SCons progress

- deploySCons.py creates an SConscript file in src subdirectory of each package based on use statements in requirements file:

tip/cmt/requiremets

```
package tip

version v1r1p1

author James Peachey <peachey@lheamail.gsfc.nasa.gov>
# $Id: requirements,v 1.11 2006/02/01 18:38:35 peachey Exp $

use STpolicy *
use ROOT * IExternal
use cfitsio * IExternal

macro_append tip_linkopts "" Linux "-lHist " Darwin "-lHist "
apply_pattern ST_library
apply_pattern shared_st_library
application sample sample/sample.cxx
```

tip/src/SConscript

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

- The equivalent of our policy package patterns should be implemented as python functions in the top-level SConstruct file:

SConstruct

```
#-*- python -*-
"""
@brief Top-level SConstruct file for example checkout package build.

@author J. Chiang <jchiang@slac.stanford.edu>
"""
#
# $Header: /nfs/slac/g/glast/ground/cvs/users/jchiang/DeploySCons/SConstruct,v 1.7 2007/05/22 01:08:06 jchiang
Exp $
#
import os, glob

baseDir = os.path.abspath(os.curdir)

if os.name == 'nt':
    bindir = 'VC8debug'
    env.Append(CCFLAGS=['/EHsc'])
    env['ENV']['TMP'] = os.environ['TMP']
elif os.name == 'posix':
    bindir = 'rh9_gcc32'
#    bindir = 'rhel4_gcc34'

def srcFiles(path):
    query = '/usr/bin/find %s -name \*.c\* -print' % path
    files = [item.strip() for item in os.popen(query)
             if item.find('./test') == -1]
```

```
    query = '/usr/bin/find %s -name \*.h -print' % path
    hfiles = [item.strip() for item in os.popen(query)
              if item.find('./test') == -1]
    cpppaths = {}
    for item in hfiles:
        cpppaths[os.path.abspath(os.path.split(item)[0])] = 1
    return files, cpppaths.keys()

def packageSetup(package, env):
    packagedir = os.path.abspath('..')
    src_files, paths = srcFiles(os.curdir)
    cpppath = [packagedir]
    cpppath.extend(paths)
    env.Append(CPPPATH=cpppath, CCFLAGS=['-g'])
    library = env.StaticLibrary(package, src_files)
    installdir = os.path.join(packagedir, bindir)
    env.Install(installdir, library)
    env.Alias(package, installdir)
    env.Default(package)
    Export('package', 'packagedir', 'env', 'installdir')
    SConscript(os.path.join('test', 'SConstruct'))

def extlibEnv(package, version, libs=None):
    packageDir = os.path.join(os.environ['GLAST_EXT'], package, version)
    if libs is None:
        libs = [package]
    return Environment(CPPPATH=[os.path.join(packageDir, 'include')],
                       LIBPATH=[os.path.join(packageDir, 'lib')],
                       LIBS=libs)

CLHEP = extlibEnv('CLHEP', '1.9.2.2')
cfitsio = extlibEnv('cfitsio', 'v3006')
ROOT = extlibEnv('ROOT', os.path.join('v5.10.00', 'root'),
                 libs=['Core', 'Cint', 'Tree', 'Matrix', 'Physics',
                       'Graf', 'Graf3d', 'Gpad', 'Rint', 'Postscript',
                       'TreePlayer', 'Hist', 'pthread', 'm', 'dl'])
fftw = extlibEnv('fftw', '3.0.1')
pil = extlibEnv('pil', '2.0.1')
xerces = extlibEnv('xerces', '2.6.0')
cppunit = extlibEnv('cppunit', '1.10.2')

exports = ['bindir', 'packageSetup', 'cfitsio', 'CLHEP', 'ROOT',
           'fftw', 'pil', 'xerces', 'cppunit']

def packageEnv(package, version):
    packageDir = os.path.join(baseDir, package, version)
    return Environment(CPPPATH=[os.path.join(packageDir)],
                       LIBPATH=[os.path.join(packageDir, bindir)],
                       LIBS=[package]), packageDir

packages = {'facilities' : 'v2r12p5',
            'tip' : 'v2r12',
            'astro' : 'v2r9',
            'st_stream' : 'v0r5',
            'hoops' : 'v0r4p5',
            'st_graph' : 'v1r7',
            'st_app' : 'v2'}

for package in packages:
    exec('%s, %sDir = packageEnv("%s", "%s")' % (package, package,
                                                 package, packages[package]))
    exports.append(package)

Export(*exports)

for package in packages:
    exec('SConscript(os.path.join(%sDir, "src", "SConscript"))' % package)
```

- In the top level SConstruct file, source code is found by looking for all filename patterns *.c* and directories to be added to CPPPATH (for includes) are found by matching *.h. Since more than one .cxx file in a given package may contain a main() function, this scheme will fail once

one tries to link. It would be better if all source code for the library lived in the src directory. The packages astro and irfs/handoff_response violate this convention.

- So far, the implementation will build only libraries. When the building of applications is implemented, it would be desireable to have each req file contain use statements for every library it needs to link against. This is preferable to relying on implicit uses inherited from other packages.
- In just the three lowest level packages facilities, tip, and astro there are **three** different ways to specify application builds. It would be better to have a single convention, with ancillary application-specific source code living in the application subdirectory.
- It would help if test programs had a standard name, test_<package>.cxx, and there was only one that lives in the src/test subdir.
- At the end of the day, it's probably just easier to write SConscript files for each package by hand instead of parsing the requirments files and deploying them automatically. Compare astro's requirements file and the SCons equivalent.