Version Control in EVE: User Manual

By: Emanuele Rudel
Supervised by: Nadia Polikarpova
Prof. Dr. Bertrand Meyer

Contents

1. Subversion Client Library

The Subversion client library wraps the Subversion command line client and enables thus developers to interact with version control through Eiffel. Some of the concepts used in the library will also be illustrated in their equivalent command line client form.
The library contains a set of Subversion commands that are at the users' disposal through one single interface, namely the svn client object. The creation of this object, which will be used extensively throughout this guide, is straightforward:

```
    -- SVN client object
svn_client: SVN_CLIENT
    -- Somewhere in your class initialization
create svn_client.make
```

Working path

The SVN client usually operates on a (fixed) working copy path and applies the commands to specific targets with possibly some arguments. It is set by default to the application's absolute path and it can be changed with the following command:

```
svn_client.set_working_path ("/Users/Bob/Work")
```

It is recommended to work with absolute paths, although relative paths can also be used. This is equivalent, in a Unix environment, to the command:

```
$ cd /Users/Bob/Work
```

Note that if the path provided does not exist, the client will keep executing commands on the default working path.

Subversion commands

As already mentioned above, a command is applied to a target with possibly a number of optional values. The available Subversion commands in the library are:

1. add
2. checkout
3. commit
4. copy (renamed to cp because of the existing copy feature)
5. delete
6. list
7. log
8. merge
9. status
10. update

From now on, when we will explain a concept that holds for all of the above commands, we will simply refer to any of them as *command* or *cmd* in the code listings. We now set a target for a command:

```
svn_client.cmd.set_target (“folder”)
```

which is equivalent to

```
$ svn cmd folder
```

If we need to add optional arguments, we can use the put_option feature. The SVN client object offers the set of supported options for each command:

```
svn_client.cmd.put_option (svn_client.cmd_list.username, “bob”)
svn_client.cmd.put_option (svn_client.cmd_list.message, “A message”)
svn_client.cmd.put_option (svn_client.cmd_list.depth,
svn_client.cmd_list.depth_infinity)
```

that will result, when calling the execute feature, in the equivalent command line

```
$ svn cmd application.e --username=bob -m=’A message’ --depth=infinity
```

Depending on the type of command and the size of the working copy, its execution could require an amount of time that blocks the user interaction with the application: this is the reason why commands are executed asynchronously and offer a publisher-subscriber mechanism to notify the user that the following events occurred:

1. Data has been received: this event returns a string containing the partial result output of the command that is being executed. It may be called several times during one execution, and it is particularly useful to provide immediate feedback to the user. As an example, during a repository’s checkout the partial data output is the list of files added to the working copy could be displayed to the user as plaintext in a dialog window;
2. The command successfully completed: the user is informed that the command terminated with no errors. At this point, the output result is available as plaintext in the last_result query;
3. The command failed: the execution terminated unsuccessfully and the description of the error occurred is stored in the last_error query as plaintext.

The code in the following listing illustrates a typical usage of the callbacks:

```
 -- Set the working path, target and options first
svn_client.cmd.on_data_received (agent cmd_data_received)
svn_client.cmd.set_on_error_occurred (agent cmd_error)
svn_client.cmd.set_on_finish_command (agent cmd_finished)
svn_client.cmd.execute
```

```
 -- A simple implementation of the three callbacks
cmd_data_received (a_data: STRING_8)
          -- Partial result `a_data' has been received
     do
          print (a_data)
     end

cmd_error
          -- The command cmd terminated unsuccessfully
     do
          display_error_dialog (svn_client.cmd.last_error)
     end

cmd_finished
          -- The command cmd terminated successfully
     do
          -- Act accordingly, e.g. notify the user
          print (svn_client.cmd.last_result)
     end
```

Remember that the target, options and callbacks must always be set before issuing the execute command.

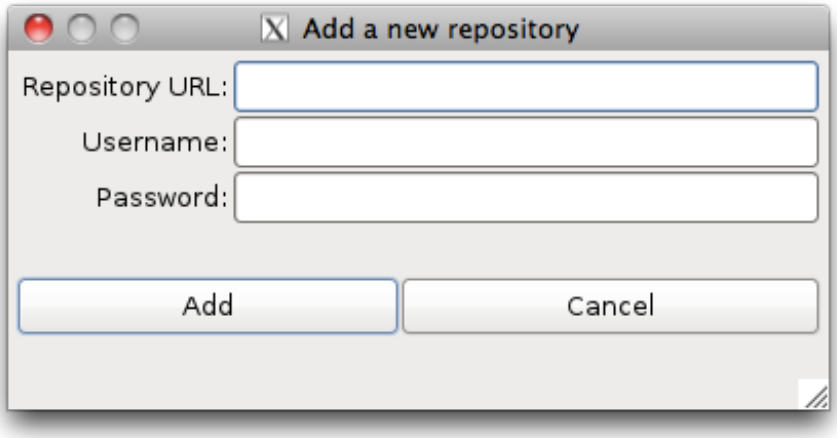
Operations like copying and merging often requires a source item, additionally to the target one. Hence, these commands let us set a source in the same way we set the target. A useful usage of the copy command is to create a new branch in the repository, which can be achieved in just four lines of code:

```
svn_client.cp.set_source ("http://svn.example.com/trunk")
svn_client.cp.set_target ("http://svn.example.com/branches/my_branch")
     -- It is assumed that the `branches' folder already exists
svn_client.cp.put_option (svn_client.cp_list.message, "Creating a new
branch")
svn_client.cp.execute
```

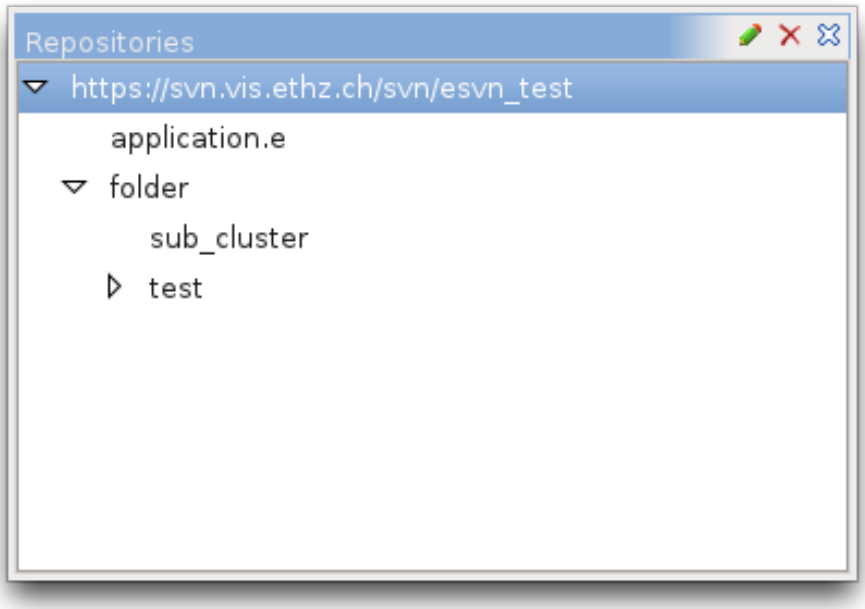
2. Repositories Tool

The Repositories tool let the user browse a repository's tree and check out a working copy of the whole repository or just a part of it. The tool can be activated from the EiffelStudio menu (select the View menu, Tools and then click the *Repositories* menu item).

There are two buttons on the top left corner to add and remove a repository from the widget. The *Add repository* button presents a dialog for typing the repository URL and username and password, if necessary.



The time required for loading the whole repository tree depends both on the size of the repository and the bandwidth available. The repository tree is eventually displayed in the Repositories tool as depicted in the next figure:



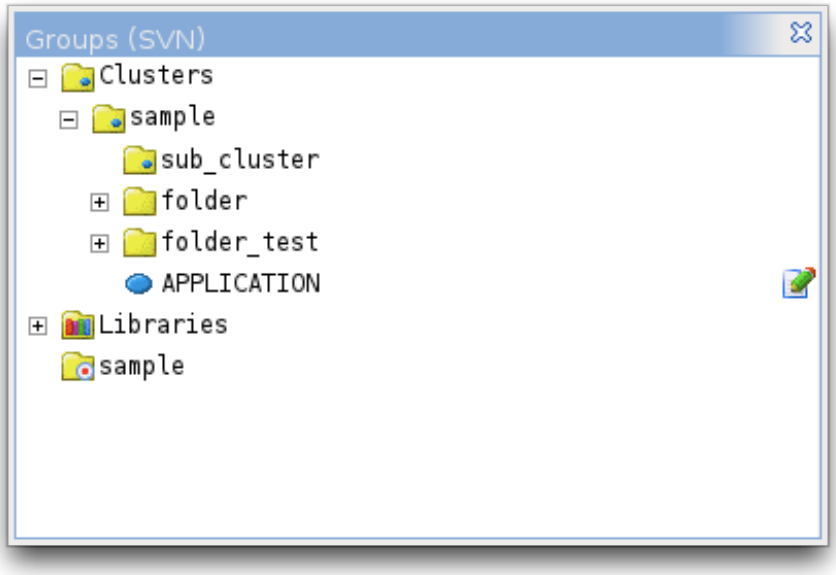
We are now able to check out a working copy of the repository (or a part of it) in three steps:

- we select and right click the item to check out; then we select the *Checkout* menu item (in the picture above, for example, we selected the root item);
- a dialog prompts us to choose a directory where the item will be checked out;
- we click the dialog's *Checkout* button and the repository is downloaded in the selected directory with the item's name (in the example above, the name would be *esvn_test*)

A repository can also be deleted from the widget. It is sufficient to select the root item of the repository and click the *Remove repository* toolbar button. If the item selected is not the root item, the repository will not be removed. This decision has been made in order to avoid any confusion with the Subversion delete command, which can actually delete the selected item (although this command is not available in the tool yet).

3. Groups Tool

The purpose of the Groups tool is to display the clusters, libraries and targets of a project. Using the Subversion client library, we are now also able to display – in the clusters – the status of the working copy, if the project is under version control. The appearance and functionalities of the widget are based on the original Groups tool's interface, hence we already are familiar with the tool. If the project opened with EVE is under version control, then an icon on the right of each cluster or class displays the status of the item in the working copy. Obviously, items that have not been modified do not display any particular icon.



In the figure above, the project named *sample* is under version control and the APPLICATION class contains some changes with respect to the repository. This representation is equivalent to the result of the following command line client execution:

```
      $ cd sample
sample$ svn status
M       application.e
```

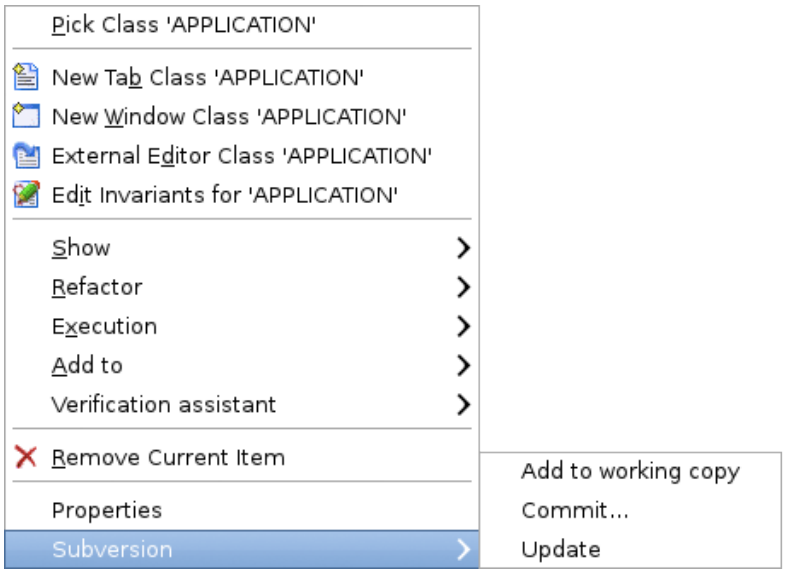
The possible statuses of an item, not including the unmodified status, are listed and associated to their icon in the table below:

Added		Conflicted		Deleted	
External		Ignored		Not under version control	
Modified		Missing		Replaced	

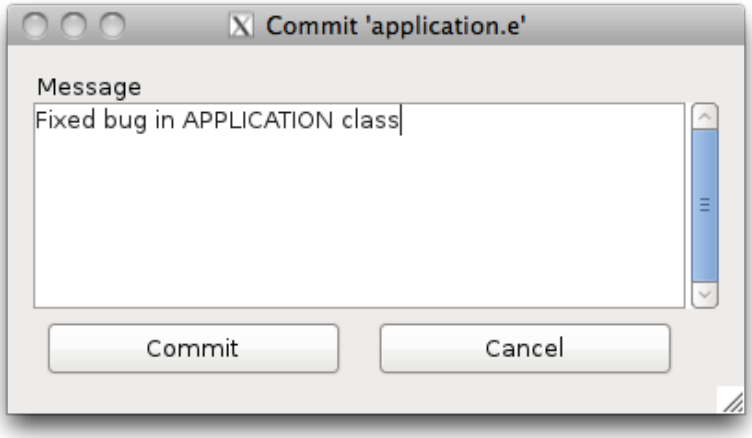
Obstructed	

These icons however, have not been designed specifically for Subversion but were already available in EiffelStudio and hence it is very likely that in the future they will be replaced with more meaningful icons.

So far we've only seen how to retrieve information and data from the repository and the working copy without the possibility of bringing changes to them. The tool let us contribute to the project under version control through a contextual menu: items can be added to the working copy or committed to the repository. Additionally, it updates the working copy to retrieve the latest changes made by other users. Right-clicking a class or a cluster displays the usual contextual menu plus a Subversion section:



The commit command always requires a log message (an empty message is also valid) and thus before the actual commit a dialog window prompts the user for a log message, as shown in the next figure:



4. SVN Output Tool

The SVN Output tool is a panel that displays the result of the executed Subversion commands in EVE in a command line interface fashion. It can be enabled from the View menu (*SVN Output* under the Tools menu item).