Chief Programmer’s Guide
for All Tau Projects

Jennifer Dempsey, Andrey Kvochko, Kyriacos Pavlou, Nathaniel Gephart,
Tapasya Patki, Amit Shrestha, and Richard T. Snodgrass

August 5, 2014

TD-1

A Tau Technical Document
Chief Programmer’s Guide for All Tau Projects


June 5, 2012
October 10, 2011

TAU Participants

Faculty
Richard T. Snodgrass (Director)
James R. Cordy (Queen’s University)
Sabah A. Currim
Clayton T. Morrison

TAU Coordinator History
Jennifer Dempsey (June 2014–present)
Young-Kyoon Suh (January 2014–March 2014)
Andrey Kvochko (March 2012–December 2014)
Kyriacos Pavlou (August 2011–March 2012)

Chief Programmers
Jennifer Dempsey (τPSM, LoCUS)
Brian Godshall (SPOCK, τDOM, τXSHEMA, AMELIE)
Young-Kyoon Suh (AZDBLAB)
Stephen Thomas (IBM Ottawa, τBENCH)

For additional information, see The TAU Homepage:
URL: http://cs.arizona.edu/projects/tau

TAU is an umbrella project comprised of a number of inter-related and complementary projects, all with the goal of providing to users, through sophisticated user languages and APIs, facilities to manage time-oriented data. The name of the project was inspired by three associations. The acronym is “Temporal Access for Users” which concisely describes the project. The project name is the English spelling of the Greek letter “τ”, which is commonly used in scientific formalisms to denote time. And the Greek letter resembles an umbrella, emphasizing the sub-projects comprising this overall project. Our goal is to realize the theoretical advances over the last decade in temporal databases in practical, efficient, correct, well-engineered and documented interfaces that can be used by the application developer or database user. A second, consistent goal is to leverage each system with the other systems, to provide comprehensive support. Hence, the temporal data types and granularities made available in τZAMAN can be utilized in XSHEMA directly and also in τXSHEMA, the latter to describe time-varying XML data, stored efficiently by τBDB, supported in Persistent Stored Modules in τPSM, queried and returned as a result by τXQUERY, evaluated for correctness and performance in τBENCH, and utilized in τDOM when manipulating time-varying XML documents.
Contents

1 Introduction .................................................. 1

2 Chief Programmer’s Responsibilities .................. 3

3 Release Management ........................................ 5
   3.1 When to Release ........................................ 5
   3.2 Version Numbering ..................................... 5
   3.3 Designating a Release .................................. 6
   3.4 Deploying a Release ................................... 6

4 Integrating Students into the Project .............. 7

5 SVN .......................................................... 9
   5.1 Creating a Workspace ................................... 9
   5.2 Creating a New SVN Repository ....................... 9
   5.3 Creating a New SVN Project ............................ 10
   5.4 Migrating an Existing CVS Repository to SVN .... 10
   5.5 Creating Branches and Tags ........................... 10
   5.6 Removing Branches and Tags ........................... 11
   5.7 Reintegrating a Branch into the Trunk ............ 11
   5.8 Renaming a Repository ................................. 11
   5.9 Guidelines for Managing Branches and the Trunk 12

6 Overall Structure of the Tau Project Directory .... 13

7 Documents .................................................. 15
   7.1 Style File ................................................ 15
      7.1.1 Tau Overall Documents ............................ 15
   7.2 Obtaining \LaTeX source ................................ 15

8 Project Website .............................................. 17

9 Using JAutodoc .................................................. 19

Appendices .................................................. 23
   A CVS (Legacy) .................................................. 23
      A.1 Creating New Projects ............................... 23
      A.2 Creating Branches/Tagging in the Repository . 23
      A.3 Untagging in the Repository ......................... 24
      A.4 Fossilizing CVS Repositories ....................... 24
Chapter 1

Introduction

Welcome to the TAU project. Please read the TAU Getting Started Guide before reading the current TD. The purpose of this document is to introduce new TAU Chief Programmers to their responsibilities, give guidelines on project management conventions, introduce chief programmers to the tools used in the TAU project. Chapter 2 briefly overviews the Chief Programmer’s responsibilities. In Chapter 3, we define the rules of version changes of TAU projects as well as various techniques that are used to manage software releases. The next chapter describes the general methods of integrating new students into the project. In Chapter 5, we provide some of the SVN commands that the Chief Programmer will need to manage the project’s SVN repository. The “TAU Project Directory Structure” chapter examines the structure of the /cs/projects/tau/ folder. The “Documents” chapter provides a brief overview of some of the packages that are used in TAU technical documents. Chapter 8 gives guidelines for managing the TAU project website, as well as suggestions to its desired structure. The next chapter introduces you to the JAautodoc Eclipse plugin. The Appendix describes several CVS commands that may be useful when dealing with legacy projects.
Chapter 2

Chief Programmer’s Responsibilities

Responsibilities for each Chief Programmer (CP) include the following.

1. Maintaining the project’s web site:

   Included is keeping the list of participants up to date on the webpage.

2. Keeping project info up to date. Project info is in /cs/projects/tau/projectInfo/Summary.


4. Choosing the project’s logo.

5. Mantis managing, as described in the Getting Started manual.

6. Managing the SVN repository.

7. Managing executable versions (hence, for example, there is a separate directory in
   /cs/projects/tau/installations/azdblab/ and /cs/projects/tau/releases/azdblab/ for each major release, with the directory name including the date and version).

8. Maintaining the style file for the TAU project documents, e.g. DRAGOON style file. Included is keeping the list of participants in the style file up to date. For more details, see chapter “Technical Documents” in the TAU Getting Started Guide.

9. Managing the schema of the project. The schema should be fully documented in a separate project-specific document number, with an EER diagram, logical schema description, and notes on anything special done on the mapping to logical (e.g., surrogate keys, temporal annotations).

10. Maintaining the README and LICENSE files.

11. Maintaining the sodb hardware lab: AZDBLAB Chief Programmer.
Chapter 3

Release Management

3.1 When to Release

Whenever there is a schema change that change automatically results in a major version change because the code will not work on prior data (hence, for example, AZDBLab went from version 4.3 to 5.0 when the schema changed).

A minor release is planned by the CP when

- it is desired that external users have access to new functionality not in the current release, or
- if a bug in the current release needs to be fixed for external users.

Note, some changes, such as to an internal API, might have far-reaching implications, requiring lots of changes throughout that will take some time. In that case, it might be wise to do a minor release immediately before starting an API change or immediately after.

All Mantis issues under the tag for the next release must have been resolved before the release can occur. This can be ensured by simply retagging an issue to the next minor release, performed by the CP in consultation with Rick.

3.2 Version Numbering

Whenever the next version of the project, either major or minor, is planned, a Mantis tag is created for the version. The name of the tag consists of the name of the project, version number and a planned date of release, e.g. Locus Version 5.1 - by May 2012.

For a release, a tag is created in SVN. The name of the SVN tag consists of the name of the project and a version, e.g. locus-version-5-1.

When appropriate, tools should check their version with the version of data or database, say, in labs or in the labshelf and exit if there is a mismatch. For example, AZDBLab checks its version with the major version, so version 5.2 of AZDBLab is consistent with version 5.0 of the schema.

Components, such as plugins or daemons, should have the same version (cf. Dragoon) or can have their own version which needs to be checked for compatibility with the version of the main program (cf. AZDBLab).

A project could use another TAU project (e.g. DRAGOON uses τBDB). In this case, the compatibility between the project’s release number and that of the used project should at minimum be stated in the project’s Chief Programmer Manual and at best checked at runtime automatically.

The same holds for third-party components, e.g. BeeCrypt for τBDB.

Tools should have some way of displaying their version (say, through the GUI when first executed, or on system.out when they are first invoked).

Each project should have a source file called Metadata to contain static variables holding various data about the project. e.g. version, release date, title, list of authors, description, etc.
3.3 Designating a Release

Each TAU project should have a script called `releaseProjectName.sh` that facilitates designating a software release. The script will be run in the tau project directory, i.e. the directory that contains the SVN working copies. This script assumes that the CP has already done all the other things (e.g., change the version in metadata source file, all the mantis issues have been resolved for this release) before running this script.

This script will perform the following operations.

1. Do a sanity check on the release number for monotonicity (a version number must be bigger than the current one) and to avoid holes (new version must be either a major release, or the closest following minor release).

2. If this a major release, the script will verify with the person this is what is desired.

3. Check to ensure that none of the working copies have local modifications that have not been committed.

4. Check that the local copy is at the latest release in the trunk.

5. If these checks all succeed, it will go into each working copy in that TAU project directory and modify each source file header to record the release number via a `sed` script.

6. Commit to SVN all the working copies.

7. Create a tag of that release number in SVN and copy everything from the trunk into the tag.

3.4 Deploying a Release

Each TAU project should also have a `deploy` script that will create an installation from the specified working directory. Installation is a folder that holds everything needed to install the software, e.g. with a `$ make install` command. This folder may contain binaries as well as source files. Installation should then be copied to `/cs/projects/tau/installations` directory, adhering to this folder’s structure, as described in chapter
Chapter 4

Integrating Students into the Project

TAU projects are especially challenging in that each relies on students, whether at the undergraduate, masters, or doctoral level, to move the project forward. Such students have a wide variety of interest levels, knowledge, and prior software development experience. Undergraduate students will generally have had CSc 335, so they will understand Java, Java Swing and collection frameworks, UML, and design patterns. Graduate students will often have a more varied experience, or lack thereof, of these technologies. All students will generally be experiencing the project code base for the first time, as well as the conventions, procedures, and expectations of a TAU project. Finally, these students are with the project often for just one semester. The combination of relying on people of widely differing abilities and knowledge and experience with high turnover can be a recipe for disaster.

There are two conflicting goals in play. The first is that each student should be afforded a productive learning experience, to the extent possible, allowing them to grow intellectually and further their mastery of software development and the project’s technologies. Certainly, having experience with such large, sophisticated, complex projects can provide an invaluable opportunity to such students. The second goal is that each student should contribute to the project’s concepts and code base, commensurate with the amount of management time by Rick and the Chief Programmer (CP) as well as the other students on the team for that project.

These goals if not properly managed can conflict. For example, there is the danger that if a student is given free rein then he or she could actually move the project backward. For example, the student could introduce poor code that must later be removed, resulting in slowing the progress of the project. Alternatively, the project could be protected by giving a student low-level tasks for their entire tenure, which doesn’t serve the student.

The proper approach is a gradual and thoughtful broadening of assigned tasks and of authority. The student should initially be assigned highly constrained tasks with little possibility of damaging the system: e.g., test cases, documentation, trying out the system, trying out installation of the system. Upon closing each issue (via changing to “feedback”), the CP is thus given the opportunity, and the responsibility, to carefully check the student’s work. The CP should ensure that the work indeed contributes to the project, while simultaneously evaluating that student’s abilities and performance. The CP should go through the task deliverables with the student, so that the student learns what s/he did right and of any concerns.

As the student demonstrates mastery, the CP can assign more challenging tasks, each granting the student an opportunity for growth and to show his or her expertise. In this way, the technical level of the tasks and their potential exposure to the project can be matched closely with the student’s displayed abilities.

The longer the student is with a project and the more trust that student engenders with the CP, the more open-ended the assignments and the less direct oversight by the CP at task resolution. With sufficient time and generated trust, the student can potentially be given wide-open tasks with little oversight by the CP. But that trust must be earned.

On the other hand, if a student does poorly on a task, the task priorities should be readjusted to give the student more time on easier tasks. For students who continually under-perform, it may be necessary to dial back the tasks to those of introductory students, with the requisite careful evaluation at task
resolution.

In this way, the CP, in close consultation with Rick, serves as a mentor for the student, challenging them with slightly more difficult tasks, while also monitoring them carefully to ascertain the evolution of their abilities.

One last piece of advice to CPs: do not assume that the student shares your enthusiasm for the project. Some students are just trying out computer science research as well as the project. The student must *demonstrate* ability and work ethic to be given more interesting assignments, with a big dose of skepticism mixed with possibility at the beginning.
Chapter 5

SVN

When following the instructions given in this chapter, complete the steps in Sections 5.2 and 5.3, unless, there is a need to migrate a project from an existing CVS repository, then jump to Section 5.4.

5.1 Creating a Workspace

All Tau projects should have a script that can be used to fetch the project. Fetch script should be named fetchProjectName.sh. This script is located in the project’s SVN repository, within the Scripts folder (please, refer to chapter “Guidelines to SVN Repository Structure” in the Tau Getting Started Guide). It is suggested that, Tau project participants use the following commands in order to retrieve the fetch script:

cd /cs/projects/tau/live/username
svn export path/fetchProjectName.sh.

Note, path is the path to the repository and ProjectName is the path to the fetch script inside the repository. For example, to obtain the LoCuS fetch script, use the following commands.

cd /cs/projects/tau/live/username
svn export svn+ssh://username@cvs.cs.arizona.edu/cs/svn/tau/locus/trunk/Scripts/fetchLocus.sh

5.2 Creating a New SVN Repository

To create a new SVN repository, follow these steps:

1. Log in to lectura.

2. Execute the following command:

   svnadmin create /cs/svn/tau/repository_name
   chmod -R 775 /cs/svn/tau/repository_name
   mkdir TMP (Create a temporary directory).
   cd TMP
   svn checkout svn+ssh://username@lectura.cs.arizona.edu/cs/svn/tau/repository_name
   cd repository_name
   svn mkdir trunk
   svn mkdir branches
   svn mkdir tags

   New repository should now be accessible.

   Note: All SVN repositories under the umbrella project should be located in the /cs/svn/tau directory. All repository names are in lower case.
For example, to create a HelloWorld project repository, execute the following.

```
svnadmin create /cs/svn/tau/helloworld
```

To create a HelloWorld paper repository, execute commands given below.

```
svnadmin create /cs/svn/tau/papers/helloworld
```

To delete an SVN repository, delete the folder specified in step 2.

### 5.3 Creating a New SVN Project

To create a new SVN project perform the following steps.

1. Create a new folder on your local machine, for example:
   ```
   mkdir ./proj
   ```
   Note that `proj` is the name of the project you are creating.

2. Execute the following command (`locus` repository is used as a reference).
   ```
   svn import ./proj svn+ssh://username@lec.cs.arizona.edu/cs/svn/tau/locus/trunk -m "Creating new project"
   ```
   where `username` is your department login.

3. If you want to continue to create an Eclipse Java Project from the new SVN project, follow steps 4 to 6.

4. Make sure you have Subclipse plugin installed. For Subclipse installation instructions, see the TAU Getting Started Guide.

5. Go to Eclipse, inside the `Window` menu select `Show view` and click `SVN Repositories`. Right click on your newly created SVN project and click `Checkout`, select `Check out as a project configured using the New Project Wizard` Select `Java Project` as the wizard type and follow the given instructions.

6. Commit the project for others to see.

### 5.4 Migrating an Existing CVS Repository to SVN

In order to migrate from CVS repository to SVN, please, use the following steps:

1. Log in to `lectura`.

2. Execute the following commands:
   ```
   cd /cs/svn/tau/
   ./tools/cvs2svn/cvs2svn -s /cs/svn/tau/project-name CVS-repository-path
   ```
   where `project-name` is the name of the project under TAU umbrella project, and `CVS-repository-path` is the path of the source CVS repository.
   For example, to convert the `helloworld` repository to an SVN repository, execute
   ```
   cd /cs/svn/tau/
   ./tools/cvs2svn/cvs2svn ./helloworld ../../cvs/helloworld
   ```

### 5.5 Creating Branches and Tags

Whenever we make a new release (or perhaps when the code reaches a stable state), we should create a tag with the current repository content. Tagging is helpful because we can checkout or restore to any tagged version of the repository whenever needed.
A separate branch is usually created for every member of the project. Note that small changes or bug fixes can go directly into the trunk, while the development of new features that require thorough testing and may take a while should be done in a separate branch.

To tag the repository execute the following commands. Note, `tag_name` is the name of your tag (for example `version-4`) and `locus` is the project (the following is all one line).

```
svn copy svn+ssh://username@cvs.cs.arizona.edu/cs/svn/tau/locus/trunk
svn+ssh://username@cvs.cs.arizona.edu/cs/svn/tau/locus/tags/version-4 -m "Tagging release 4.0"
```

where `username` is your department login.

Branches are created the same way, only in the branches folder of the root of the repository, rather than, tags.

### 5.6 Removing Branches and Tags

To remove a particular tag (for example if the tag was created by error or you have a good reason to remove the tag), execute the following command. Note that `tag_name` is the name of your tag (for example `version-4`) and `locus` is the project.

```
svn delete svn+ssh://username@cvs.cs.arizona.edu/cs/svn/tau/locus/tags/version-4
```

where `username` is your department login.

### 5.7 Reintegrating a Branch into the Trunk

To transfer the changes made in a branch back to trunk, execute the following. Note that in the example `branch-qsim` is the branch and `locus` is the project.

1. Make sure you have an up-to-date working copy of the trunk.
2. Execute the following command:
   ```
   svn merge -reintegrate svn+ssh://username@cvs.cs.arizona.edu/cs/svn/tau/locus/branches/branch-qsim .
   ```
3. Build the project and make sure it works correctly.
4. Commit changes to the repository:
   ```
   svn ci -m "Reintegrating branch-qsim into trunk"
   ```

### 5.8 Renaming a Repository

The correct way to rename a repository is to create a repository with the new name, and then import all revisions from the old repository into the newly created one. This way, all the revision history will be preserved. To do this, follow these steps.

1. Create a new repository with the following command.
   ```
   svnadmin create /cs/svn/tau/new-repository-name
   ```
   where `new-repository-name` is the new name for you repository.

2. Dump all the data from the old repository into a dump file.
   ```
   svnadmin dump /cs/svn/tau/old-repository-name > old-repo.dump
   ```

3. Load the dump file into the new repository like this.
   ```
   svnadmin load /cs/svn/tau/new-repository-name < old-repo.dump
   ```

4. Remove the dump file and the old repository.
   ```
   rm old-repo.dump
   rm -rf /cs/svn/tau/old-repository-name
   ```

Note that in case of a large project this process might take a while.
5.9 Guidelines for Managing Branches and the Trunk

We do not want to allow a branch to accumulate too many changes and commits after its creation. In order to avoid this whenever a set of changes is made to a branch that branch should be merged back into the trunk. Then a new branch based on the updated trunk will need to be created. However, if another branch’s changes do not conflict with those of the merged branch, then it is not necessary to create a new branch for the former. At some point, the trunk should also pick up all the changes from both branches, and new branches should be created to replace both of the merged branches from the up-to-date trunk. Additionally, with each new release (major or minor) every branch should be reintegrated and re-created for that release.
Chapter 6

Overall Structure of the TAU Project Directory

This chapter examines the structure of the /cs/projects/tau/ folder. Figure 6.1 shows the structure of this directory and is elaborated on below.

There are six directories under /cs/projects/tau. 

Installations

An installation is a folder containing everything necessary to run the software. Under the installations directory there are folders for each project (e.g., azdblab). Under the project-specific directory there should be named releases (e.g., 20111123 v5 19) and under that—tar files (or VM images as required by some projects like Dragoon). Also keep in mind that when we create a release though SVN we just create a tag of the existing files in SVN. Do not tag the tar file. For an internal release place the tar file under a separate directory, namely, /cs/projects/tau/deployments/projectName.

releases

Under releases there will be a similar structure as installation, but with sources rather than executable files.
live

A live directory is where the working directories for each TAU participant are. Subdirectories’ names should be usernames of TAU participants. How to organize the working space inside the user’s folder is up to the user.

fossilized

This directory contains the fossilized workspaces of users that are no longer in TAU, as well as TAU projects that are no longer maintained.

projectInfo

This directory holds project information specific to the University of Arizona (e.g., passwords to private websites, list of machines that can be used to deploy Dragoon, etc).

third_party_releases

This folder contains any third-party software that may be required by a TAU project. This directory should contain subdirectories, one for each release, e.g., Tetrad5.3, with the jar file inside that directory. Additionally there should be a README file stating where we obtained the third-party software and when.
Chapter 7

Documents

PDF versions of technical documents can be found at http://www.cs.arizona.edu/projects/tau/technicaldocuments.

7.1 Style File

One of the responsibilities of the Chief Programmer is to maintain the style file for his or her project. This style file is derived from the tau.sty file for the Tau project.

TAU style file uses the minted package to display listings of code. This package is not a part of most of the \LaTeX\ installations and thus should be installed separately. minted is already installed on the lectura server. For instructions on how to install minted on a different machine, please, refer to the minted package documentation, which can be found at http://minted.googlecode.com/files/minted.pdf. The minted package itself, as well as its proper parameters used throughout the TAU projects, are described in more detail in chapter “Technical Documents” of the TAU Getting Started Guide.

7.1.1 TAU Overall Documents

The Chief Programmers are responsible for the TAU overall documents. For guidelines on formatting TAU documents, please refer to the “Technical Documents” chapter of the TAU Getting Started Guide.

7.2 Obtaining \LaTeX\ source

There is a separate SVN repository for TAU technical documents, which can be found at /cs/svn/tau/tau. See the chapter “Using SmartSVN” of the TAU Getting Started Guide for how to check out this repository using SmartSVN.

Figure 7.1 shows the structure of the trunk of the repository at /cs/svn/tau/tau

Level 1

At the top level, the tau project repository contains a single LocalDocs directory.

Level 2

Under the LocalDocs directory there are:

- **documentName** - directories containing .pdf and source files for a particular document. tauChiefProgrammersGuide is shown here as an example.

- **tau.sty** - A style file for the TAU project. The style file for a particular project should be based on this tau.sty file.
Level 3

At this level, document folders contain source files for the documents, figures and any other resources needed to compile the document, as well as their current pdf versions.
The Chief Programmer should maintain the project’s website. The complete list of the websites for the projects can be found in Chapter 2.

The website should contain the following:

- A brief overview of the project, its purpose and goals, the current stage in development, as well as explanation the underlying theories and general usage techniques.

- The complete list of past and presents participants of the project. This list should be divided into at least five groups: Faculty, Graduate Students, Previous Graduate Students, Undergraduate Students and Previous Undergraduate Students.

- List of publications.

- Latest news for the project: conference talks, new publications, new release information, etc.

- Funding information, for example, the NSF grant number.

- Software section should provide download links to the latest release of the project. The latest release under /cs/projects/tau/releases/ should be mirrored in this section. It should also provide the links to various documents that the end-user may require to use the software (installation instructions, architecture diagrams, etc). Note that local documents should not be copied to the webpage.
Chapter 9

Using JAutodoc

To add file headers to existing projects, one may use JAutodoc. JAutodoc is a plugin to Eclipse that generates Javadoc comments from existing code. The JAutodoc Update site for installing from Eclipse is [http://jautodoc.sourceforge.net/update/](http://jautodoc.sourceforge.net/update/). The installation process is similar to that described in the Section “Setting up Eclipse for SVN” of the TAU Getting Started Guide.

To add file headers, follow these steps.

1. In Eclipse, go to **Window → Preferences**.
2. Under **Java**, select **JAutodoc**.
3. In **File Header** section check **Add file header**, **Replace existing header** and click **Edit**.
4. Modify the header.
5. Click **OK** in Preferences window click **OK**.
6. In **Project Navigator**, right click the project you wish to add header to.
7. In the dropdown menu, navigate to **JAutodoc** and click **Add Header**.

Files should now have the header specified in step 4.

For projects that do not use Eclipse, the CP must add the header manually.

For details on the header structure, please refer to chapter “Setting Up Eclipse for SVN” of the TAU Getting Started Guide.
Appendices
A CVS (Legacy)

All active Tau projects have been moved to SVN. This appendix applies only to legacy projects.

A.1 Creating New Projects

To create a new CVS project perform the following steps.

1. Log in to lectura.
2. Make sure CVSROOT is defined (see the Getting Started Guide: General, Chapter 1).
3. Execute the following commands. Note that proj is the name of the project you are creating say in the repository locus.
   cd /cs/cvs/tau/locus
   mkdir proj
   cd proj
   cvs import -m "new project" proj locus start
   cd ..
   chgrp tau proj -R
   chmod 770 proj -R
4. If you want to create an Eclipse Java Project from the new CVS project follow steps 4 and 5.
5. Go to eclipse, inside the Window menu select Show view and click CVS Repositories. Right click on your newly created CVS project (under Head, under locus) and click Check Out As, select Check out as a project configured using the New Project Wizard. Select Java Project as the wizard type and follow the giving instructions.
6. Commit the project for others to see.

A.2 Creating Branches/Tagging in the Repository

Whenever we make a new release (or also perhaps when the code reaches a stable state), we should tag the current repository content. Tagging is helpful because we can checkout or restore to any tagged version of the repository whenever needed.

If possible always tag from the checkout available at /cs/cgi/projects/locus/repo in lectura. This is a special checkout as described earlier.

To tag the repository perform the following steps.

1. Log in to lectura.
2. Make sure CVSROOT is defined (see the Getting Started Guide: General, Chapter 1).
3. Execute the following commands. Note that tag_name is the name of your tag (for example version-4) and locus is the project.
   cd /cs/cgi/projects/locus/repo
   cvs update
   cvs rtag tag_name locus

To list the tags on a particular file, use the command cvs status -v filename
A.3 Untagging in the Repository

To remove a particular tag (for example if the tag was created by error or you have a good reason to remove the tag), perform the following steps:

1. Log in to lectura.
2. Make sure CVSROOT is defined (see the Getting Started Guide: General, Chapter 1).
3. Execute the following commands. Note that tag_name is the name of your tag (for example version-4) and locus is the project.
   
   cd /cs/cgi/projects/locus/repo
   cvs rtag -d tag_name locus

A.4 Fossilizing CVS Repositories

As Tau project moves from CVS to SVN, old SVN repositories are fossilized. Repository is said to be fossilized when the following operations are executed.

1. A tarball of the repository is created and moved to /cs/cvs/tau/fossilized
2. Write bit of the tarball is removed for everyone but the owner of the file.