TrigRTA.htm

Trigger Root Tree Analysis Utility

Introduction

This utility is the standard offline analysis framework for trigger test data based on the plain Root Tree Analysis. It contains two subpackages under users /Trg/ package in CVS. A current working development version reside at /afs/slac/g/glast/trigger/current, but this development area may be unstable as it is being updated continuously so that it is recommended to check out production tagged versions for private development. The two subpackages:

RTAutil	this is a lower level utility containing basic Root Tree Analysis utilities such as formatted dump of various digis, recon objects and GEM data, GEM event time, ideal geometry etc.
TrgTest	this contains the actual higher level trigger test analyses, all constructed as classes inheriting from the (slightly modified) standard RootTreeAnalysis class.

These packages have standard CMT structures setup to build sharable ROOT libraries. The reasons for going to the ROOT sharable libraries:

- Once compiled externally, don't have to recompile whole image with CINT each time executing it in ROOT.
- The CINT compiler optimization is very poor so that the externally compiled shlib with g++ is much faster in execution.
- Some part of the official digi code (probably Trk hit Digi) can cause fatal memory overwrites and crashes if compiled by CINT.

This utility is only tested for running on Linux platform (Noric) at SLAC. To work with these utilities, the basic setup needed are just the standard GLAST ground user .cshrc and ROOTSYS and lib paths defined for running ROOT. An example of this is /afs/slac/g/glast/trigger/glast.setup.

Usage and Development through CVS and CMT

CVS Repository

The code CVS repository reside at the $CVSROOT/users/Trg/[Packagename] (CVSROOT is /afs/slac/g/glast/ground/cvs on SLAC UNIX). As part of the "users" code, they are not attached to any office release so that you need to check them out and compile yourself for all usage. Because the packages live under users/ subdirectories, some care needs to be taken to not accidentally checkout or tag the whole users package with many people's code (you will regret it!). The general usage of the users packages are explained by Toby's posting. As an example, if you want to develop the TrgTest code, starting from tag v0-01, under your own release directory *myRelease,* then the cvs checkout command is:

```
cd myRelease
cmt co -r v0-01 -o users/Trg TrgTest
```

The -o option indicates the directory offset for the package. This should make a directory myRelease/TrgTest/v0-01/. Within the package directory, you typically should have a subdirectory with the same name as the package name which contains the class headers; a subdirectory /src containging the class implementation C++ source files and the /cmt subdirectory containing the build options. The TrgTest pacakge also has a subdirectory /workdir for running applications in ROOT. When you commit the revised version for Trgtest, you should also issue the cvs commit from the directory myRelease/TrgTest/v0-00/ or it subdirectories to only commit code within the package. When you want to tag an updated version, again you should be sitting in myRelease /TrgTest/v0-01/ and using the -rtag recommended by Toby for the packages residing in subdirectories:

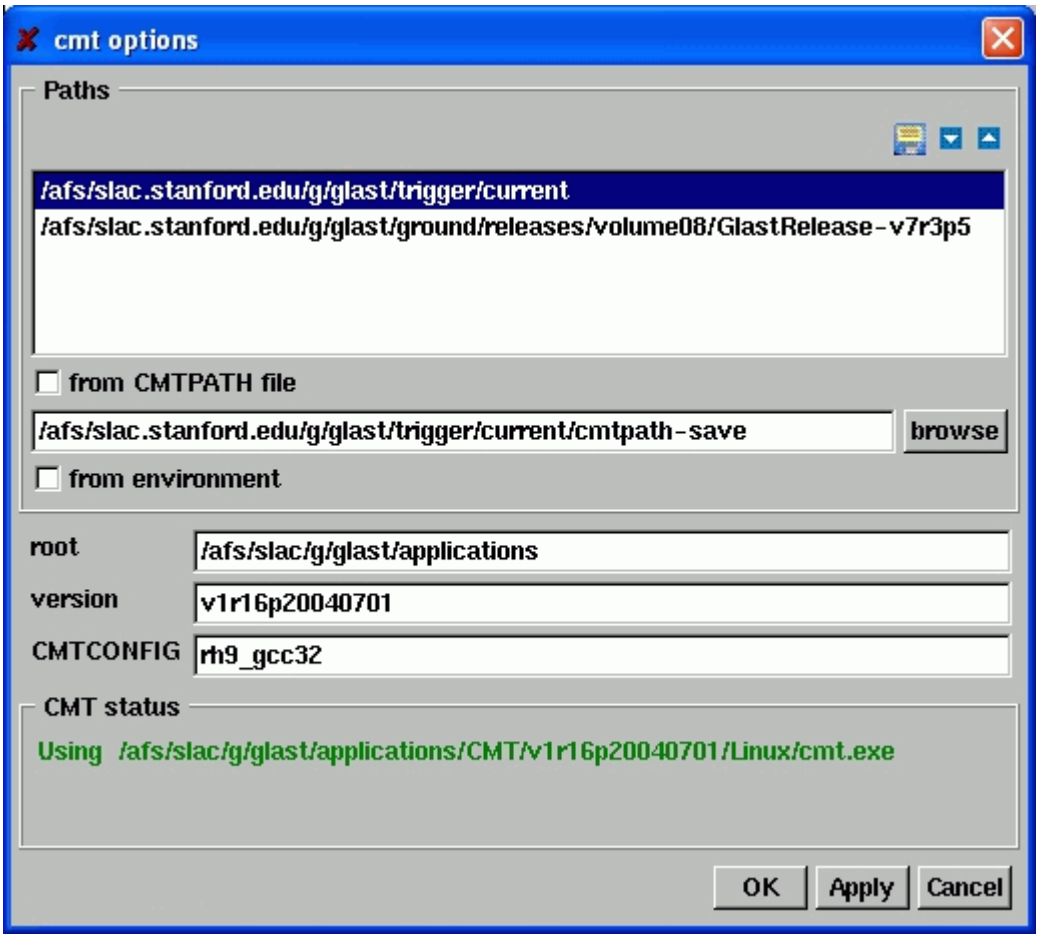
```
cvs rtag TagName users/Trg/TrgTest
```

To check the tags, you need to spell out the full package directory offset:

```
listtag users/Trg/TrgTest
```

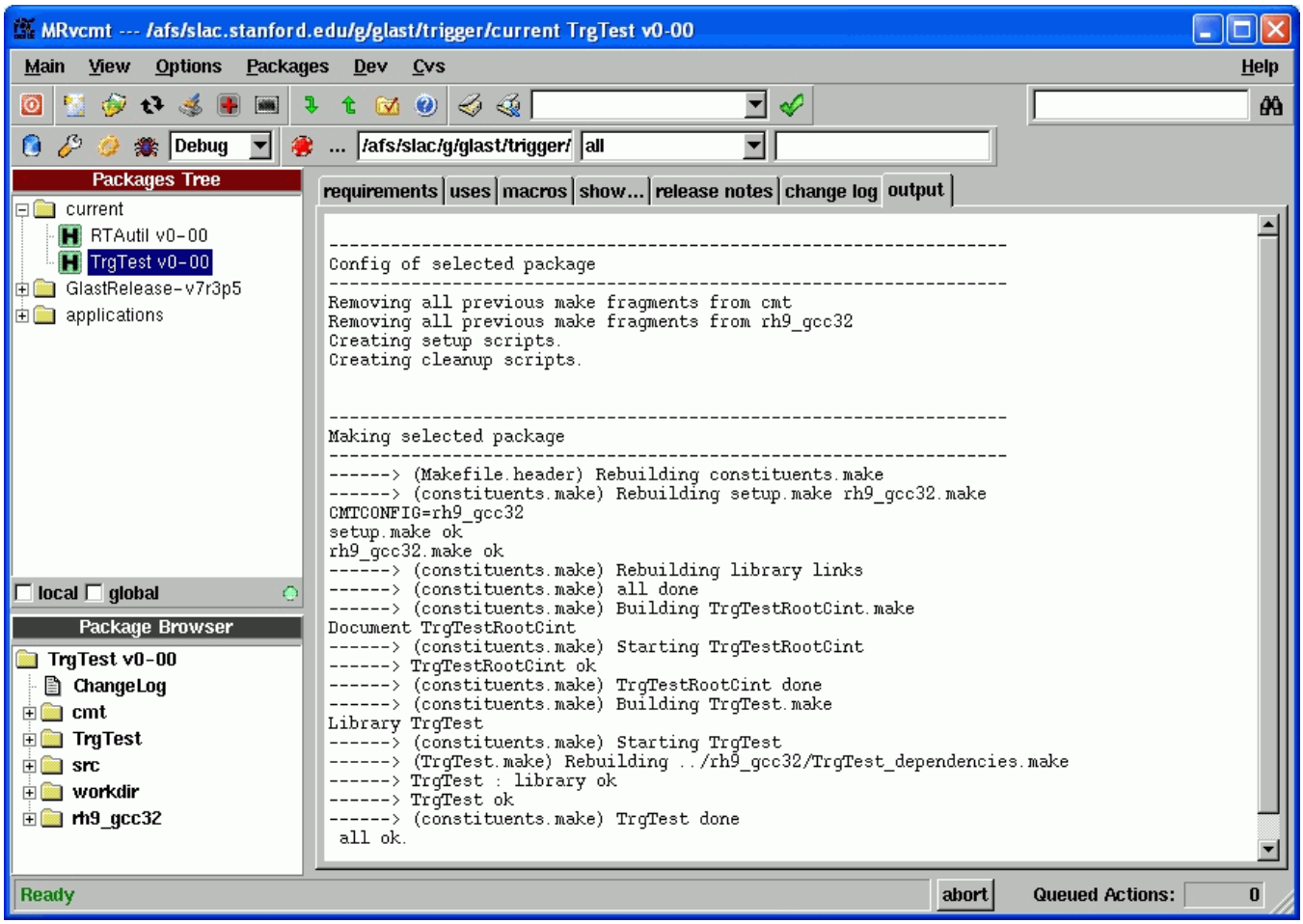
CMT Code Build

For packages checked out from CVS, you will work within CMT to build and update them. The interactive CMT GUI is the recommended tool for doing development. It can be started with the command MRvcmt& after the setup is run. The essense of the CMT operations are introduced in the offline workbookunder the tab: MRvcmt. Upon startup of MRvcmt for the first time you need to configure the paths. From the MRvcmt top menu, select the Options tab then "Cmt Options" to bring up the the following panel:



which is not yet filled. Click the "from CMTPATH file" box and then browse for the /afs/slac/glast/trigger/current/cmtpath-save file. Once located and entered the file, the panel should then be loaded with the contents shown above. Uncheck the "from CMTPATH file" box, then change the top line (highlighted in the picture above) of the paths box to whatever your own current working GLAST release directory, then use the save button (the little floppy symbol on the top right of the panel) to save this configuration to your own release directory as e.g. mycmtpath. MRvcmt has a cached buffer so that next time you fire up MRvcmt, it will reload the same configuration as last time. With this saved file mycmtpath, you can choose to set the cmtpath file field to this file and check the "from CMTPATH file" box to make sure the configuration will load this file instead next time (this may be useful when developing different things under different releases).

Once done with the CMT configuration options and returned to the MRvcmt main panel. As an example of working in /afs/slac/glast/trigger/current as the release directory, and already used cvs to checkout both RTAutil and TrgTest packages, you should see something like this:



with the two packages shown under the release directory. If you didn't checkout the packages yet, you can also try the top "packages" menu to check out (package name TrgTest, Offset users/Trg). You can then click on e.g. TrgTest v0-00 which will set it to be the current working package. For subsequent work of updating and building the selected package, you pretty much only needs to go to the top menu "Dev" tab to do "Make" (may be occassionally "configure" or "clean" in case of new files added or major structural/build option changes). Note that the build process created the platform binary output directory rh9_gcc32, which in particular contains the ROOT sharable library libTrgTest.so. The cmt subdirectory originally only contains the requirements file, while the Make files are generated by the cmt processing.

Updating/Adding Code

For editting an existing file, all you need to do is to select the checked out package in your dvelopement release in the MRvcmt Package Tree, then use the Dev menu "Make" to compile the sharable library (may need to clean/configure first if changed lower level .h file). For adding new files, an example to add a new class named MyTest to the TrgTest package:

- Add the header file Mytest.h to the TrgTest subdirectory, and the MyTest.cxx file to the src subdirectory. The code always needs to be an inherited class of the RootTreeAnalysis so that there are certain accessors such as HistDefine() and Go() you almost always need to provide, and to connect up between the daughter and parent class there are certain lines e.g. MakeHistList in HistDefine() always has to be there (sorry that I couldn't find a way to hide that in the RootTreeAnalysis.h). The best way is to start from e.g. AcdRate class as a template when creating your new class.
- Edit cmt/requirements to add the the line ../TrgTest/MyTest.h to the list of files included for the root shlib headers.
- Edit TrgTest/LinkDef.h to add the line #pragma link C++ class MyTest+; as the shlib access point.
- In MRvcmt Dev menu: configure, then Make.
- You need to quit ROOT and restart to load the new shlib into root (this is controlled by .rootrc->RooLogon.C).

Most of the trigger test analysis code should be just additional classes within TrgTest. In case there is need to create another package under users/Trg, it needs to follow Toby's posting: You can make the NewPackage/ directory first and creating the various subdirectories as needed. However, if you are copying the sturecture of another package to create this new package, you should watchout: ***Only keep the original source files which you want to put into cvs in those directories***. So various files should NOT be present: e.g. all derived files other than the requirements in /cmt; NewPackage_rootcint.* in /NewPackage subdirectory; /rh9_gcc3 binary directory; any derived files/links in workdir etc. While sitting in the Newpage/ main directory, issue command:
cvs import users/Trg/NewPackage *vender* v0
where the *vender* is you initials or username, which will create the package NewPackage at $CVSROOT/users/Trg/NewPackage which you can immediately check. In fact it is useful to make a trial send with cvs -n import ... first, which won't actually send to cvs yet but give you a list of what will be sent (I did mess up once without trying this first).

Running Examples in the Utility

Assuming you are a absolute beginner, trying this on SLAC Linux nodes e.g. Noric, the following prerequisits are needed:

- Check your environment setup if it effectively executed things needed in /afs/slac/g/glast/trigger/glast.setup:

echo $GLASTROOT	/afs/slac.stanford.edu/g/glast
echo $CVSROOT	/nfs/slac/g/glast/ground/cvs
echo $ROOTSYS	/afs/slac.stanford.edu/g/glast/ground/GLAST_EXT/rh9_gcc32/ROOT/v4.02.00/root

- Established a private development release directory and checked out the packages RTAutil (tag v0-00) and TrgTest (v0-01) into that private release directory following the instructions in the "CVS Repository" section above.
- Follow the instructions in the "CMT Code Build" section above to build both RTAutil and TrgTest packages and check the rh9_gcc32 directories of each package to see the the shlibs libRtautil.so and libTrgTest.so are indeed successfully built.
- cd to [your test release]/TrgTest/v0-01/workdir and take look atthe release.setup to see if all the softlink pointers are sensible amd modify if necessary (***this is rather fragile and may not keep up with updates so that you should check this carefully***). Then source release.setup which should setup various softlinks to the sharable libraries and the parent RELEASE pointer in the workdir. The setup script is made to allow making changes to release/shlib by removing old links first so that first time execution may result in a bunch of "no such file or directory" messages which you can ignore. For normal usage, you only need to do this once at the first time setting up the TrgTest package.
- Still in the workdir, you can fire up ROOT (which will load the sharable libs through .rootrc pointing to RooLogon.C) and execute the following CINT macros with simple e.g. .x RunDump.C.

RunDump.C	Formated dumps of digis, recon objects and GEM data (digi/recon)
RunTestGeom.C	Test to validate the ideal geometry utility (recon/histograms)
RunAcdRate.C	Analysis of ACD rate test data (histogram, GEM event time)

  The execution of AcdRate.C will take ~10min to grind through many runs (the job runs at 2-3K events/sec) and create a bunch of output histograms in subdirectory results/AcdRate.
- Still within ROOT, you can try the plotting macro for AcdRate (assuming you executed RunAcdRate.C to produce the root files in results/AcdRate already) e.g.:
  - .x plotAcdRate.C(1,5171) (time counter summary for run 135005171)
  - .x plotAcdRate.C(5,0) (GEM veto hit rate for various FreebBoard combos and thresholds)