes to the repository. Thus it's a good idea to merge any changes into your working copy before commiting. Just choose VisualSVN ⇒ Update main menu command and all changes committed by others but not reflected in your working copy will be incorporated into your working copy.

Let's suppose that you and your collaborator both edited the same file at the same time. Fortunately this is not a problem in most cases and simultaneous changes will be merged by Subversion. However, sometimes Subversion can't automatically merge changes within a file and this file is marked as “conflicted”. You will be notified about conflicted files in TortoiseSVN's Update window.

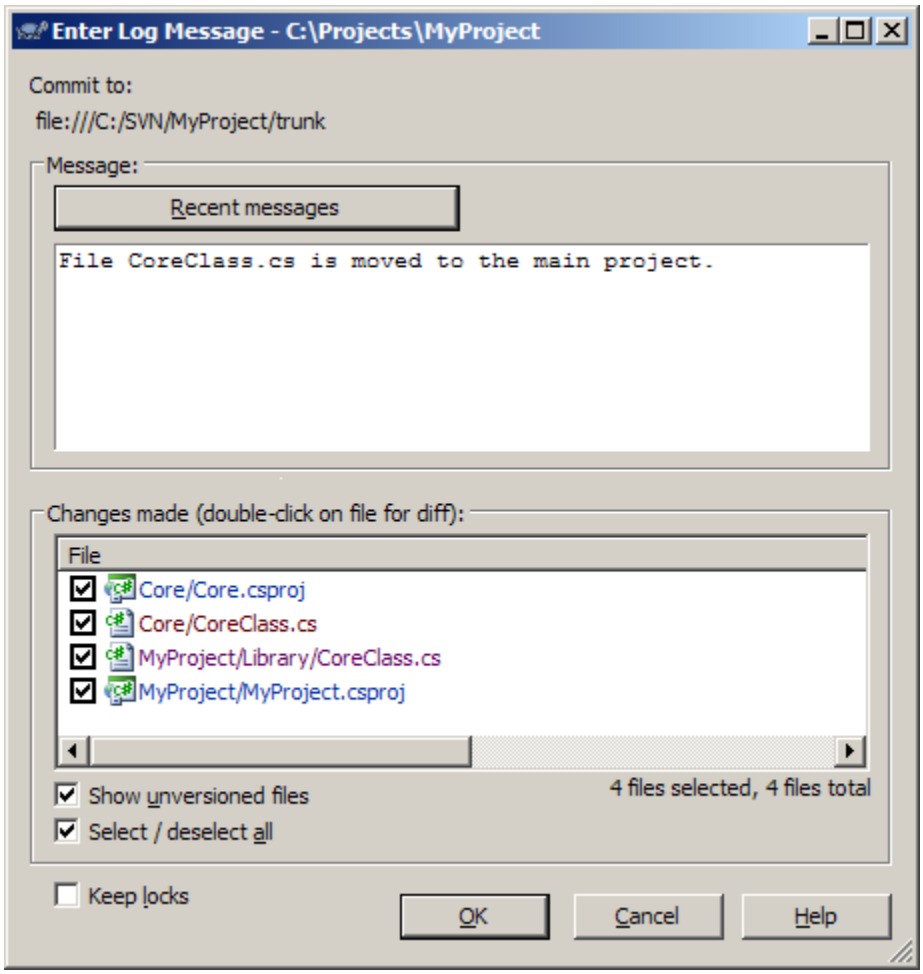


For further details about resolving conflicts please consult the chapter called *Resolve Conflicts (Merging Others' Changes)* [http://svnbook.red-bean.com/en/1.5/svn.tour.cycle.html#svn.tour.cycle.resolve] in the *Version Control with Subversion* book.

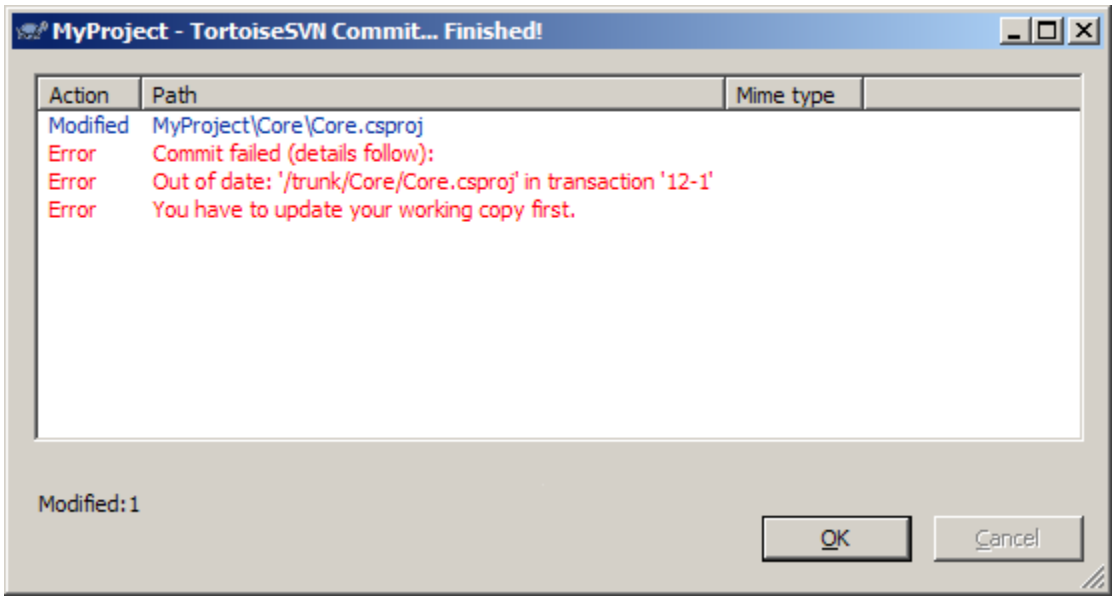
Commit your changes

Your edits are finished, you've merged all changes from the server into your working copy, and you're finally ready to commit your changes to the repository.

Please choose the VisualSVN ⇒ Commit main menu command to send all of your changes to the repository. When you commit a change, you need to supply a “log message”, describing your change. Your log message will be attached to the new revision you create.



The repository doesn't know or care if your changes make any sense as a whole; it only checks to make sure that nobody else has changed any of the files that you changed when you weren't looking. If somebody *has* done that, the entire commit will fail with a message informing you that one or more of your files are out-of-date:



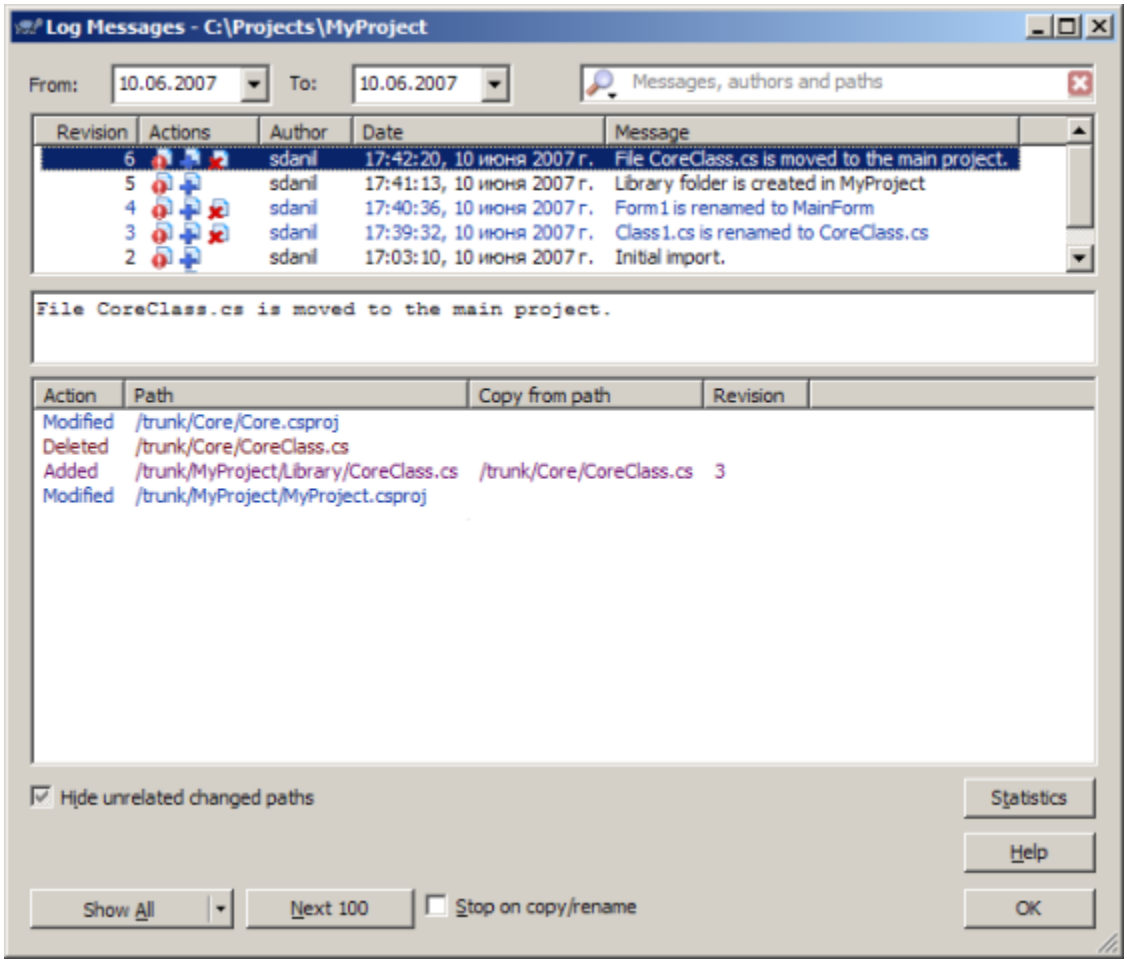
At this point, you need to update your working copy, deal with any merges or conflicts that result, and attempt your commit again.

That covers the basic work cycle for using VisualSVN. There are many other features in VisualSVN that you can use to manage your repository and working copy, but most of your day-to-day use of VisualSVN will involve only the commands that we've discussed so far in this chapter. We will, however, cover a few more commands that you'll use fairly often.

Examining History

Your Subversion repository is like a time machine. It keeps a record of every change ever committed, and allows you to explore this history by examining previous versions of files and directories as well as the metadata that accompanies them. With VisualSVN, you can check out the repository (or restore an existing working copy) exactly as it was at any date or revision number in the past.

However, sometimes you just want to *peer into* the past instead of *going into* the past. You can choose the VisualSVN ⇒ Show Log main menu command to view the history of all changes ever committed in your repository.



After launching the Show Log command from the main menu you'll see the whole history of your working copy root. You also can view the log for a particular file or folder by launching the Show Log command from the context popup menu in Solution Explorer.