

GLAST SSC CVS Repository

This document describes the use of the GSSC CVS repository, its layout and general guidelines for the layout of directories within GSSC developed packages. For detailed information about using CVS look at the CVS Manual which can be found at the CVS website: <https://www.cvshome.org/>.

CVS Repository Location

The GSSC's CVS repository is located on `daria.gsfc.nasa.gov` and maintained by the LHEA system administrators. Use of the repository requires a username and password. The CVSROOT for our repository is

```
:pserver:username@daria.gsfc.nasa.gov:/glast
```

CVS Repository Access

Here's a brief tutorial on using the CVS server. It is aimed at a developer who has write permission in the `glast` area only. Start with

```
cvs -d :pserver:username@daria.gsfc.nasa.gov:/glast login
```

which will prompt you for your CVS password. Once you've logged into a particular repository from a particular machine you won't need to repeat the login step. Then

```
cvs -d :pserver:username@daria.gsfc.nasa.gov:/glast checkout GSSC
```

should check out the whole GSSC package directly from its repository and put it in your local directory under `GSSC/`.

If you want to avoid having to type the `-d :pserver:username@daria.gsfc.nasa.gov:/glast` every time you start you should set the CVSROOT environment variable. ie.

```
setenv CVSROOT :pserver:username@daria.gsfc.nasa.gov:/glast (csh version)
export CVSROOT=:pserver:username@daria.gsfc.nasa.gov:/glast" (sh version)
```

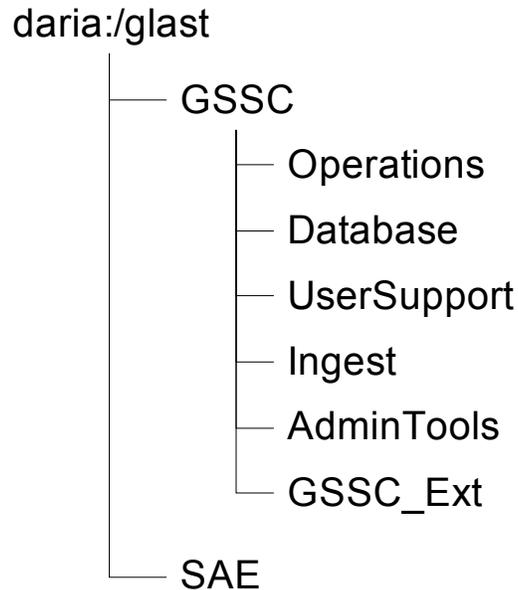
Once you've got things checked out you can skip the `-d :pserver` business when executing commands inside a checked-out tree, ie:

```
cvs log mytool.c
cvs update
cvs diff
cvs commit
```

should all work as you'd expect. On any properly configured machine one should be able to execute `"info cvs"` to get detailed help on using CVS.

Top Level CVS Directory Structure

All GSSC software being developed will be stored in the GSSC directory under the head glast directory in the repository. This will allow the checkout of the entire GSSC software system with a single command to checkout the GSSC directory as illustrated above. Under the GSSC directory the following directory structure will exist:



There are other directories under the daria:/glast head from previous usage of the CVS repository. This code will eventually be either removed or moved into the appropriate place in the new data structure. The SAE directory will be used to hold the SAE code once maintenance of that code becomes the responsibility of the GSSC later in the mission.

All code will be checked into the CVS repository under one of these six directories as follows:

Operations: This directory will contain all the software tools from the Operations subsection.

Database: This directory will contain all the software tools from the Data Archive and Software Support subsection

UserSupport: This directory will contain all the software tools from the User Support subsection

Ingest: This will contain all the software associated with the data ingest system such as OPUS scripts, DTS scripts, etc.

AdminTools: This directory will contain the software tools for administration of GSSC software and other general tools such as the Nightly Build system, the Paging tool, etc.

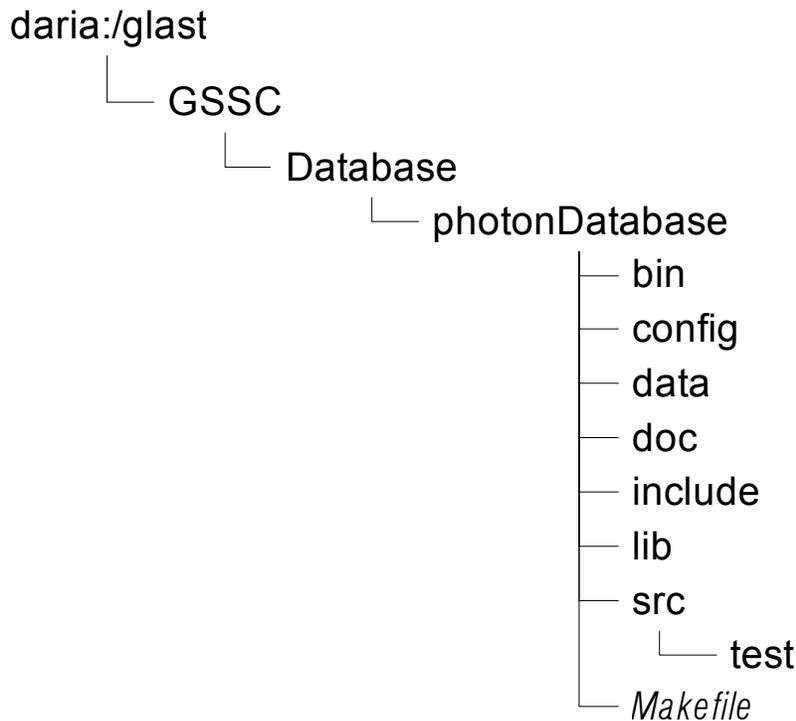
GSSC_Ext: This directory will contain all external libraries used by the other GSSC software such as MPICH, HTM, etc.

Package Directory Structure

Packages checked into the repository should have a descriptive name. For each package developed at the GSSC, whether it is a tool or a library, the directory structure inside the package should contain directories from the following list:

- bin: The directory to hold the executable scripts or binaries. This directory should be empty in the CVS repository as it will be populated by the build system.
- cmt: The directory to hold the CMT config and requirements files if CMT is used.
- config: The directory to hold any configuration files needed by the tool to execute. This includes PIL parameter files if appropriate.
- data: This directory is to contain any test data needed to test the software.
- doc: This directory is to contain any documentation for the package including the doxygen Doxyfile when appropriate.
- include: This directory is to contain all the header files for the package
- lib: This directory is to hold all the compiled libraries and/or script modules for the packages. Like the bin directory, this directory should be empty in the CVS repository and filled locally by the build system.
- pfiles: This directory would hold pil parameter files if needed by the project.
- src: The directory to hold all the source code and scripts. The script modules are copied from here to the lib directory as part of the build process.
- src/test: This directory is to hold all the code necessary to test the package. The code in this directory will be build and executed as part of the nightly build system.

Additionally, in the main package directory, there should be a makefile that the nightly build system could execute to build the project and run the tests. Thus the directory for the photon database, which doesn't use CMT or pil, would look like this:



Checking a New Package into the Repository

At some point, you have a new package that needs to be checked into the repository. This is done by following these steps:

- 1) Move to the root directory of the package you have created:

```
cd newPackageDir
```

where *newPackageDir* is the directory containing the package code

- 2) Import the package into to repository by issuing the following command replacing the italicized item appropriately:

```
cvs -d $CVSROOT import -m "Initial package import message"  
GSSC/category/newPackageDir GSSC tag
```

The string after the *-m* is a comment about the initial package check-in. If the *-m* and following string are omitted, a text editor will appear asking you to type in a message. The *category* corresponds to one of the six areas specified above in the discussion of the repository. *newPackageDir* is the directory name that your code will be stored in in the repository. Finally, the tag should be the version number of the code you are checking in, typically *v1*.

At this point the code is checked in to the repository and can be checked out, updated, etc using normal CVS commands.

Modifying an Existing Package

Check out the package

To modify an existing package first check the package out of the repository by issuing the following commands:

```
cd root/directory/for/files  
cvs -d $CVSROOT/GSSC/category checkout -d directory package
```

where *root/directory/for/files* is your development directory, *category* corresponds to the sub folder under GSSC in the repository structure, *directory* is the local directory you want the project extracted into and *package* is the name of the package directory in the CVS repository you want to check out.

For example to check out the *photonDatabase* package (part of the Database category) into my Development directory in a subdirectory called *pdb* I would issue the following commands:

```
cd Development  
cvs -d $CVSROOT/GSSC/Database checkout -d pdb photonDatabase
```

Once the package has been checked out, you can edit the files, add files and add directories as needed to develop the code.

Adding Files or Directories to the Project

If you create new files or directories in the project, you need to tell CVS about them so that they are added to the repository. This is done using the *cvs add* command. To add a file or directory to the repository simple type

```
cvcs add name
```

where *name* is the name of the file or directory you wish to add to the repository.

Committing Changes

When you are done editing the project or have made changes that you want to commit into the repository. Issue the following commands to check-in your changes

```
cd directory
cvcs commit -m "Message about change"
```

where *directory* is the root directory of the package that you checked the code out into. If you omit the *-m* flag and message a text editor will appear for you to enter the message into. You should always include some message with a commit command to describe what changes were made to the code.

Use of CMT

As a majority of the internal GSSC software will be scripts and software coming from other build systems, the GSSC will not be requiring the use of CMT as its build manager for the internal GSSC software. However, if a developer wishes to use CMT they may. If they do so, the following Makefile should be included as the Makefile in the root directory of the package (ie on the same level as the *cmt* and *src* directories).

```
# Makefile stub to use as the root directory makefile for CMT
# packages when developing in Eclipse and using the CMT package
# with our nightly build system. This will allow the developer to
# use the build and rebuild Project options from the Project menu
# and still have CMT do the building of the project. It also
# allows the nightly build system to execute a 'make' in the
# primary directory and have the package built.
#
# Created 07/20/04 - Tom Stephens

# A default target to execute if the user executes a make
# with no targets.
all:
    $(MAKE) all -C cmt

# Any specified target is redirected to the cmt directory for
# building.
%:
    $(MAKE) $* -C cmt
```

Figure 1 - Root level Makefile for use with CMT projects