Committing Code

When you think you are ready to commit code to SVN

This is the process that all QuikView / LiveCycle developers must use for committing code. Not following these steps will most likely result in a broken build and prevent other developers from getting the latest code and committing their changes.

This document refers to the current code, current branch that you're working in.

1. **Update**: Begin by updating your workspace with **all** of the latest code from the repository. A partial update is worthless.
    a. Execute svn update at the root project level, using your tool of choice. Example:
    `C:\quikview\2.4>svn up`
    b. If you are using Subclipse, be sure to select **all** of the projects when you synchronize (including the `Parent` project)
2. **Merge**: Merge changes from the repository into your workspace:
    a. Do all merges manually. Regardless of what merge tool you are using, do not trust it to know what changes to accept.
    b. If you do not understand the code you are merging or see something of concern, then contact the person who committed the changes to the repository and ask for their help in completing the merge.
    c. Merging up and merging down between branches and trunk is a different process. See Source Code Merging elsewhere in this qwiki for information about that type of code merge.
3. **Maven Build**: Build your updated workspace locally using the Maven build. Code that builds with Eclipse may not build with Maven, and the Maven build is the one that counts.
    a. Run `mvn clean package` from the `Parent` directory. Example:
    `C:\quikview\2.4\Parent>mvn clean package`
    b. The build is only successful if you see
    `[INFO] BUILD SUCCESSFUL`
    at the bottom of the Maven output.
    c. If you see anything else, the build failed – do NOT commit your changes until you have fixed the problem and completed a successful Maven build. Sometimes the FlexUnit TestRunner.swf will timeout if your computer is under heavy load. This does NOT constitute a successful build – do NOT commit your changes until you get a successful build.
    d. Do NOT make any changes to the code after you get the successful build. If you do have to make changes, you will need to re-run the Maven build until you get a successful build.
4. **Test**: If the code builds, install/run the AIR file and make sure it is functioning as expected. **(Not only should you not break the build, you should not break the executable.)**
5. **Repeat**: If you spent any amount of time doing the merge locally, you need to update your code from the repository again and make sure that your merge has not already become stale. Repeat steps 1-4.
6. **Commit**: Commit your updated files *with a verbose description of the changes* – something that other developers will understand and allow them to **differentiate this change from other changes in the repository**, including the related JIRA ID (for example, MMXMARCH-319). See Jira and Fisheye Integration for more details.
    a. The JIRA ID is **mandatory** in your checkin comments.
    b. Be sure to commit all files that you changed IN A SINGLE COMMIT. Using Subclipse is the easiest way to accomplish this, but make sure you *Refresh* all of the projects in Eclipse before going to the *Team Synchronization* view. If you use multiple commits, you are likely to break the build because TeamCity may kick off the build before you get all the commits done. Another way to think about it is, if you use multiple commits, other developers updating from SVN at the same time may get some of your changes, but not all, and therefore their code will be in an inconsistent state.
    c. Do not forget to add any new files to the repository that your changed files may be dependent upon. Execute svn status just to

make sure.

7. **Monitor**: Watch TeamCity to make sure the build with your changes succeeds.

When you break the build (or the executable)

1. Immediately notify `Corporate UI Development - All Teams` by replying to the TeamCity e-mail notification – indicate that you are working on fixing the break. A broken build affects all developers, and we have a lot of developers. When the build is broken, it prevents other developers from getting the latest code and from committing their changes. Don't break the build.

2. Fix the build. Get help if needed.

3. When you have confirmed that the build is fixed AND it succeeds in TeamCity, reply to your initial e-mail stating:
    a. What caused the build to break
    b. What you did to fix it

We use a pre-commit hook

Our subversion source repositories require developers to supply a JIRA ID for every checkin. Before any file can be added or updated in the repository, a script called a "pre-commit hook" runs that parses the check in comments of the files being checked in. If it finds a valid JIRA ID, the code is checked in. A valid ID looks like this "QV-65".If the pre-commit hook finds no JIRA ID, or an ID that doesn't exist in our JIRA database, the commit is rejected.

That doesn't sound very friendly. It actually is quite friendly.

We need that ID

There are two very good reasons for requiring that ID, and several other just good reasons.

- We need that JIRA ID to do code merging. Our code merging is based on fixed bugs and finished stories in JIRA. When we are cherry picking files for the merge, rather than just doing a wholesale merge, these JIRA IDs are critical. The best way to determine what files are safe to merge is to compare them against JIRA status.
- We need that JIRA ID for project status tracking with Greenhopper. These make the reports possible.
- Developers can determine who fixed what, using the JIRA/Greenhopper tools.

Please put JIRA IDs in your commit comments.