CMT Status May 2007

From Core Minutes May 8, 2007

CMT v1r18p20061003 is now in use by RM. Navid and Joanne teamed up to find the problem, which was due to non-backwards-compatible handling of CMTPATH. However, this exercise has demonstrated that we can expect no support from CMT-central. It would be prudent to come up with an alternative which would allow us to dispense with CMT altogether. There are two candidates:

- SCons-based, perhaps in conjunction with Toby's Python scripts on Windows to produce suitable project files. Jim has volunteered to write a Python package would would take our CMT requirements files as input and create SCons SConstruct files, using a subset of ScienceTools to try it out.
- A roll-your-own solution from Navid. He has already done some work on it. He doesn't have sufficient time to put into it at the moment, but likely would in a few weeks.

The plan to eliminate unnecessary environment variables and define necessary ones at run time (not before) has been on hold but is being revived. (Toby) A large fraction of the CMT-generated ones, the <package>CONFIG variables, are unused and could be dispensed with immediately. [Post-meeting Navid has confirmed this. He tagged a newGlastPolicy which turns off the generation of these variables.]

Status from Jim Chiang May 17, 2007

Here is a rudimentary first cut at using python to parse the
requirements files to deploy SConstruct build files through a checkout
package directory tree:
/nfs/farm/g/glast/u33/jchiang/SConsBuilds
The script deploySCons.py finds and reads the requiements files and
writes "SConscript" files in the src subdirectories of each package.
I use the unix find command to discover the req files, but the python
function os.path.walk could also be used to enable portability to
windows.
The top-level SConstruct file is what one uses to do the builds. In
that file, we should put the equivalent of our various policy package
patterns as python functions and Export them so that subpackage
SConstruct files can Import them as needed.
You'll note that at the bottom of the SConstruct file there are the
lines

```
#SConscript(os.path.join(facilitiesDir, 'src', 'SConstruct'))
#SConscript(os.path.join(tipDir, 'src', 'SConstruct'))
#SConscript(os.path.join(astroDir, 'src', 'SConstruct'))
SConscript(os.path.join(facilitiesDir, 'src', 'SConscript'))
SConscript(os.path.join(tipDir, 'src', 'SConscript'))
SConscript(os.path.join(astroDir, 'src', 'SConscript'))
```

If you uncomment the first three of these and comment out the last
three, SCons will do a full build using the hand-crafted "SConstruct"
files I placed in each package, whereas using the lines as is will
give you a build using the deployed SConscript files and will only try
to build the libraries.
The astro package req file has a lot of cmt code in it that is not
used in any other package so I decided not to try to have
deploySCons.py attempt to parse and replicate all that logic in
writing its SConscript files. As a result, the astro library does not
build correctly using the Sconscript files.
To build a single package, one simply does
scons <package_name>
from the top-level directory, so we can control the order in which
things are built.
FYI, the perl scripts flattenSAE and cmt2hmake.pl are what Larry Brown
and James Peachey use to create an hmake file for ScienceTools.
-Jim

VRVS Meeting Summary May 23

SCons and Windows support
We should talk to Toby - but the plan would be to create real Visual Studio solution files that people (like Tracy) could really use. Jim mentioned SCons' Builders:
http://www.scons.org/doc/0.97/HTML/scons-user/c2338.html
which could be used to create our Visual Studio files, rather than using the default mechanism that SCons provides, which apparently doesn't produce very useful Visual Studio files.

Note: SCons 0.97 is now installed on the UW TS.

SCons will not require a package reorganization. [I'll stop asking 🙂]

Both Jim and Navid feel it would be best to avoid parsing our current requirements files, and rather move towards utilizing SCons or the syntax for the NPMS.

NPMS
With momentum building behind SCons should we continue to pursue the custom Navid build tool? Navid provided two remaining concerns about SCons:
1.) We don't know how well supported SCons will be by their development community.
Later Jim mentioned that SCons has a wider community than CMT and one would expect it would work out better than our CMT experience has.
2.) SCons does not terminate a build when a dependency fails - this will lead to problems for RM to report what package failed - we've discussed this in previous emails.

What about Windows?
Navid would provide something similar to SCons' Builder option - to allow users to define Visual Studio files.

Joanne feels we should carry on down the two paths a bit longer and Heather concurs.
Joanne requested that we provide a specification for the command line interface for either choice and Jim had queried Navid earlier concerning his command line interface. Navid responded that it will look a lot like SCons. Navid will try to have something we can try out next week...perhaps a package like facilities and also some package that has other package dependencies?

RM
Navid says any new build tool will require a new RM. He estimates about one month (sooner if a new GSFC hire materializes) to create a new RM. The current RM is patched up for various CMT work-arounds, it would be better to start fresh.

MRStudio
Need to talk to Riccardo - but Heather feels that we should not ask Riccardo to make any major modifications until we are sure what path we are taking - SCons or NPMS.

Update from Navid concerning NPMS May 29

My finger got better over the long weekend. I managed to get my thing
to compile facilities package from science tools. I have taken a few
shortcuts though and I'll be working on fixing those over time. But
technically my program can now be used to tell it to compile
facilities and it'll create the static library and the applications.
The shortcuts I've taken are:
1) When creating the .so and .a files it'll dump them into the
current directory at the moment.
2) When creating an app, instead of searching for the correct paths
to include for the .so and .a files it is right now hardcoded with
the current directory
3) Compile options are not tested. Theoretically they should work
(although somewhat limited) but I haven't tested it yet
4) Which brings up an odd problem when compiling Clock.cxx from
facilities, if you compile it without -g then ranlib complains that
Clock.o has no symbols. It doesn't seem to cause any errors. This
"error" will go away once I have compile options fully working.
I'll continue developing my version unless told otherwise =)
Navid

EVO Meeting Summary May 30th

Joanne Bogart, Toby Burnett, Jim Chiang, Navid Golpayegani, Riccardo Giannitrapani, Heather Kelly, Tracy Usher

VS2005 and SCons

Tracy has done some investigation into getting SCons to provide real Visual Studio solution files that would be useful for developers. He located where SCons makes its calls to its own Windows build scripts and suggests replacing them with Toby's Python scripts. This perhaps would only be done for developers, while general users would have no interest in Visual Studio project files anyway, and would use the standard SCons implementation.

Jim asked if anyone has looked into SCOns' Custom Builders and at this point, it seems no one has. It seems to move forward, there should be some further communication between Toby, Tracy, and Jim to see what they can work out for windows.

Tracy had asked via email if we are abadoning the CMT requirements files? Heather mentioned in the meeting that the answer is yes, ultimately.

Tracy also wonders about how SCons generates shared libraries on windows and how to incorporate gaudi component libraries and linker libraries. Toby has mentioned via email that he can handle shared libraries fine via his python scripts.

Toby brought up some things he needs to further understand concerning SCons:

- How to obtain a dependency tree from SCons?
- How to set link and cpp options
- A mechanism to find all packages given a set of paths

Jim responded that SCons does provide a way to get the dependencies. Searching paths is a CMT function that would not be provided by SCons.

NPMS

Navid reports that he has an initial version of his build tool which will compile the facilities package. There were some short cuts taken, outlined in the email posted above these minutes. He's working with the v8 version of SciTools, which only creates static libraries. It would be nice to provide a shared library example as well.

A version is available for testing, a binary is installed at SLAC: /nfs/farm/g/glast/u06/golpa/builderCompiled/builder/src/builder and a checkout of ST is located: /nfs/farm/g/glast/u06/golpa/ScienceTools-npms
but keep in mind this is in active development and there will be changes soon.

We then got into a discussion about Navid's primary concern about SCons which is its inability to build a single package so that the RM will know what package fails. If we go with SCons, the GLAST development community needs to be aware that RM may not be able to tell them precisely what package (s) failed to build. Navid was assuming he would have to parse the output to provide a best guess at the failure.
Jim provided an example, for the astro package which uses tip. So one would do something like:
scons tip
scons astro
Assuming the tip build failed, when on builds astro, scons will go back and try to rebuild tip and then link which will cause the astro build to fail.
CMT will only build astro, and provides the build order via cmt show uses.
Jim went on to explain that SCons can provide the dependency tree. We could separate the building of executables and linking into separate stages such that a build would only generate the object files. This should allow RM to determine the cause of the failure.

Some decision should be reached soon concerning whether we're interested in having Navid continue with NPMS or if we are suitably happy with SCons that we can all focus our attention in one direction. It is hoped that over the coming days, we can evaluate Navid's initial version of his build tool and get a sense if this is the best path or not.

MRStudio Upgrade

MRStudio was written to be much more flexible than MRvcmt, so removing CMT is certainly possible. We must retain the package structure of our code. MRStudio uses some CMT functionality:

- CVS for checkouts but uses CMT to get a list of packages for recursive checkouts.
- CMT provides a list of tags via "show tags"
- CMT is used to determine dependencies

Once we have chosen between SCons and NPMS, we should move to get MRStudio updated to provide a parallel path to CMT. Riccardo estimates it would take a few weeks to update MRStudio, assuming we can provide him with details for replacing CMT functionality listed above, as well as how to initiate builds.

Replacing CMT Functionality

Our discussion pointed out some areas where typical build tools will not replace some portion of CMT, such as the cmt checkout from CVS which puts our code into our typical package structure. Toby suggested that we could retain CMT just for those little bits. The general consensus seems to be that getting rid of CMT entirely would be the best goal. Some of those CMT routines are already done in other ways within the RM and the NPMS. We should pay some attention to what functions we lose when discarding CMT and make sure we provide suitable mechanisms to reproduce them.

Followup to May 30 EVO Meeting (Jim Chiang, May 31, 2007)

I've looked into a couple of the items that came up during the meeting:

Obtaining the interpackge dependencies

It is true that one can get the dependency tree from SCons, for example:

```
ki-rh2[jchiang] scons --tree=prune | more
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
scons: `facilities' is up to date.
+-facilities
  +-facilities/v2r12p5/rhel4_gcc34
  | +-facilities/v2r12p5/rhel4_gcc34/libfacilities.a
  | | +-facilities/v2r12p5/src/libfacilities.a
  | |   +-facilities/v2r12p5/src/ScheduledEvent.o
  | |   | +-facilities/v2r12p5/src/ScheduledEvent.cxx
  | |   | +-facilities/v2r12p5/facilities/ScheduledEvent.h
  | |   | +-facilities/v2r12p5/facilities/Scheduler.h
  | |   +-facilities/v2r12p5/src/Clock.o
  | |   | +-facilities/v2r12p5/src/Clock.cxx
  | |   +-facilities/v2r12p5/src/Scheduler.o
  | |   | +-facilities/v2r12p5/src/Scheduler.cxx
  | |   | +-[facilities/v2r12p5/facilities/ScheduledEvent.h]
  | |   | +-[facilities/v2r12p5/facilities/Scheduler.h]
  | |   +-facilities/v2r12p5/src/Timestamp.o
  | |   | +-facilities/v2r12p5/src/Timestamp.cxx
  | |   | +-facilities/v2r12p5/facilities/Timestamp.h
  | |   +-facilities/v2r12p5/src/Util.o
  | |   | +-facilities/v2r12p5/src/Util.cxx
  | |   | +-facilities/v2r12p5/facilities/Util.h
  | |   +-facilities/v2r12p5/src/ScheduledEvent.cxx.~1.1.1.1.~
  | +-facilities/v2r12p5/rhel4_gcc34/testUtil
  | | +-facilities/v2r12p5/src/test/testUtil
  | |   +-facilities/v2r12p5/src/test/testUtil.o
  | |   | +-facilities/v2r12p5/src/test/testUtil.cxx
  | |   | +-[facilities/v2r12p5/facilities/Util.h]
  | |   +-[facilities/v2r12p5/src/libfacilities.a]
  | +-facilities/v2r12p5/rhel4_gcc34/test_time
  |   +-facilities/v2r12p5/src/test/test_time
  |     +-facilities/v2r12p5/src/test/test_time.o
  |     | +-facilities/v2r12p5/src/test/test_time.cxx
  |     | +-[facilities/v2r12p5/facilities/Timestamp.h]
  |     +-[facilities/v2r12p5/src/libfacilities.a]
  +-[facilities/v2r12p5/rhel4_gcc34/libfacilities.a]
  +-[facilities/v2r12p5/rhel4_gcc34/testUtil]
  +-[facilities/v2r12p5/rhel4_gcc34/test_time]
scons: `tip' is up to date.
+-tip
  +-tip/v2r12/rhel4_gcc34
```

However, as you can see this gives the dependencies at the **file** level, whereas we would need it at the package level, a much less detailed set of output. The above would be parseable and probably will suffice, but with SCons we will need to put the package dependencies in the SConscript files in the package src subdirectories anyways, so it may make more sense to parse those directly. Here, e.g., it is for astro:

astro/src/SConscript

```
#-*- python -*-
Import('packageSetup', 'facilities', 'cfitsio', 'CLHEP', 'tip', 'ROOT')
env=Environment(CPPPATH=[], LIBPATH=[], LIBS=[])
for item in ('CPPPATH', 'LIBPATH', 'LIBS'):
    env[item].extend(facilities[item])
    env[item].extend(cfitsio[item])
    env[item].extend(CLHEP[item])
    env[item].extend(tip[item])
    env[item].extend(ROOT[item])
packageSetup('astro', env)
```

It should be a simple matter to build a dependency tree from the entries in the Import command.

SCons going off and trying to build libraries from an upstream package if that package build has already failed.

I think this is a non-issue. I have looked into writing an SConstruct file that will stop the build of the application if a library it needs does not already exist. Unfortunately, because SCons reads and processes all of the SConscript files **before** trying to build anything, the best I could do was disable the building of the application entirely, such that no build failure is registered as it would be if there were a simply a missing library. Maybe this is ok, but it is not how the present RM works.

However, in the case where we allow SCons to work as designed, the extra overhead in having it go off to try to build the errant library (or libraries) is trivial since the build will fail at exactly the same point as it did the first time and furthermore will not propagate to the other libraries that it will want to build downstream.

In the following example, I introduced bugs in the facilities and tip source code so that both packges fail to build their libraries, then I issued the command to build astro after already having done so for both tip and facilities:

```
ki-rh2[jchiang] scons astro
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o astro/v2r9/src/healpix/healpix_base.o -c -g -Ifacilities/v2r12p5 -I/afs/slac/g/glast/ground/GLAST_EXT
/rhel4_gcc34/cfitsio/v3006/include \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/CLHEP/1.9.2.2/include -Itip/v2r12 \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/ROOT/v5.10.00/root/include -Iastro/v2r9 -Iastro/v2r9/src
/wcslib -Iastro/v2r9/src \
-Iastro/v2r9/src/jplephem -Iastro/v2r9/src/healpix astro/v2r9/src/healpix/healpix_base.cc
g++ -o astro/v2r9/src/healpix/cxxutils.o -c -g -Ifacilities/v2r12p5 \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/cfitsio/v3006/include \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/CLHEP/1.9.2.2/include -Itip/v2r12 \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/ROOT/v5.10.00/root/include -Iastro/v2r9 -Iastro/v2r9/src
/wcslib -Iastro/v2r9/src \
-Iastro/v2r9/src/jplephem -Iastro/v2r9/src/healpix astro/v2r9/src/healpix/cxxutils.cc

<...skip compiling of astro source code...>

ranlib astro/v2r9/src/libastro.a
Install file: "astro/v2r9/src/libastro.a" as "astro/v2r9/rhel4_gcc34/libastro.a"
g++ -o astro/v2r9/src/test/test.o -c -g -Ifacilities/v2r12p5 -I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34
/cfitsio/v3006/include \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/CLHEP/1.9.2.2/include -Itip/v2r12 \
-I/afs/slac/g/glast/ground/GLAST_EXT/rhel4_gcc34/ROOT/v5.10.00/root/include -Iastro/v2r9 -Iastro/v2r9/src
/wcslib -Iastro/v2r9/src \
-Iastro/v2r9/src/jplephem -Iastro/v2r9/src/healpix astro/v2r9/src/test/test.cxx
g++ -o facilities/v2r12p5/src/ScheduledEvent.o -c -g -Ifacilities/v2r12p5 -Ifacilities/v2r12p5/src \
facilities/v2r12p5/src/ScheduledEvent.cxx
facilities/v2r12p5/src/ScheduledEvent.cxx: In static member function `static void ScheduledEvent::schedule
(double, ScheduledEvent*)':
facilities/v2r12p5/src/ScheduledEvent.cxx:10: error: `nehxt' was not declared in this scope
scons: *** [facilities/v2r12p5/src/ScheduledEvent.o] Error 1
scons: building terminated because of errors.
ki-rh2[jchiang]
```

So it just stops as soon as it encounters the error in facilities and further does not recompile any other facilities source code. This doesn't seem like a huge extra overhead to incur and will behave almost exactly like the present RM does. I can't think of any other cases where there will be any significant burden imposed by this behavior of SCons.