Building tools for SuperB

Marco Corvo

CNRS and INFN

SuperB R&D Computing Workshop

November 17, 2010

Current build system

SoftRelTools was the standard in BaBar, but it has several limitations. Besides technologies have improved since SRT was developed.

SRT issues:

- SRT changed a lot over years
- It supports very old and no more used OSes and software
- Hand written Makefiles, which are difficult to manage and debug
- Impractical code dependencies and complex dependency management (what goes where, etc.)
- Online/SRT base issues where you want most flexibility & agility depended on this huge blob of SRT base

In other words: difficult to clean up or reorganize, Better to write from scratch

Solution(s) for these issues

1. Write SRT from scratch
   - Possible, but not very practical (at least in terms of man power)
2. Use available third party tools
   - Autotools
   - SCons
   - CMake

What are *Autotools*?

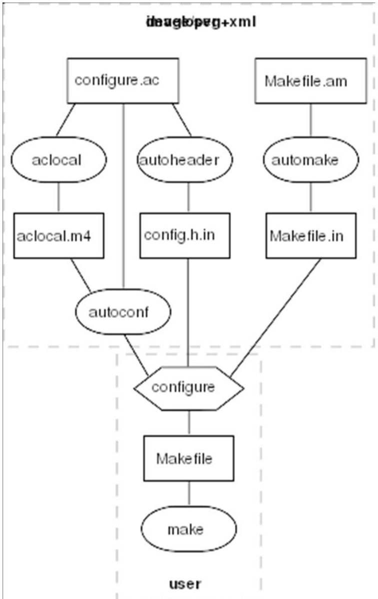
PROS

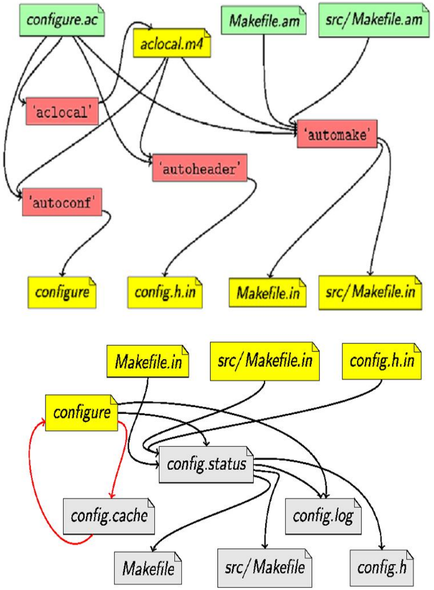
- Complete tool chain of several programs, each with different "macro" syntax
- Easy to use for users (./configure && make && make install)

CONS

- Same as point one of Pros (too many programs)
- Creates big build scripts and helper files even for a hello world example
- Hard to extend, hard to understand

Autotools flowcharts





What is *Scons*?

SCons is an Open Source software construction tool: it's a cross-platform substitute for the classic Make utility with integrated functionality similar to autoconf/automake and compiler caches such as ccache.

- Written in Python (a real OO programming language)
- Reliable, automatic dependency analysis built-in for C, C++ and Fortran
- Built-in support for C, C++, D, Java, Fortran, Yacc, Lex, Qt and SWIG, and building TeX and LaTeX documents
- Improved support for parallel builds

Very similar to CMake (features, cross platform support, behaviour) has the advantage of being written in Python. From my experience not so intuitive as CMake. Need to spend some time to get comfortable with it.

What is *CMake*?

1. Generates native build environments
   - UNIX/Linux: **Makefiles**
   - Windows: **VS Projects/Workspaces**
   - Mac OS: **Xcode**
2. Opensource
3. Cross-platform
4. Integrates testing and packaging systems

CMake features

1. Manage complex, large build environments (KDE4)
2. Very Flexible and Extensible
   - Support for Macros
   - Modules for finding/configuring software (bunch of modules already available)
   - Extend CMake for new platforms and languages
   - Create custom targets/commands
   - Run external programs
3. Very simple, intuitive syntax
4. Support for regular expressions (*nix style)
5. Support for In-Source and Out-of-Source builds
6. Cross Compiling
7. Integrated Testing and Packaging (Ctest, CPack)

Why Use *CMake*?

PROS

1. CMake depends only on C++ compiler
2. CMake supports great variety of platforms (basically every ***ix, Mac OS, Windows**)
3. CMake generates only Makefiles for all supported platforms
4. CMake additionally can produce project files for IDE's (KDevelop, XCode, VStudio)

Why Use *CMake*?

Pros (cont'd)

1. More usefull error messages when making a mistake in editing input files
2. Easy to use configure-like framework
3. CMake has simple syntax
4. CMake has a testing framework
5. CMake is faster than autotools (does not use libtools)

Furthermore, talking with CMS people, they also would use CMake if they were to write from scratch their build system

Why Use *CMake*?

Special interesting features

CMake combines further subsystems

1. CTest: used to automate updating (using CVS for example), configuring, building, testing, performing memory checking, performing coverage, and submitting results to a CDash or Dart dashboard system
2. CPack: software packaging tool which can be used with or without CMake and is able to generate many different flavours of installers (RPM, Debian, DragNDrop, PackageMaker)
3. CDash: CDash is an open source, web-based software testing server. CDash aggregates, analyzes and displays the results of software testing processes submitted from clients located around the world. Developers depend on CDash to convey the state of a software system.

Current status of FastSim build

Currently the prototype to build SuperB software with *CMake* works with the Head (trunk) of FastSim V0.2.6:

- CMakeLists files in place for every FastSim package
- Bunch of *CMake* macros and scripts to configure the release
  - Third party packages configuration and management (CLHEP, Root. . .)
  - Specific platform settings (compiler definitions and flags)
  - Bash script to run cmake executable in a more friendly way
- Already ongoing tests using the CTest framework

Next development release V0.2.7 will come with the support to *CMake* build.

Short term

- First impression and first experience with FastSim is good. *CMake* is simple to use, flexible and has a large number of modules to set up and manage third party software (for FastSim I used *CMake* modules to configure Root and CLHEP)
- Current system is still a prototype which needs further and deeper work in order to turn it into a stable and widely usable one

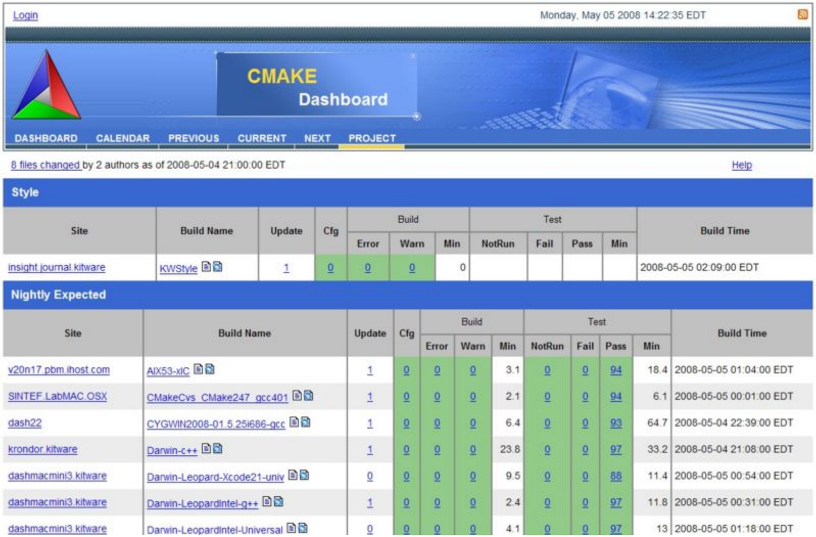
First goal is to get a really usable and useful *CMake* framework for FastSim builds

Future plans

Future plans consider developing prototypes with *CMake* combined with *CPack*, *CTest* and *CDash*

1. *CPack*
   - CPack can be used also without *CMake* as a standalone tool
   - Same syntax as *CMake*
   - Support for many different package generators (RPM, Debian, OSX, Cygwin)
2. *CTest*
   - Useful also combined with Valgrind to perform code profiling and to submit results to a CDash
3. *CDash*
   - Useful to set up a sort of alarm system based on email notification when build fails

CDash snapshot



Login   Monday, May 05 2008 14:22:35 EDT

8 files changed by 2 authors as of 2008-05-04 21:00:00 EDT   Help

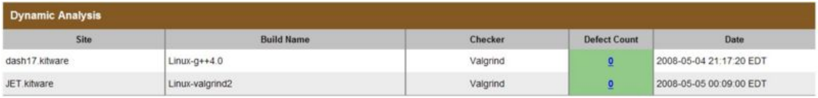
Style

Site	Build Name	Update	Cfg	Build			Test				Build Time
				Error	Warn	Min	NotRun	Fail	Pass	Min	
insight.journal.kitware	KWStyle	1	0	0	0	0					2008-05-05 02:09:00 EDT

Nightly Expected

Site	Build Name	Update	Cfg	Build			Test				Build Time
				Error	Warn	Min	NotRun	Fail	Pass	Min	
v20n17.pbm.ihost.com	AIX53-xlC	1	0	0	0	3.1	0	0	94	18.4	2008-05-05 01:04:00 EDT
SINTEF.LabMAC.OSX	CMakeCvs_CMake247_gcc401	1	0	0	0	2.1	0	0	94	6.1	2008-05-05 00:01:00 EDT
dash22	CYGWIN2008-01.5.25i686-gcc	1	0	0	0	6.4	0	0	93	64.7	2008-05-04 22:39:00 EDT
krondor.kitware	Darwin-c++	1	0	0	0	23.8	0	0	97	33.2	2008-05-04 21:08:00 EDT
dashmacmini3.kitware	Darwin-Leopard-Xcode21-univ	0	0	0	0	9.5	0	0	88	11.4	2008-05-05 00:54:00 EDT
dashmacmini3.kitware	Darwin-LeopardIntel-g++	1	0	0	0	2.4	0	0	97	11.8	2008-05-05 00:31:00 EDT
dashmacmini3.kitware	Darwin-LeopardIntel-Universal	0	0	0	0	4.1	0	0	97	13	2008-05-05 01:18:00 EDT

CDash with Valgrind



Dynamic Analysis

Site	Build Name	Checker	Defect Count	Date
dash17.kitware	Linux-g++4.0	Valgrind	0	2008-05-04 21:17:20 EDT
JET.kitware	Linux-valgrind2	Valgrind	0	2008-05-05 00:09:00 EDT