Modular Software Building with Python SCons

SEA Conference February 21, 2012 Gary Granger NCAR, Earth Observing Laboratory





Build System Goals

- Modular
 - Change how a module is built without changing the builds which depend on it
- Portable
 - One build system for multiple platforms which runs from IDEs and CI tools.
- Extensible
 - Build more than programs
- Configurable
 - Let developer define and configure build options

SCons Key Points

- Definition and procedure in one powerful scripting language: python
- Build configuration divided into modular tools, including C, C++, Java, FORTRAN...
- One complete dependency tree assembled from build scripts in sub-trees
- Cross-platform: tool scripts can be portable across OS's because python is portable

SConscript Example

env = Environment(tools = ['default', 'log4cpp'])

sources = Split("""Logging.cc ...""")
objects = env.Object(sources)

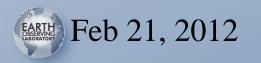
lib = env.Library('logx', objects)

env.Default(lib)



SConscript Basics

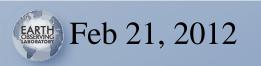
- Environment: Construction variables, methods, context
- Builders: Run commands to generate TARGETS from SOURCES
- Tools: Extend the Environment with new builders and modify construction variables
- Virtual Filesystem: All nodes have a path even before they exist



SCons Build Phases

SCons does not execute the SConscript to build the targets:

- 1. Read all of the SConscript files and execute them to build the dependency tree and configure the builders.
- 2. Run the build engine to analyze dependencies and update the default or explicit targets.



SCons Distinctions

- Strict Environment
- Careful and thorough dependencies
 - scanners for implicit dependencies
 - implicit executables
 - checksums and not just timestamps
- Developer-defined build Variables
- Autoconf-like compiler and linker checks
- Parser for pkg-config and similar scripts
- Parallel builds

Feb 21, 2012

• Source code control interfaces

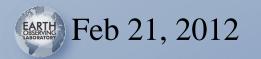
EOL SCons

- eol_scons package loaded automatically by site_scons in top level directory
- Custom tools
- Module tools within the source tree
- Wrapper Environment methods
- Build Variables
- Global Target References
- Optimizations

Feb 21, 2012

boost_date_time.py

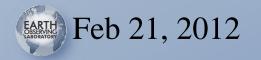
def generate(env):



tool_logx.py

def logx(env):

Export('logx')



Source Tool Example

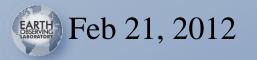
aeros/

SConstruct:

env = Environment(tools = ['default']) SConscript('datastore/SConscript') site_scons [svn:external] site_scons/site_tools/netcdf.py logx [svn:external] logx/tool_logx.py: def logx(env): datastore/SConscript: env = Environment(tools = ['logx'])



Test Wrapper Method

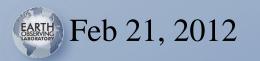


Optimization: rerun.py

env = Environment(tools = ['default', 'rerun'])

if env.Rerun():
 Return()

> scons rerun=1



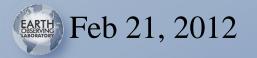
Build Variables

Config file:

QWTDIR="/opt/local/qwt-6.0.1-svn" NIDAS_PATH="/opt/local/nidas" OPT_PREFIX="/opt/local/aeros-qt4" COIN_DIR="/opt/local/Coin-3.1.3" buildmode="debug"

Command line:

scons buildmode=debug



Further Developments

• SCons interactive mode

Feb 21, 2012

- More cross-platform work to do, especially cross-platform tests
- Consolidate test harness scripting, such as running valgrind and analyzing output
- "Next-level integration": multiple EOL projects all built together
- Using the build engine for data processing

SCons and Best Practices

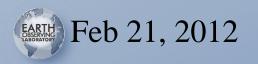
- Build the entire source tree and unit tests in one command, with a single build system
- Careful about *repeatable builds*
- Reusable build configuration scripts for reusable software libraries
- Incorporate standard build products like version headers and documentation
- Software distributions

Feb 21, 2012

Consistent application of compiler flags

SCons Drawbacks

- Performance and scalability
- Internal Python can be complex
 - Hard to track down how build commands are generated
 - Some mysterious bugs
 - Confusion over differences with Make
- More platform-specific coding than we might like
- Learning curve in how to extend or where to insert hooks, but no more than other systems
- Non-mainstream build system hinders code sharing



Conclusion

SCons is a welcome evolution towards a modular build system, in regular use in several EOL software projects, and I see no reason to turn back.

SCons: www.scons.org Email: granger@ucar.edu

NCAR is supported by the National Science Foundation.

