Transport4You1
SVN Policy

Version 1.0

Revision History

Date	Version	Description	Author
2010-09-29	1.00	First version	Dino Bartošak

Doc. No.:

1. Introduction

1.1 Purpose of this document

The purpose of this document is to provide guide for our project team so we can all use same conventions in using Subversion repository. It is very important for the team to use unique convention so the number of conflicts is reduced to a minimum. And even if conflict happens it can be resolved quickly if we stick to this guide. Each member should read this document carefully so there would be no mistakes from the beginning of coding and integration.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction,* describes contents of this guide and main purpose of this document.
- Section 2, *Subversion (SVN),* subversion conventions.

1.3 Intended Audience

The intended audience is:

- All team members of Transport4You1 project.
- Supervisor of the Transport4You1 project.

1.4 Scope

The scope of this document is adjusted to this project. In this document are only Subversion repository usage conventions that we will use in our project. It includes basic rules for using repository.

1.5 Definitions and acronyms

1.5.1 Acronyms and abbreviations

Acronym or abbreviation	Definitions
SVN	Subversion, a version control system
IDE	Integrated development environment

1.6 References

[1] Subversion wikipedia, http://en.wikipedia.org/wiki/Apache_Subversion
[2] Subversion conventions, http://geoserver.org/display/GEOSDOC/2+Subversion+conventions
[3] Subversion conventions, https://weblion.psu.edu/trac/weblion/wiki/SubversionConventions
[4] TortosieSVN standalone, http://tortoisesvn.net/downloads
[5] Eclipse SVN plug-in, http://www.eclipse.org/subversive/
[6] SVN policies and commit guidelines, http://techbase.kde.org/Policies/SVN_Commit_Policy
[7] LiveTV project SVN policy, http://fer.hr/_download/repository/SVN_Server_Policy.doc

2. Subversion (SVN)

Subversion is a revision control system founded and sponsored in 2000 by CollabNet Inc. Developers use Subversion to maintain current and historical versions of their files such as source code, web pages and documentation. We will use Subversion for our source code backups.

SVN offers many operations but the most important ones are:

1. Checkout – downloads all the files locally that are stored in the SVN repository
2. Update – downloads all the files that have been changed from the last update
3. Commit – stores the local changes in the SVN repository

2.1 Tools

To use an SVN repository we need an SVN client. That client can be standalone application or a plug-in for an IDE. In this project we will use both of them, depends of personal preference.

For a desktop application we will use TortoiseSVN [4] client that is easy to use as it only adds some functionalities to right-click mouse menu. As we will develop our project in Eclipse IDE, we will use Eclipse plug-in for subversion that can be found on [5].

2.2 Guidelines

These are very useful guidelines on how to use SVN and they can be found on [6] .

2.2.1 Think Twice before Committing

Committing something to SVN has serious consequences. All other developers will get your changes once they are in SVN, and if they break something, they will break it for everybody. All commits will be publicly available in the SVN repository forever.

On the other hand SVN allows one to revert changes, so it's possible to recover from mistakes. This is relatively easy for commits to single files but it can also be a significant amount of work for bigger changes. The baseline is: Be aware of the consequences of your commits. Take time to think about them before committing.

2.2.2 Never commit code that doesn't compile

Compile the code and correct all errors before committing. Make sure that newly added files are committed. If they are missing your local compile will work fine but everybody else won't be able to compile.

You certainly should make sure that the code compiles with your local setup. You should also consider what consequences your commit will have for compiling with the source directory being different from the build directory.

Be especially careful if you change the build system, i.e. Makefiles, ANT script, MVN etc.

2.2.3 Test your changes before committing

Start the application affected by your change and make sure that the changed code behaves as desired. Run unit test.

2.2.4 Double check what you commit

Do a SVN update and a SVN diff before committing. Take messages from SVN about conflicts, unknown files, etc. seriously. SVN diff will tell you exactly what you will be committing. Check if that's really what you intended to commit.

2.2.5 Always add descriptive log messages

Log messages should be understandable to someone who sees only the log. They shouldn't depend on information outside the context of the commit. Try to put the log messages only to those files which are really affected by the change described in the log message.

In particular put all important information which can't be seen from the diff in the log message.

2.2.6 Honor KDE coding policies

This includes security (shell quoting, buffer overflows, format string vulnerabilities), binary compatibility (d pointers), i18n, usability, user interface style guide, (API) documentation, documentation and definition of memory management and ownership policies, method naming, conditions for inclusion in kdelibs, portability issues and license notices.

These policies are defined separately in coding policy document.

2.2.7 Respect other developer's code

Respect the policies of application and library maintainers, and consult with them before making large changes. Source control systems are not a substitute for developer communication.

2.2.8 Announce changes in advance

When you plan to make changes which affect a lot of different code in SVN, announce them on the mailing list in advance. For instance, changes in libraries might break other code even if they look trivial, e.g., because an application must also compile with older versions of the library for some reasons. By announcing the changes in advance, developers are prepared, and can express concerns before something gets broken.

2.2.9 Code review by other developers

Don't commit changes to the public API of libraries without prior review by other developers. There are certain special requirements for the public APIs of the KDE libraries, e.g., source and binary compatibility issues. Changes to the public APIs affect many other KDE developers including third party developers, so requiring a review for these changes is intended to avoid problems for the users of the APIs and to improve the quality of the APIs.

2.2.10 Take responsibility for your commits

If your commit breaks something or has side effects on other code, take the responsibility to fix or help fix the problems.

2.2.11 Don't commit code you don't understand

Avoid things like "I don't know why it crashes, but when I do this, it does not crash anymore." or "I'm not completely sure if that's right, but at least it works for me.".

If you don't find a solution to a problem, discuss it with other developers.

2.2.12 Don't commit if other developers disagree

If there are disagreements over code changes, these should be resolved by discussing them on the mailing lists or in private, not by forcing code on others by simply committing the changes to SVN.

2.2.13 Backport bugfixes

If you commit bugfixes, consider porting the fixes to other branches. Use the same comment for both the original fix and the backport, that way it is easy to see which fixes have been backported already.

2.2.14 Use bug tracking system numbers

If you fix bugs reported on the bug tracking system, add the bug number to the log message. In order to keep the bug tracking system in sync with SVN, you should reference the bug report in your commits, and close the fixed bugs in the bug tracking system.

This doesn't mean that you don't need an understandable log message. It should be clear from the log message what has been changed without looking at the bug report.

2.2.15 Don't use SVN keywords in source files

SVN keywords like `Id` or `Log` cause unnecessary conflicts when merging branches and don't contain any information which wouldn't be available in the SVN repository anyway.

2.2.16 Commit complete changesets

SVN has the ability to commit more than one file at a time. Therefore, please commit all related changes in multiple files, even if they span over multiple directories at the same time in the same commit. This way, you ensure that SVN stays in a compileable state before and after the commit, that the commit history (svn log) is more helpful and that the kde-commits mailing list is not flooded with useless mails.

2.2.17 Don't mix formatting changes with code changes

Changing formatting like indenting or white spaces blows up the diff, so that it is hard to find code changes if they are mixed with re-indenting commits or similar things when looking at the logs and diffs later. Committing formatting changes separately solves this problem.

2.2.18 GUI Changes

If your commit causes user visible GUI changes, add the `GUI` keyword to the log message. This makes sure that the documentation writers get notified of your changes.

2.3 Transport4You Committing Rules And Conventions

1. All members have commit privileges.
2. The same SVN repository will be used for multiple projects, so every project should be placed in its own folder on the SVN.
3. Always write a short comment of the changes you are committing.
4. Every version committed to the SVN must compile and pass all tests.
5. Project's binary files shouldn't be committed, unless they are a part of some library that the project uses.
6. When resolving conflicts, be careful not to override other people's changes.
7. Commit all related changes in multiple files, even if they span over multiple directories at the same time in the same commit. This way, you ensure that SVN stays in a compileable state before and after the commit.
8. SVN ignore on local dependent files.

Files that should **NOT** be committed:

General files and folders:

- thumbs.db

Eclipse:

- All files from the **bin** folder (*.class, …)

NetBeans:

- All files from **build** folder
- All files from **dist** folder
- All files from **nbproject/private** folder

With TortoiseSVN above files and folders can easily be excluded with a global ignore pattern. This pattern can be set by right clicking on any folder and choosing TortoiseSVN -> Settings command. Global ignore pattern can be defined in the dialog that opens under option General.

An example of a global ignore pattern that could be used to ensure that above files and folders don't get committed:

```
thumbs.db build dist private bin
```