SCons A Demo Implementation using SciTools

Introduction

Navid has been busy setting up a demonstration environment using SCons and the ScienceTools v8r2.

Location of Installation

⚠ A new version of SCons has been installed since the previous version had a bug with swig and variant builds.

SCons is installed here at SLAC: `/afs/slac/g/glast/applications/SCons/0.97.0d20070809/bin/scons`
There's a copy of SciTools in Navid's area: `/nfs/farm/g/glast/u06/golpa/ST-v8r2-scons/ScienceTools`
If you desire to run SCons, please read below on how to check it out from the CVS copy.

CVS

The CVS copy is located in `/nfs/farm/g/glast/u33/golpa/cvs`. To check out the code set your CVSROOT environment variable to point to that location. Once that is done you can check out a v8r2 version of ScienceTools by issuing the command `cvs co -r ScienceTools-v8r2 ScienceTools-scons`.

To allow CVS to do such checkouts while staying compatible with CMT, the structure of CVS has been changed as follows:

```
CVSROOT
  |
  +--ScienceTools/
  |
  +--astro/
  |    |
  |    +--cmt/
  |    |
  |    +--src/
  |    |
  |    +--...
  +--...
  |
  +--ScienceTools-scons
       |
       +--astro/ --> Symlink back to above package
       |
       +--Likelihood/ --> Symlink back to above package
       |
       +--...
       |
       +--SConsctruct
       |
       +--externals.scons
       |
       +--site_scons/
            |
            +--site_tools/
                 |
                 +--addLibrary.py
                 |
                 +--registerObjects.py
```

Basic SCons commands

- To see a list of available command line options type `/afs/slac/g/glast/applications/SCons/0.97.0d20070809/bin/scons -h`.
- To do a build of all the libraries and the main applications type `/afs/slac/g/glast/applications/SCons/0.97.0d20070809/bin/scons with-GLAST-EXT=/path/to/GLAST_EXT`.
- To build only a specific package (and all the libraries it depends on) type the previous command adding the name of the package(s) to the end.
- To build everything above and test applications type the previous command but using the package name `all`.

Interesting Details and things we may need to discuss

There is a top-level SConstruct file which contains configuration/build information to be used "globally". This includes options for the compilation such as debug and optimization flags and contains code to find the SConscript files part of this build. See the Gory Details of the SConstruct file for more information. There is another file at the top-level called externals.scons which is used to set up the location of the external libraries. More on this point further down in the External Libraries section.

Location of SCons specific files and package structure

Each package contains its own SConscript file which act as SCons' version of CMT's requirements file. You can see an example for the facilities package further down on this page. The SConstruct file communicates the location of the SConscript file for each package.

The virjpk subdirectories are not in place or assumed in this organization. There is a note out to Riccardo to check if that arrangement is acceptable to MRStudio. Doing without the package-tag subdirectories allows us to utilize CVS directly for our checkouts, rather than devising a way to recreate CMT checkouts.

External Libraries

Code in the SConstruct file provides a mechanism similar to automake that allows users to specify the location of external libraries and their headers using `with-LIBRARY`, `with-LIBRARY-lib` and `with-LIBRARY-include`.

There is also an option to specify GLAST_EXT style directory layout by specifying the command line option `with-GLAST-EXT`. Any of the libraries found in the GLAST_EXT style layout can still be overridden with the `with-LIBRARY[-lib | -include]` options.

The code that provides the above functionality is located in `externals.scons`. It is responsible for locating the third party libraries and adding appropriate -L options to the compile environment. It also adds to the compile environment a list of libraries that should be linked against when a certain third party library is required.

Variant Builds

To allow for building the same code with different compile options, I have enabled SCons variant builds. What this means is that when a compile is performed, SCons will create a "virtual" copy of the code in a subdirectory. It will perform all builds in that directory. The directory name that is used can be modified by specifying the `variant=` option at the command line. If this option is not specified, SCons will determine the the OS it is running on and use that as the variant name. Additionally, it will determine if debug and/or optimized is enabled. If that is the case, it will add a `-debug` and/or `-optimized` to the variant name. Once the variant name is determined, builds are performed in `[packageName]/build/[variant]`. Once compiled in there, the libraries and other platform dependent data is installed in subdirectories that contain this variant name. For example if a build is done without specifying the `variant=` option on a Linux OS then the astro package would be built in `astro/build/Linux`. Once everything is compiled, the libraries would be installed in `lib/Linux/`, and the binaries would be in `bin/Linux`. Header files, on the other hand, would be located in `include/`.

Overriding packages

There are two ways two override packages. The first method you are doing a full build in your own environment with multiple versions of the same package. In the second method you are building only a few packages (again with potentially multiple versions) against an already built build version of the application.

1. Doing a full build with multiple versions of the packages.
   With this method there's nothing "special" you need to do. As long as your packages have the format packageName-distinctionString, then SCons will pick up only one version of this package. For example let's say you wish to compile ST-v8r2 with multiple versions of astro. You would check out ST-v8r2 as before which will check out the standard version of astro. Now you wish to compile a different version of astro. You simply check out the version of astro into a directory with slightly modified name that follows the convention above. For example you check it out as astro-1. When SCons sees this package it automatically knows that it's a different version of astro and compiles astro-1 instead of astro. If you have multiple several versions (for example astro, astro-1, astro-2,etc.), then SCons will pick the last one in lexical order (in this case astro-2).
2. Building a partial set of packages against a full build
   In this case you run SCons with the argument override=somePath where somePath is the **full path** to the override directory. For example let's say the full build is located in /ST-v8r2 and you have overridden astro in ~/ST-v8r2-override then, **while located in /ST-v8r2**, you execute scons with-GLAST-EXT=/path/to/glast-ext override=~/ST-v8r2-override. This will build the new astro in ~/ST-v8r2-override against other libraries located in /ST-v8r2. Of course ~/ST-v8r2-override can contain multiple packages including multiple versions of the same package. In the latter case the same rules as the previous option apply.
   In the likely event you don't wish to be located in /ST-v8r2 when compiling the override packages you can also do this build by adding an additional argument to scons. This argument is -C /ST-v8r2. This will tell SCons to change dir to /ST-v8r2 before executing and return back to your current dir after execution.

Gory Details of the SConstruct file

The SConstruct filfe begins by setting up some global variables. It then goes on and sets up the basic compile environment used by all packages to compile code.

```
import os,platform,SCons,glob,re
platform = platform.system()
prefix = '/usr/local/bin'
override = '.'
########################
#   Global Environment  #
########################
baseEnv=Environment()
if ARGUMENTS.get('debug',0):
        baseEnv.Append(CCFLAGS = "-g")
        platform+="-Debug"
if ARGUMENTS.get('optimized',0):
        baseEnv.Append(CCFLAGS = "-O2")
        platform+="-Optimized"
if ARGUMENTS.get('CC',0):
        baseEnv.Replace(CC = ARGUMENTS.get('CC'))
if ARGUMENTS.get('CXX',0):
        baseEnv.Replace(CXX = ARGUMENTS.get('CXX'))
if ARGUMENTS.get('CCFLAGS',0):
        baseEnv.Append(CCFLAGS = ARGUMENTS.get('CCFLAGS'))
if ARGUMENTS.get('CXXFLAGS',0):
        baseEnv.Append(CXXFLAGS = ARGUMENTS.get('CXXFLAGS'))
if ARGUMENTS.get('variant',0):
        platform = ARGUMENTS.get('Variant')
if ARGUMENTS.get('prefix',0):
        prefix=ARGUMENTS.get('prefix')
if ARGUMENTS.get('override',0):
        override=ARGUMENTS.get('override')
        SConsignFile(os.path.join(override,'.sconsign.dblite'))
```

It then continues on to set up some help strings that will be displayed by SCons to the user when requesting help on compiling.

```
helpString = """
Usage:
        scons [target] [compile options]
Targets:
        Default:                Build release binaries and libraries
        test:                   Build test binaries and required libraries
        binaries:               Build release binaries and required libraries
        libraries:              Build all libraries
        install:                Install release binaries, includes, and libraries
        isntall-test:           Install test binaries

Compile Options:
        optimized:              Set to 1 to compile with optimization. Default 0.
        debug:                  Set to 1 to compile with debug. Default 0.
        CC:                     Set to the compiler to use for C source code.
        CXX:                    Set to the compiler to use for C++ source code.
        CCFLAGS:                Set to additional flags passed to the C compiler.
        CXXFLAGS:               Set to additional flags passed to the C++ compiler.
        variant:                Compile binaries into subdirectory pointed by this varia$
        override:               Directory where overridden packages are located. Relativ$
"""
Export('baseEnv')
```

After that, the project environment is updated to include the location where libraries/binaries/includes/etc. will be both after compiling and later when asked to officially install the code into its final location (aka make install):

```
########################
#  Project Environment  #
########################
baseEnv.Append(LIBDIR = os.path.join(os.path.abspath(override),'lib',platform))
baseEnv.Append(BINDIR = os.path.join(os.path.abspath(override),'bin',platform))
baseEnv.Append(INCDIR = os.path.join(os.path.abspath(override),'include'))
baseEnv.Append(PFILESDIR = os.path.join(os.path.abspath(override),'pfiles'))
baseEnv.Append(TESTDIR = baseEnv['BINDIR'])
baseEnv.Append(INSTBIN = os.path.join(prefix,'bin',platform))
baseEnv.Append(INSTLIB = os.path.join(prefix,'lib',platform))
baseEnv.Append(INSTINC = os.path.join(prefix,'include'))
baseEnv.Append(INSTPFILES = os.path.join(prefix,'pfiles'))
baseEnv.Append(INSTTEST = baseEnv['TESTDIR'])
baseEnv.Append(CPPPATH = ['.'])
baseEnv.Append(CPPPATH = ['src'])
baseEnv.Append(CPPPATH = [baseEnv['INCDIR']])
baseEnv.Append(LIBPATH = [baseEnv['LIBDIR']])
baseEnv.AppendUnique(CPPPATH = os.path.join(os.path.abspath('.'),'include'))
baseEnv.AppendUnique(LIBPATH = os.path.join(os.path.abspath('.'),'lib',platform))
```

Next, it sets up the external libraries and adds the external libraries command line options to the help string displayed to users and register that help string with SCons.

```
########################
#  External Libraries   #
########################
helpString += SConscript('externals.scons')

helpString += """
Install Options:
        prefix: Location to install files ($PREFIX/bin, $PREFIX/include, etc.). Default: /usr/local
"""

Help(helpString)
```

The SConstruct file then defines a function responsible for listing files in the "virtual" directory for the variant build.

```
def listFiles(files):
        allFiles = []
        for file in files:
                if file.find('*') == -1:
                        allFiles.append(file)
                else:
                        newFiles = glob.glob(os.path.join(str(Dir('.').srcnode()),file))
                        for newFile in newFiles:
                                newFile = str(Dir('.').srcnode().rel_path(File(newFile)))
                                allFiles.append(newFile)
        return allFiles

Export('listFiles')
```

A check is then performed to see if the target being performed is the install target. If that's the case, additional files are registered to be installed

```
if 'install' in COMMAND_LINE_TARGETS:
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                         baseEnv.Install(prefix, 'SConstruct'))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                         baseEnv.Install(prefix, 'externals.scons'))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                         baseEnv.Install(os.path.join(prefix, 'site_scons', 'site_tools'),
                        os.path.join('site_scons','site_tools', 'registerObjects.py')))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                        baseEnv.Install(os.path.join(prefix, 'site_scons', 'site_tools'),
                        os.path.join('site_scons','site_tools', 'addLibrary.py')))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                        baseEnv.Install(os.path.join(prefix,'bin',platform),
                        glob.glob(os.path.join(baseEnv['BINDIR'],'*'))))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                        baseEnv.Install(os.path.join(prefix,'lib',platform),
                        glob.glob(os.path.join(baseEnv['LIBDIR'],'*'))))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                        baseEnv.Install(os.path.join(prefix,'include'),
                        glob.glob(os.path.join(baseEnv['INCDIR'], '*'))))))
        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                        baseEnv.Install(os.path.join(prefix,'pfiles'),
                        glob.glob(os.path.join(baseEnv['PFILESDIR'], '*'))))))
```

Towards the end, a the SConstruct file attempts to locate all SConscript files and all tools. It regsiteres all the tools with SCons and stores the location of the SConscript files in a list. Additional logic checks to see if a package is listed twice. If that's the case the package that is lexically last is only added. The other versions of the package are ignored.

```
directories = [override]
packages = []
# Add packages to package list and add packages to tool path if they have one
while len(directories)>0:
    directory = directories.pop(0)
    listed = os.listdir(directory)
    listed.sort()
    pruned = []
    #Remove duplicate packages
    while len(listed)>0:
        curDir = listed.pop(0)
        package = re.compile('-.*$').sub('', curDir)
        while len(listed)>0 and re.match(package+'-.*', listed[0]):
            curDir = listed.pop(0)
        pruned.append(curDir)
    #Check if they contain SConscript and tools
    for name in pruned:
        package = re.compile('-.*$').sub('',name)
        if name != 'build':
            fullpath = os.path.join(directory,name)
            if os.path.isdir(fullpath):
                directories.append(fullpath)
                if os.path.isfile(os.path.join(fullpath,"SConscript")):
                    packages.append(fullpath)
                if os.path.isfile(os.path.join(fullpath, package+'Lib.py')):
                    SCons.Tool.DefaultToolpath.append(os.path.abspath(fullpath))
                    if 'install' in COMMAND_LINE_TARGETS:
                        baseEnv.NoClean(baseEnv.Default(baseEnv.Alias('install',
                                baseEnv.Install(os.path.join(prefix,'site_scons','site_tools'),
                                os.path.join(fullpath,package+'Lib.py')))))
```

Finally, if the target being build is not the install target, SCons is told to load all SConscript files one at a time and control is handed over to SCons to start the build process.

```
if 'install' not in COMMAND_LINE_TARGETS:
        for pkg in packages:
                baseEnv.SConscript(os.path.join(pkg,"SConscript"),
                build_dir = os.path.join(pkg, 'build', platform))
```

An Example SConscript File

This is an example of the structure of a SConscript file. This example is from the fitsGen package. First some variables and objects are imported into the SConscript file. The `import` command is a python command while the `Import()` command is an SCons command importing variables exported previously by an `Export()` command.

```
import glob,os,platform

Import('baseEnv')
Import('listFiles')
```

Next the basic environment is cloned for local use

```
progEnv = baseEnv.Clone()
libEnv = baseEnv.Clone()
```

It is important that the environment is cloned before changes are made to it. If that is not done, any changes will propagate to all other packages.

Optionally, if there are changes that need to be made to the environment the can be done. In this case a CPP define is added to the compile command if the OS is a Linux system

```
if platform.system() == 'Linux':
        progEnv.Append(CPPDEFINES = 'TRAP_FPE')
```

Now, a static library with the name `fitsGen` is created that consists of the files in `src/*.cxx`.

The library is created in the `libEnv` environment. This is because the `progEnv` environment adds additional link commands to linker that are not needed for a library and will in fact cause circular dependency if specified.

```
fitsGenLib = libEnv.StaticLibrary('fitsGen', listFiles(['src/*.cxx']))
```

After telling SCons to create a library, we specify that `progEnv` environment should add the fitsGen library to be linked in when linking the applications. All libraries the fitsGen library depends on are included as well.

This time we use the progEnv environment so that the previous environment does not link against fitsGen and its dependencies

```
progEnv.Tool('fitsGenLib')
makeFT1Bin = progEnv.Program('makeFT1', 'src/makeFT1/makeFT1.cxx')
makeFT2Bin = progEnv.Program('makeFT2', 'src/makeFT2/makeFT2.cxx')
makeFT2aBin = progEnv.Program('makeFT2a', 'src/makeFT2a/makeFT2a.cxx')
egret2FT1Bin = progEnv.Program('egret2FT1', listFiles(['src/egret2FT1/*.cxx']))
convertFT1Bin = progEnv.Program('convertFT1', 'src/convertFT1/convertFT1.cxx')
partitionBin = progEnv.Program('partition', 'src/partition/partition.cxx')
irfTupleBin = progEnv.Program('irfTuple', listFiles(['src/irfTuple/*.cxx']))
```

Finally, we register all these objects so that they can be assigned to the correct targets to be built and that their final locations are computed.

```
progEnv.Tool('registerObjects', package = 'fitsGen', libraries = [fitsGenLib],
             binaries = [makeFT1Bin, makeFT2Bin, makeFT2aBin, egret2FT1Bin,
             convertFT1Bin, partitionBin, irfTupleBin],
             includes = listFiles(['fitsGen/*.h']),
             pfiles = listFiles(['pfiles/*.par']))
```

Dependency Computation

SCons uses something called tools. Tools are nothing more than python functions that modify the compile environment. This feature was recommended by SCons developers to use for calculating dependencies recursively. To this this a package creates a file that is `[packageName]Lib.py`. This file contains two functions `generate()` and `exists()`. The `generate()` function does the actual modifications of the environment while the `exists()` function is to give an option to disable the tool under certain conditions. Here's an example from the Likelihood package.

The LikelihoodLib.py file first defines the `generate()` function. This function uses the addLibrary tool to add its own libraries to the link command. It then calls the tool for other packages it **directly** depends on. It also calls the `addLibrary` tool to register external libraries it depends on.

It is important that only libraries that are needed for the library being added are listed here. For example liblikelihood.[so|a] directly depends on astro lib. The astro library should therefore be called. Likelihood test applications might also depend on cppunit external library but they should **n ot** be listed here since they are not needed for linking against liblikelihood.[so|a].

```
def generate(env, **kw):
    env.Tool('addLibrary', library=['Likelihood'], package = 'Likelihood')
    env.Tool('astroLib')
    env.Tool('xmlBaseLib')
    env.Tool('tipLib')
    env.Tool('evtbinLib')
    env.Tool('map_toolsLib')
    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'])
    env.Tool('addLibrary', library=env['fftwLibs'])
```

Finally, the `exists()` function is created which does nothing more than return a true value to notify SCons that this tool is enabled.

```
def exists(env):
    return 1
```

Interesting Python and SCons filesystem manipulation routines

os.path.join joins 'src' and '*.cxx' together in the correct way ie \ for windows and / for linux
glob.glob then takes the src/*.cxx and gets a list of files that match that pattern
I have added a listFiles() function that'll list the files in the "virtual" environment that SCons creates.

Environment variables

Our use of environment variables set by CMT needs to be removed. All use of [PACKAGENAME]ROOT environment variables is being removed in favor of using commonUtilities functions such as getDataPath(). Some other variables are, however, being used in parts of the code. These need to be addressed separately.

Name	Status	Solution	Used In
CALDB	Needed	Set at runtime using commonUtilities::setEnvironment()	irfs/caldb, irfs/dc1Response, irfs/dc1aResponse, irfs /irfUtil
CALDBCONFIG	Needed	Set at runtime using commonUtilities::setEnvironment()	irfs/caldb, irfs/irfUtil
CALDBALIAS	Needed	Set at runtime using commonUtilities::setEnvironment()	irfs/caldb
PULSAROUTFILES	Not Needed	Obtain log directory using commonUtilities::getLogPath()	celestialSources/Pulsar
SKYMODEL_DIR	Not Needed	Remove use of this variable	celestialSources/Pulsar, Gleam
PULSARDATA	Not Needed	Use getDataPath("Pulsar") instead	celestialSources/Pulsar, Gleam
BYPASS_ACCUMULATOR	Unknown		Likelihood
USE_OLD_LOGLIKE	Unknown		Likelihood
HANDOFF_IRF_NAME	Unknown		irfs/handoff_response
INTERPOLATE_EDISP	Unknown		irfs/handoff_response
CALIB_DIR	Unknown		irfs/irfLoader
OPTIMIZERSROOT	Unknown	Leave as is. It's written out to an xml file	optimizers

TIMING_DIR	Unknwon	Replace with EXTFILESSYS	timeSystem
EXTFILESSYS	Needed	Set at runtime using setupEnvironment()	various packages
MERIT_INPUT_FILE	Unknown	Unknown	GlastClassify, merit
MERIT_OUTPUT_FILE	Unknown	Unknown	GlastClassify, merit
CTREE_PATH	Unknown	Unknown	GlastClassify
PRUNEROWS	Unknown	Unknown	GlastClassify
Fred_DIR	Not Needed	Use $GLAST_EXT/Fred/<version>	Gleam
SKYMODEL_DIR	Unknown	Unknown	Gleam
POINTING_HISTORY_FILE	Unknown	Unknown	Gleam
MOOT_ARCHIVE	Unknown	Unknown	MootSvc, configData
OBFXFCBINDIR	Unknown	Unknown	OnboardFilter
OBFXFC_DBBINDIR	Unknown	Unknown	OnboardFilter
OBFEFCBINDIR	Unknown	Unknown	OnboardFilter
OBFGFC_DBBINDIR	Unknown	Unknown	OnboardFilter
OBFCPP_DBBINDIR	Unknown	Unknown	OnboardFilter
OBFCPG_DBBINDIR	Unknown	Unknown	OnboardFilter
OBFGGF_DBBINDIR	Unknown	Unknown	OnboardFilter
MYSQL_METATABLE	Unknown	Unknown	calibUtil
MYSQL_HOST	Unknown	Unknown	calibUtil
USER	Unknown	Unknown	calibUtil, rdbModel
TESTJOBOPTIONS	Unknown	Unknown	gr_app
GLEAM_CHDIR	Unknown	Unknown	gr_app
JOBOPTIONS	Unknown	Unknown	gr_app
stdout	Unknown	Unknown	gui
CTREE_PATH	Unknown	Unknown	merit
MOOT_ARCHIVE	Unknown	Unknown	mootCore
USERNAME	Unknown	Unknown	rdbModel
mycalibs	Unknown	Unknown	rdbModel