Primonics SVN Migration Process

**Current Structure**

The current code versioning structure used by Primonics has worked in the past due to the different corporate structure, low number of developers as well as low number of concurrent projects. Given the recent changes to all of the above points, a new more organized structure is required. This will help make the flow of code run more smoothly from development, to internal QA, to external QA and finally to a production environment.

While we currently do make use of the tree metaphor implemented by SVN software, our use of it is very light. Some of the powerful features that SVN offers are going completely unused (proper branching and merging) and this is creating difficulties in the development process when more than one version of a particular project is needed. Furthermore, the organizational structure of the folders is in disarray. Some folders are stored in the root directory, others are stored under subdirectories, and others are deprecated versions of projects that are not under development anymore.

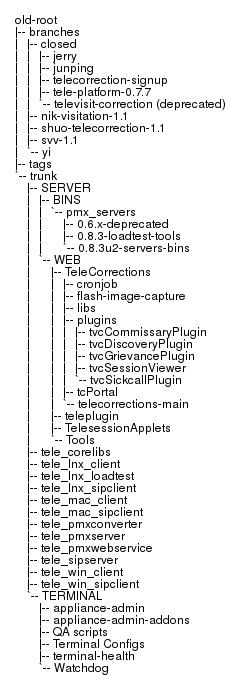
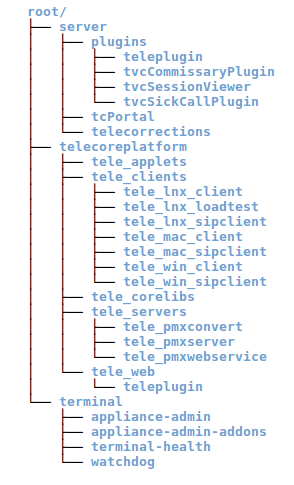
****The SVN structure (as of September 4th, 2012) is as follows (individual files have been omitted):

Figure 1: Current SVN Structure

**New Structure**

The following is an excerpt from the current SVN book[[1]](#footnote-1):

****“[…] *So we first recommend that each project have a recognizable project root in the repository, a directory under which all of the versioned information for that project—and only that project—lives. Secondly, we suggest that each project root contain a trunk subdirectory for the main development line, a branches subdirectory in which specific branches (or collections of branches) will be created, and a tags subdirectory in which specific tags (or collections of tags) will be created.*”

In order to counter the problems mentioned above and improve our overall development, testing, and release processes, I would like to propose an SVN structure like the one illustrated on the right, with each child-less folder having a trunk, branches and tags folder. It works directly off of the above quote, with a few modifications for our specific needs.

Since we have (for the most part) multiple projects for each aspect of TeleCorrections (telecoreplatform, server, terminal), the main root will be divided as such. Under each of those main directories we might find more sub-directories to further organize that specific aspect of TeleCorrections.

Inside each of those sub-directories we will find the structure proposed by the SVN book. Each of those sub-directories (trunk, branches, and tags) will be used in a very specific way:

Figure 2: New SVN Structure

* **Trunk**
  + **Description:** The *trunk* folder will house all code and project directories related to it. The state of the trunk is always working (i.e. it always compiles and/or runs) and its stability grows over time until it is stable enough to make it into its own tag. When a developer wants to work on that project, he will make a branch of the HEAD revision into the branches folder.
  + **Special Characteristics:** Absolutely no one is ever allowed to work (or commit) directly with the trunk folder. The only commit allowed is when a user is merging a change from his own branch (tracked internally with the svn:mergeinfo property and the revision logs).
* **Branches**
  + **Description:** The *branches* folder will contain individual users’ folders which are usually branched off of the latest HEAD revision from the trunk.
  + **Special Characteristics:** These branches will be relatively short-lived, i.e. a branch is created from the trunk, work is done to it and once that work is done the branch is re-integrated to the trunk and destroyed. In order to start new work on the trunk a new branch is to be created rather than continuing work on that same branch (such is the nature of SVN’s re-integrate[[2]](#footnote-2) operation when merging back into the trunk).
* **Tags**
  + **Description:** The *tags* folder is to be used as a copy-only location. It will store versions of projects deemed ready for release to QA and/or production. The naming convention of the folder will depend on the project that is being tagged; however, one characteristic will be common to every single tag created: The revision number must be appended to the name of the folder in the format *rREVISION\_NUMBER,* where REVISION\_NUMBER is the revision number of the folder being copied to the tags folder.
  + **Special Characteristics:** All tags are created using SVN’s branch feature since this will prevent the repository from growing too large. Similar to the trunk folder, no one is ever allowed to work (or commit) to the tags folder. If a bug fix is needed for a specific tag, then a branch is created from that tag to the branches folder, the bug is fixed, and a new tag is created with the same name and a new revision number.

**Moving to the new structure**

The conversion process from our current structure to the new one should proceed in the following manner:

1. The server software would be upgraded to the latest version of Subversion.
2. Old versions of code would be moved into the tags folder that corresponds to that project. Given that they are old versions of code, this will be the only time where the revision tag can be omitted.

**Note:** The moving of folders themselves requires committing these changes. As such, the revision number before and after the restructuring should be noted in order to prevent anyone from making checkouts in that range of revision numbers.

1. Projects where the trunks are in active development would be verified to have their SVN code in a working state and then frozen (whether a feature is currently being developed or not). This code would become the trunk’s HEAD revision for that project. Further work on that trunk is then performed on a branch instead of the trunk itself.
2. Once the trunk is established SVN properties must be assigned to the project (if applicable). Example: The tele\_lnx\_client project (along with others) depends on tele\_corelibs. The svn:externals[[3]](#footnote-3) property would be added to account for this dependency in order to facilitate future commits and checkouts.
3. At this point the SVN repository is ready to be worked on again following the new procedures.

**Working with Branches[[4]](#footnote-4)**

**Creating a Branch**

* Check out contents of trunk into one folder (ex: projectName-trunk). This folder is **not modified or worked on directly**.
* Right click the folder, TortoiseSVN > Branch/tag (**Linux:** svn copy TRUNK\_URL BRANCHES\_URL/branchName -m "Creating a private branch of the trunk to add X feature." **Note:** At this point the Linux user is done and must check out the new branch to work on it.)
  + **To path:** Change it so it uses the branches folder in the repository along with a branch name (use the "..." button or type the path directly)

Example: /projectName/branches/projectName-cristian

* + **Log message:** Type in a significant log message to explain the reason to create the branch.
  + **Create copy in the repository from:** Make sure the HEAD radio button is chosen in order to obtain the latest version of the trunk in the branch.
  + **Set explicit revision... :** Leave as is
  + **Switch working copy to new branch/tag:** Leave unchecked
* After clicking OK you might get a warning message saying your working copy is still on the previous path. This is fine; you can just close the message.
* Do a second check out, this time selecting the branch folder that was just created for you.
* Start your work on the branch folder that was just checked out, working as usual and doing regular commits.

**Merging trunk changes into own branch (minimum once per day)**

* Run an SVN update on the original trunk folder that was checked out (in this case it's projectName-trunk). There should not be conflicts to resolve since you're only receiving updates.
* Commit any changes you've made to your own branch.
* Right click on your branch folder, TortoiseSVN > Merge (**Linux:** svn merge TRUNK\_URL **Note:** At this point the Linux user must resolve conflicts (if any) and then commit the changes that were made to his local branch.)
* Choose the first radio button (Merge a range of revisions) and click Next
* In the Merge Revision Range window:
  + **URL to merge from:** It's the folder where we're receiving updates from. In this case it is the trunk folder, so make sure that it's selected.
  + **Revision range to merge:** Leave blank
  + **Working copy:** Make sure that the path being displayed is **your branch’s** folder
  + Click Next
* In the Merge Options window leave the settings as is. If you would like to see what the merge will be like first, you may click on the Test Merge button. Otherwise, click on Merge.
* If you have conflicts:
  + You should get a prompt telling you about the conflict. You can choose between keeping your file (possibly overwriting someone else's work), keeping the repository's file (possibly overwriting your own work), or editing the conflict (so you can keep both contents). Edit conflict is the most used option, click it.
  + In the TortoiseMerge window, right click on the first highlighted line and choose the most appropriate option. Usually we'll pick "use text block from 'mine' before 'theirs'" or "use text block from 'theirs' before 'mine'".
  + Repeat the previous steps for all conflicts, save, and then close the TortoiseMerge window.
  + Back in the conflicts prompt, you can now click the Resolved button.
  + If there are more conflicts, then you'll have to do the same process for those.
* If there aren't any conflicts (or you resolved them), then you will see a success message and the revision number range that was merged into your branch. **Note:** The merging occurred in your **LOCAL** branch's folder. You must now do an SVN Commit on your branch's folder in order to save those changes into the actual SVN repository. It is possible that the actual SVN information related to the branch will also change after committing and you'll have to do a second commit after doing the first one.

**Re-integrating own branch into the trunk**

* SVN update the original trunk that was checked out so there won’t be any conflicts when merging
* Right click the trunk folder, TortoiseSVN > Merge (Linux: svn merge –reintegrate YOUR\_BRANCH\_URL

**Note:** At this point the Linux user must resolve conflicts (if any) and then commit the changes that were reintegrated to his local trunk to the trunk in the repository.)

* Choose the second radio button (Reintegrate a branch)
* In the Reintegrate Merge window:
  + **From URL:** Make sure the URL of your branch is selected
  + **Working copy:** Make sure the file path of the trunk is displayed
* Click Next
* In the Merge Options window leave the settings as is. If you would like to see what the merge will be like first, you may click on the Test Merge button. Otherwise, click on Merge.
* Once the merge completes successfully you will have your branch merged with your **LOCAL** trunk’s folder. Run an SVN Update first (to make sure no changes are missed) and then run SVN Commit in order to save the merged changes into the SVN repo.

1. <http://svnbook.red-bean.com/en/1.7/svn.tour.importing.html#svn.tour.importing.layout> [↑](#footnote-ref-1)
2. <http://svnbook.red-bean.com/en/1.7/svn.branchmerge.basicmerging.html#svn.branchemerge.basicmerging.reintegrate> [↑](#footnote-ref-2)
3. <http://svnbook.red-bean.com/en/1.0/ch07s03.html> [↑](#footnote-ref-3)
4. Different method to branch/merge is described here: <http://stackoverflow.com/questions/1461922/what-is-the-simplest-way-to-do-branching-and-merging-using-tortoisesvn> [↑](#footnote-ref-4)