Meeting Minutes August 14, 2007

Attendees: Joanne Bogart, Toby Burnett, Jim Chiang, Richard Dubois, Navid Golpayegani, Heather Kelly, Eric Winter

Note: Navid has since progressed in his SCons development and many of the examples contained in the following minutes have changed.

/
SConstruct
externals.scons **List of externals and their versions. Can utilize GLAST_EXT or over-ride individual libraries. Check scons -h for available options.**
(Likelihood) **Directories are in ( )**
(facilities)
SConscript
facilitiesLib.py **<package>Lib.py lives in each package**
(site_scons)
(site_tool) **SCons looks here by default for tools**
(include) **installation location for public headers**
(lib) **installation location for libraries**

automated dependencies

Recommended fix from SCons developers is to introduce a SCons tool (a python function) which is used to update the environment. Tools are located in the site_tool directory by default, this can be over-ridden via an option when invoking the tool. Rather than use the site_tool location, there is a <package>Lib.py file within each package. Similar in function to CMT's "use" statements, this file lists what other packages this package requires. Only lists direct dependencies, SCons figures out the rest.

SConstruct

Next Navid provided a run-through of the contents of the SConstruct file.

The SConstruct file appends to the environment such as cflags. Currently this is implemented in a linux-centric fashion. This will be updated for platform independence in the near future as we need to start testing on Windows.

Next the incDir and libDir are defined, which defines the installation directories. These are the (include) and (lib) directories specified above. Note that while the lib directory is flat, such that all libraries are stored without any subdirectories, the include directory has a structure where we have /include /<package>*.h All public header files will be stored in the include directory. In the case of a container package such as celestialSources, the structure would be /include/GRB/.h, so we omit the celestialSources, just as we do now with CMT. We could expand our definition of installation directories to include pfiles, xml files, jobOptions, etc.

Next a list of packages is specified, similar to a checkout package without explicit versions.

Then a function named registerObjects is defined, which accepts two arguments: packageName and list of objects. This function is called by each SConscript file within each package to register the package's objects. Set up such that we can build any target, and SCons will be able to build any other required libraries as well as the requested target.

Finally, addSubPackages(topLevel, packages) causes all the SConscript files to be called.

Note for SWIG: we can ultimately eliminate the *.in files

Layout of SConscript Files using the facilities package as an example

Python imports:
import os, glob
Import('baseEnv') **Note the big "I" for a SCons import. Also note that the objects are writable and changes will propogate to all.**
Import('registerObjects'

Navid is looking for a way to make the objects read-only, but for now, we make a copy of the baseEnv so that clients will not modify it and muddy it for others.

env = baseEnv.Copy() Jim suggests supplying a function instead, Navid is planning to do that.

Now to define the facilities static library, similiarly for a shared library.
facilitiesLib = env.StaticLibrary('facilities', library name
glob.glob(os.path.join('src', '*.cxx'))*** * glob.glob explands wildcards os.path.join inserts the appropriate slashes for each OS.
We can utilize subdirectories for the source location as well.

registerObjects('facilities', {'libraries':[facilitiesLib], 'includes':[glob.glob(os.path.join('facilities', '*.h'))

Each package contains a tool, <package>Lib.py which defines a python function according to SCons' specific form. The facilitiesLib.py file contains two functions:
generate which does the work, and *exists* which provides the ability to turn off the tool.

def generate(env, **kw): dictionary of keywords, which is not used now for ST where single libraries are created but GR will create packages where libraries may have different dependencies.

env.Tool('addLibrary', library=['facilities']) to call a tool addLibrary was created by Navid. This adds the facilities to a list of libraries.

site_tools contains addLibrary
ordering in gcc matters for libs - can't find symbols otherwise. The addLibrary tool allows developers to avoid worrying about the order themselves, since the tool reorders as necessary. addLibrary makes sure the order is correct, if the item is not already the list, add it to the end, if it is already there, move it to the end. Each library only appears once - this helps reduce the chance of creating a command line that is too long (which can occur when dealing with the Gaudi libraries).

Discussion

After building the libraries will exist in both the packages and in the installation lib directory. After the build is completed, all we need is the include and lib directories for runtime.
[Actually this was true at the time of the discussion, but since then Navid has backed off having the SCons build perform the installation step and will copy the necessary files to the installation directories as a separate step. Aug 20, 2007]

Jim reminded us that it would be helpful to provide a mechanism for tags such as rh9_gcc32 and rhel4, similarly for opt and debug builds. Navid could implement subdirectories with expanded English names as he's found some users are confused by our use of rh9_gcc32 for instance, something like redhat9_optimized.

Toby made a request that the new RM avoid "big bang builds" which take oodles of time. He suggested that packages common to both ST and GR could be pulled into their own checkout package. Navid stated that the SCons builds are much faster than the CMT builds.

Visual Studio 2005 - Toby is hoping Riccardo can fix up MRvcmt to use his python scripts to work with VS2005. Or perhaps MRvcmt is considered frozen - we need to discuss this with Riccardo as soon as possiblee.

Defining Applications using Likelihood as an example

For applications in the SConscript file:

```
import glob,os

Import('baseEnv')
Import('registerObjects')
env = baseEnv.Copy()
env.Tool('LikelihoodLib')          The applications can depend on the Likelihood library, LikelihoodLib.py sets up the environment to handle this.

LikelihoodLib = env.StaticLibrary('Likelihood', [glob.glob(os.path.join('src','.c')), glob.glob(os.path.join('src','.cxx'))])
gtlikelihoodBin = env.Program('gtlikelihood',glob.glob(os.path.join('src','likelihood','.cxx')))*
```

The contents of LikelihoodLib.py:

```
 def generate(env, **kw):
   env.Tool('addLibrary', library=['Likelihood'])
   env.Tool('astroLib')
   env.Tool('xmlBaseLib')
   env.Tool('tipLib')
   env.Tool('evtbinLib')                          The libraries can be added in any order, they will be sorted later
   env.Tool('map_toolsLib')
   env.Tool('optimizersLib')
   env.Tool('optimizersLib')
   env.Tool('irfLoaderLib')
   env.Tool('st_facilitiesLib')
   env.Tool('dataSubselectorLib')
   env.Tool('hoopsLib')
   env.Tool('st_appLib')
   env.Tool('st_graphLib')
   env.Tool('addLibrary', library=env['cfitsioLibs'])          Note how the external libraries are handled
   env.Tool('addLibrary', library=env['cppunitLibs'])
   env.Tool('addLibrary', library=env['fftwLibs'])
```

Navid also mentioned that ROOT libraries are setup similarly to how they are handled now with a standard set of RootLibs, and a set of GUI libs.

Issues

By default, SCons searchs for tools in the site_tools directory, or the user can provide a list of tool paths. This mechanism is working except in the case of a container package such as celestialSources. Navid is trying to avoid hard-coding anything into the SConscript files and external tools. The worst case scenario is that a container will have to announce itself as such.

Private includes should remain in the src directory as suggested currently. Actually, there were some public includes in some of the ST package, such as optimizers. Jim will look into that.

Navid is working on overriding package, aka multiple CMTPATHs.

We spoke a little about alternatives to the SCons' suggestion of automating the locating of dependencies. Jim had a method where a new environment was defined for each package. One could them import that environment and the appropriate dependencies. Some potential draw backs is that the ordering of dependencies does matter and there is a problem when working with GR and Gaudi in that the command line can get too long. Navid's method avoids those issues.

At runtime, after a build, SCons is no longer necessary.

We still need to address our use of environment variables such as <package>ROOT. Navid's commonUtilities class was created to allow pacakges to register their environment variables and perform the setenv and getenv calls. There will be a handful of remaining environment variables that must be defined a head of time, such as the location of the calibration files.

Navid's ultimate goal is to create the mechanism for SCons to build and generate files for the IDE of the developer's choice, including Visual Studio and KDE.

CVS Layout Discussion

A Request From Our Friends at the GSSC

Eric Winter, on behalf of the Science Support Center, asked about the future of the CMT requirements files. Currently hmake reads in the requirements files to set up its builds of the science tools. We plan to keep the CMT requirements file for about six months, as we continue to prepare for our move to SCons. We will have to provide some replacement, some SCons tool, for the GSSC to use instead. Currently, the SConscript and <package>Lib.py files contain the same information as the CMT requirements files. Brian Irby is man in charge of hmake.

To Do List

- Finish demonstration of SCons using ScienceTools, including over-riding packaegs / multiple CMTPATHs
- Windows and Visual Studio
- Use commonUtilities to replace our current use of environment variables
- Create ScienceTools-scons in CVS
- Create a new RM which is separate from our current one.
- Determine the schedule for MRStudio and what is necessary for the SCons-world.