SVN CONVENTIONS

Note: This document is based on the SVN Red Book which has been written by Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato and released under the Creative Commons Attribution licence. The book is available at http://svnbook.red-bean.com/en/1.7/.

What is SVN?

Subversion is a free/open source version control system (VCS). That is, Subversion manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data or examine the history of how your data changed. In this regard, many people think of a version control system as a sort of ‘time machine.

Progress can occur more quickly without a single conduit through which all modifications must occur. And because the work is versioned, you need not fear that quality is the trade-off for losing that conduit—if some incorrect change is made to the data, just undo that change.

Structure of the SVN repository

Subversion provides the ultimate flexibility in terms of how you arrange your data. Because it simply versions directories and files, and because it ascribes no particular meaning to any of those objects, you may arrange the data in your repository in any way that you choose. Unfortunately, this flexibility also means that it's easy to find yourself “lost without a roadmap” as you attempt to navigate different Subversion repositories which may carry completely different and unpredictable arrangements of the data within them.

To counteract this confusion, for the duration of this project we will follow a repository layout convention in which a handful of strategically named Subversion repository directories convey valuable meaning about the data they hold. These are the recognizable “main line”, or *trunk*, of development; some *branches*, which are divergent copies of development lines; and some *tags*, which are named, stable snapshots of a particular line of development. Both the *tags* and *trunk* folders will have subdirectories called *SmartCart_MobileApplication* and *SmartCart_ServerApplication* which will store the files of the respective projects.

Trunk

There are several guidelines which we should keep in mind during the development of the SmartCart project in regards to the *trunk* folder.

- the *trunk* should always contain a stable build and should you discover that the code in the repository does not build, you should revert it to the last stable state or fix the problem as soon as possible.
- if you are editing a binary file (any file which is not text based including doc and docx), acquire an svn lock first using the `svn lock` option/command. This will ensure that no other person will be able to commit in the same time as you are editing which in turn will prevent you overwriting his changes when you commit. Remember to unlock the file when you are done.
- if you need to edit a file which is locked, find out who holds the lock using the `svn status` and `svn info` options/commands. Even though, it is possible to steal a lock from another person, refrain from doing it.
- commit your changes as often as can and do not wait until you are done editing a file or writing a new one. The more often you commit, the easier to rollback the changes or merge in case of a conflict. Do keep in mind however, that you should not commit code which you know will breat the build of the *trunk*.

Branches

The branch is a copy of the main development line which exist independently from the trunk, yet still shares a common history if you look far enough back in time. A branch always begins life as a copy of something, and moves on from there, generating its own history. Branches allow you to work on a piece of code which is unstable/contains experimental features which may or may not be used in the main project. This allows you to work undisturbed by the changes to the main trunk and integrate with trunk when you feel the branch is mature enough.

Subversion has commands to help you maintain parallel branches of your files and directories. It allows you to create branches by copying your data, and remembers that the copies are related to one another. It also helps you duplicate changes from one branch to another. Finally, it can make portions of your working copy reflect different branches so that you can “mix and match” different lines of development in your daily work. Please have a look at the chapter 4 in the SVN red book for the detailed steps on creating a new branch.

Tags

Another common version control concept is a tag. A tag is just a “snapshot” of a project in time. In Subversion, this idea already seems to be everywhere. Each repository revision is exactly that—a snapshot of the filesystem after each commit.

However, people often want to give more human-friendly names to tags, such as release-1.0. And they want to make snapshots of smaller subdirectories of the filesystem. After all, it's not so easy to remember that release 1.0 of a piece of software is a particular subdirectory of revision 4822.

No file should be committed to the *tags* folder manually. A tag is created by copying the state of the trunk. In order to avoid any confusion, version tagging should be performed only by the SVN coordinator

Documentation

The documentation folder stores the formal documents and presentation create for this project. The internal folder structure should mimic the folder structure of the DSD site.

Basic work cycle

Subversion has numerous features, options, bells, and whistles, but on a day-to-day basis, odds are that you will use only a few of them. In this section, we'll run through the most common things that you might find yourself doing with Subversion in the course of a day's work.

The typical work cycle looks like this:

1. Update your working copy. This involves the use of the svn update command.
2. Make your changes. The most common changes that you'll make are edits to the contents of your existing files. But sometimes you need to add, remove, copy and move files and directories—the svn add, svn delete, svn copy, and svn move commands handle those sorts of structural changes within the working copy.
3. Review your changes. The svn status and svn diff commands are critical to reviewing the changes you've made in your working copy.
4. Fix your mistakes. Nobody's perfect, so as you review your changes, you may spot something that's not quite right. Sometimes the easiest way to fix a mistake is start all over again from scratch. The svn revert command restores a file or directory to its unmodified state.
5. Resolve any conflicts (merge others' changes). In the time it takes you to make and review your changes, others might have made and published changes, too. You'll want to integrate their changes into your working copy to avoid the potential out-of-dateness scenarios when you attempt to publish your own. Again, the svn update command is the way to do this. If this results in local conflicts, you'll need to resolve those using the svn resolve command.
6. Publish (commit) your changes. The svn commit command transmits your changes to the repository where, if they are accepted, they create the newest versions of all the things you modified. Now others can see your work, too!

Backup

Full repository backup is performed daily and can be accessed at code.google.com/p/smartcart. In order to prevent the desynchronization of the backup SVN server, no one (including the SVN coordinator) is allowed to directly commit changes to the backup SVN server.