**Repository Layout and Version Numbering for Migrated Local Applications**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#RepositoryLayoutandVersionNumberingforMigratedLocalApplications)

This page provides:

* a description of the layout of the Subversion repository for local applications and smart tools;
* a policy for assigining version numbers to local applications freshly ported to AWIPS II;

To get started with Subversion, see [HowToSubversion](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToSubversion), and the [Working with Subversion](https://collaborate.nws.noaa.gov/trac/asdt/wiki/AiiHbDevEnvUseSu) section of the *AWIPS II Developers' Handbook.* (Note that the quality control processes mentioned in the Handbook apply to development of AWIPS II core software. Processes for porting and maintaining local applications may differ.)

**Repository Layout**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#RepositoryLayout)

Anything that was or would be suitable for putting on the LAD and STR shall be placed in the toplevel **ladroot** directory of the the NCLADT repository (Subversion URL: <https://collaborate.nws.noaa.gov/svn/ncladt>). This repository also has toplevel trunk, tags and branches directories. In general, applications, utilities, files, etc. that are used for the migration process will be in these directories. The reasoning here is that once the migration process is complete, these items will no longer be relevant. Non-migration specific applications that were put in the toplevel trunk before the ladroot path was established, should be renamed (using the svn rename command) and reorganized to be in ladroot.

Top-level directories beneath ladroot:

* **apps**. Contains one subdirectory for each local application.
* **lib**. Contains per-language subdirectories (perl, tcl, etc.) Those subdirectories contain one subdirectory for each local application library.
  + **perl**
  + **python**
  + **tcl**
* **gfe**
  + **procedures**
  + **utilities**
  + **formatters**
  + **smartinits**
  + **smarttools**
  + **apps**. Any GFE-related code that covers more than one of the areas above.
* **plugins**. For applications installed as Cave or edex plugins.

GFE files should be named "name.class" where name and class are the values used with the -n and -c arguments to ifpServerText. SmartInits should use a .py extension. GFE files should be installed using the ifpServerText program and not written directly to the edex file system. The valid ifpServerText classes are:

* EditAreaGroup
* EditArea
* WeatherElementGroup
* ColorTable
* SelectTR
* SampleSet
* Tool
* Procedure
* Utility
* TextUtility
* TextProduct
* Config
* Combinations

The local applications will all be directly under apps with no subgroups.

The three traditional Subversion directories: branches, tags, and trunk, will appear under the directories for individual applications, libraries, etc.

**trunk[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout" \l "trunk" \o "Link to this section)**

The application files (source code, configuration files, etc.) will be in the trunk directory. The structure of the trunk directories will vary from application to application, however, the application's directory structure should follow the recommendations of the [Local Applications Guide](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/LocalApplicationsGuide#DirectoryStructure) (i.e., contain bin, config, data, doc, etc, lib, log subdirectories as needed). Code in the trunk is the latest and greatest version, but it might not be ready for release.

**branches[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout" \l "branches" \o "Link to this section)**

The branches directory under each application may contain:

* Temporary directories used by individual developers if the developer needs a safe place to park or share work-in-progress but feels that it is not ready for the trunk. That kind of "development branch" should be named with the developer's login name (for example, george.washington).
* More long-lasting branches for separate lines of development. If, for example, it becomes necessary to maintain an AWIPS I version of an application in parallel with an AWIPS II version that could be done in a separate branch named "awips1."

Branches should be created initially by making a copy from the trunk.

Developers might work by checking code in and out of the trunk, or they might make their own temporary development branches. The tradeoffs involved with that decision are summarized in the [Best Practices](http://svn.apache.org/repos/asf/subversion/trunk/doc/user/svn-best-practices.html) document on the Subversion web site.

**tags[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout" \l "tags" \o "Link to this section)**

Tags are snapshots of trunk. They never change. Typically a tag is created when code is released, but one may be created whenever it would be convenient to have a named pointer to a certain revision. Tags are created by using the svn copy command to copy trunk to tags/<version>.

The tags directory will be used to "tag" a specific revision of an application with a version number. Development is always done in trunk (or branches), but users will obtain the application for use from tags. Two special directories will be in tags. First, when initially migrating an application, the AWIPS 1 version should be tagged in an awips1 subdirectory. The second special directory is "latest\_stable". This will be used as a convention for users to obtain the latest release. This will just be a copy of the the latest version in tags.

In this example, the original AWIPS 1 version of myApp was 2.2. The first AWIPS 2 version is 3.0. There was also an update to 3.1 and latest\_stable is a essentialy copy of 3.1. Users of the application can always get the latest release by going directly to latest\_stable without having to list and determine the latest version numbers. This also allows beta test versions to be tagged, but latest\_stable to remain as the still the last official version.

* apps
  + myApp
    - branches
    - tags
      * awips1
        + 2.2
      * 3.0
      * 3.1
      * latest\_stable
    - trunk

To assist with tagging AWIPS 2 versions of applications and maintaining the latest\_stable directory, the utility script [tag\_release.sh](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/AppSvnTagRelease) should be used.

Remember that Subversion stores changes very efficiently so tagging does not really add a lot of space or overhead to the repository. Of course, if you check out the entire tags directory from the repository, the result could be quite large.

**Example Layout**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#ExampleLayout)

Here's an example. Not all directory levels are shown.

* apps
  + appTextDb
    - branches
    - tags
    - trunk
  + UErunner
    - branches
    - tags
    - trunk
* lib
  + python
    - somePythonLibrary
      * branches
      * tags
      * trunk
    - someOtherPythonLibrary
  + tcl
    - someTclLibrary
  + perl
    - somePerlLibrary
* gfe
  + procedures
    - gfeProcOne
      * branches
      * tags
      * trunk
  + utilities
  + formatters
  + smartinits
  + smarttools
  + apps
* plugins

**Version Numbering**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#VersionNumbering)

Increment the major version number of the application once the port is completed. The configuration management steps involved will typically be as follows:

* Create the trunk directory for a local application by importing the AWIPS I source code.
* Update the trunk by checking in the ported (or partially ported) code.
* When the port is complete, create a tag in the tags directory for the project. Increment the major version number and use that number for the name of the tag.
* Optionally, create a tag for the AWIPS I version. If you do that, put the tag in a subdirectory (called awips1) of the tags directory and use the AWIPS I version number as the tag name.
* It is highly recommended to insert the repository Revision number and date inside all program files. This is basically a one time task when adding a new file. See [SubversionTips](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/SubversionTips) for instructions. This will allow developers and users to unequivocally know what version of a program or file is being used.

**SVN Examples**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#SVNExamples)

Here are some examples of the svn commands you would use in typical scenarios. You will notice that the examples refer to a top-level directory called ladroot in the ncladt repository. We will use the ladroot directory hierarchy, following the structure defined above, until the new Local Applications repository is ready on the VL machine. Then the entire ladroot hierarchy will be moved to the new repository.

The examples use five svn subcommands: checkout, commit, add, cp, and mkdir. The checkout subcommand fetches a working copy from the repository. The commit subcommand commits changes you have made back to the repository.

The working copy is a normal Unix directory hierarchy. It has one special feature: Each directory contains a hidden subdirectory called .svn/. The .svn/ subdirectories contain metadata about the repository and the history of the files in the working copy.

The cp subcommand copies files and directories in the working copy. It has much the same effect as the Unix cp command. In addition to copying data, however, it also updates the affect .svn/ metadata subdirectories, so the history of the copying is preserved. Likewise, the mkdir command updates the affected .svn/ subdirectories as well as creating the specified new directory.

If you use a regular Unix command to create a new file (or directory) in a working copy, that file is said to be "unversioned." It did not come from the repository -- svn knows nothing about it. Svn ignores the file. To make the file part of the working copy you must use the svn add subcommand to add it to the working copy. If the argument to the add subcommand is a directory all the files and directories under it will be added recursively.

Of the five svn subcommands used in the examples, only one of them (commit) alters the repository. The others affect only the working copy. Before a commit command you can use normal Unix commands (ls, find, etc.) to see exactly what you are committing back to the repository. Other svn subcommands can change the repository, but under most circumstances it is good practice to avoid using them.

For more information on Subversion:

* [HowToSubversion](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToSubversion)
* Use the help facility installed with subversion. If, for example, you type "svn help cp" at the command line you will see a brief description of the cp subcommand.
* See the [Red Book](http://svnbook.red-bean.com/): *Version Control with Subversion.*

**Questions and Answers**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#QuestionsandAnswers)

Question: Why import the AWIPS I code? Why not create the trunk from the ported, AWIPS II code?

Answer:

* If you have the AWIPS I code in the repository it will be easy to determine exactly what the starting point for the port was. That is, it will be easy to determined exactly what got ported.
* If it becomes necessary to maintain both AWIPS I and AWIPS II versions in parallel having them both in the repository should save some work.

Question: Why would you want to create a tag pointing to the AWIPS I version?

Answer:

* Basically the same answers as for the previous question: To make it clear where you started and to save work if parallel maintenance becomes necessary.
* Note also that you don't have to create the tag at the time of the initial import. You can create it later by referring to the Subversion revision number of the initial import. But if you create it right away you don't have to figure out what that revision number is, so it's a little easier.

Question: What's all this business about saving work on parallel maintenance? How would having the original code in the repository save work if that became necessary?

Answer: If Subversion has the full history of the code from its AWIPS I starting point then, if and when it becomes necessary, you can create an AWIPS I branch. In that branch you can make whatever changes are required for AWIPS I operations. Then you can "merge" those changes in to the trunk, which has the AWIPS II code.

"Merge" is the term used in the Subversion documentation, but it would be more descriptive to call it "diff and patch." When Subversion does the merge, it will figure out the differences between the original AWIPS I version and the updated AWIPS I version. Then it will apply those differences to the AWIPS II version in the trunk.

There will probably still be some fixups required. But if (for example) the changes for the port to AWIPS II consisted of changing pathnames in many locations and the changes for AWIPS I operations were in the "science" part of the code, then the merge should update the science parts of the AWIPS II code without undoing the new pathnames.