Modern CMake

Open source tools to build, test and package software: CMake, CTest, CPack, CDash
Bill Hoffman

• CTO and a founder of Kitware Inc
• Originator of CMake build tool
• Barefoot/Sandals Ultra distance runner
Collaborative software R&D

Technical computing
Algorithms & applications
Software process & infrastructure
Support & training
Open source leadership

Supporting all sectors

Industry, government & academia
Kitware’s customers & collaborators

Over 75 **academic** institutions…

- Harvard
- Massachusetts Institute of Technology
- University of California, Berkeley
- Stanford University
- California Institute of Technology
- Imperial College London
- Johns Hopkins University
- Cornell University
- Columbia University
- Robarts Research Institute
- University of Pennsylvania
- Rensselaer Polytechnic Institute
- University of Utah
- University of North Carolina

Over 50 **government** agencies and labs…

- National Institutes of Health (NIH)
- National Science Foundation (NSF)
- National Library of Medicine (NLM)
- Department of Defense (DOD)
- Department of Energy (DOE)
- Defense Advanced Research Projects Agency (DARPA)
- Army Research Lab (ARL)
- Air Force Research Lab (AFRL)
- Sandia (SNL)
- Los Alamos National Labs (LANL)
- Argonne (ANL)
- Oak Ridge (ORNL)
- Lawrence Livermore (LLNL)

Over 100 **commercial** companies…

- Automotive
- Aircraft
- Defense
- Energy technology
- Environmental sciences
- Finance
- Industrial inspection
- Oil & gas
- Pharmaceuticals
- Publishing
- 3D Mapping
- Medical devices
- Security
- Simulation
Open source platforms

**VTK & ParaView** interactive visualization and analysis for scientific data

**ITK & 3D Slicer** medical image analysis and personalized medicine research

**CMake** cross-platform build system

- CDash, CTest, CPack, software process tools

**Resonant** informatics and infovis

**KWIVER** computer vision image and video analysis

- Other areas include: Simulation, ultrasound, physiology, information security, materials science, …
What is CMake?

• CMake is the **cross-platform, open-source build system** that lets you use the **native development tools** you love the most.

• It’s a build system **generator**

• It takes **plain text files** as input that describe your project and **produces** project files or make files for use with a wide variety of **native development tools**.

• **Family of Software Development Tools**
  – Build = CMake
  – Test = CTest/CDash
  – Package = CPack
Modern CMake

• CMake is code, treat CMakeLists.txt like the rest of the code, comments
• CMake Targets are objects with public and private properties
• Import third party libraries as imported targets
• Export your libraries so they can be used by other CMake projects
CMake: History

- Built for the Insight Segmentation and Registration Toolkit (ITK) [http://www.itk.org](http://www.itk.org)

- Funded by National Library of Medicine (NLM): part of the Visible Human Project
  - Data CT/MR/Slice 1994/1995
  - Code (ITK) 1999
    - Cmake Release-1-0 branch created in 2001
CMake has broad usage in the C++ world

KDE 2006 - Tipping Point!

- 7000+ downloads per day from www.cmake.org

Indeed.com CMake jobs Full-time (263)
Visual C++ Team Blog

C++ tutorials, C and C++ news, and information about the C++ IDE Visual Studio from the Microsoft C++ team.

CMake support in Visual Studio

October 5, 2016 by Marian Luparu [MSFT] // 56 Comments
CMake: Features

• Automatic **dependency** generation (C, C++, CUDA, Fortran)
  – build a target in some directory, and everything this target depends on will be up to date
  – If a header file changes the correct files will be built.
Fortran Module Order

Yes, it can get confusing. I am not aware of any references, others might be. The Intel Fortran Users guide discusses using modules and states the requirement rather succinctly as:

You need to make sure that the module files are created before they are referenced by another program or subprogram.

• Old way:  make;make;make until it works
• CMake way:  cmake; make or cmake; ninja
  – CMake will automatically order Fortran files based on use statements in the code for a library
Ninja

• Improved parallelism for ninja builds in CMake 3.9

• Can control pools to limit concurrent links
Random list of things CMake does well

- Excellent install commands
- Excellent packaging tools
- Ability to find/link system libraries
- Handles shared libraries and versioning across platforms (linux, mac, windows)
- Keeps up to date with current and obscure compilers
- Cross platform development support (Linux/Mac/Windows/android/HPC)
- Integration of static/dynamic analysis tools
- Integration of code coverage tools
- Excellent backwards compatibility with itself (policy system)
- Open and dynamic community accepting of changes small and large
- Supports many workflows and IDEs
CMake Workflow

1. Edit files in the source tree

2. Run cmake-gui (or cmake or ccmake) to configure and generate native build system files

3. Open project files from the build tree and use the native build tools

```
 cmake -GNinja
```
Out of source builds

Project Source Tree
Library1 (CMakeLists.txt foo.cxx bar.cxx)
Library2 (CMakeLists.txt car.cxx car.h fun.F90)
Library3 (CMakeLists.txt gpu.cu ml.cxx)
App1 (CMakeLists.txt exe.cxx)
App2 (CMakeLists.txt exegui.cxx)

Run CMake

-Clang Build Tree
CMakeCache.txt – stores info specific to this build
build.ninja

-GCC Build Tree
CMakeCache.txt – stores info specific to this build
build.ninja

-GNinja

-GNinja

-Gunix Makefiles

GCC Build Tree
CMakeCache.txt – stores info specific to this build
Makefile
Modern CMake
CMake Then and Now

CMake 2001

CMakeLists.txt
SUBDIRS = \Code/Common \
ME = ITK

Code/Common/CMakeLists.txt
ME = ITKCommon

COMPILE_CLASSES =\itkDataObject \itkDirectory

WIN32_CLASSES =\itkWin32OutputWindow

CMake 2008

CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(ITK)
add_subdirectory(Code/Common)

Code/Common/CMakeLists.txt
set(ITKCommonSources itkDataObject.cxx itkDirectory.cxx)
if(WIN32)
  set(ITKCommonSources ${ITKCommonSources} itkWin32OutputWindow.cxx)
endif()
add_library(ITKCommon ${ITKCommonSources})

CMake 2018

CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(ITK)
add_subdirectory(Code/Common)

Code/Common/CMakeLists.txt
add_library(ITKCommon)
target_sources(ITKCommon PRIVATE itkDataObject.cxx itkDirectory.cxx ...) 
if(WIN32)
target_sources(ITKCommon PRIVATE itkWin32OutputWindow.cxx) 
endif()
Targets are Objects

Library

add_library()
target_compile_definitions
target_compile_features
target_include_directories
target_link_libraries
target_sources
get_target_property
set_target_property

Executable

add_executable()
target_compile_definitions
target_compile_features
target_include_directories
target_link_libraries
target_sources
get_target_property
set_target_property
Targets are Objects

• Developer can focus on a single target and not the whole system
  – What include directories will users need?
  – What –D flags will users need?
  – What compile flags will users need?
  – What version of C++ will users need?
  – What flags and options will the users not need?
    • controlled with public and private declarations
“Usage Requirements” aka Modern CMake

Modern style: target-centric

```cmake
target_include_directories(mylib PUBLIC "mydir")
```

mylib and anything that links to gets `-Imydir`

Classic style: directory-centric

```cmake
include_directories("mydir")
```

Targets in this directory and subdirs get `-Imydir`
Before Usage Requirements existed we used directory scoped commands such as:

- `include_directories`
- `compile_definitions`
- `compile_options`

Consumers have to know:

- Does the dependency generate build tree files
- Does the dependency use any new external package
Modern CMake / Usage Requirements

• Modern CMake goal is to have each target fully describe how to properly use it.
• No difference between using internal and external generated targets
Modern CMake layout independent

Diagram:
- Root
  - Directory
    - Executable
    - Library B
    - Library A
  - Directory
    - Executable
    - Library A
    - Library B

- Root
  - Executable
  - Library A
  - Library B
Modern CMake Mostly about not using these commands

- `add_compile_options()`
- `add_definitions()`
- `include_directories()`
- `link_directories()`
- `link_libraries()`

And treating targets like objects
Usage Requirements

- `target_link_libraries` is the foundation for usage requirements
- This foundation is formed by
  - PUBLIC
  - PRIVATE
  - INTERFACE

```
target_link_libraries(trunk PRIVATE root)  
target_link_libraries(leaf PUBLIC trunk)
```
target_include_directories

- Propagates include directories

```cpp
target_include_directories(leaf INTERFACE ${zlib_dir})
```

- Anything that links to leaf will automatically have the zlib_dir on the include line
target_compile_options

• Propagates compiler options

```target_compile_options(trunk PRIVATE -march=native)```

• Only trunk will be built optimized for the current hardware. Anything that links to trunk will not get this flag
target_compile_definitions

• Propagates pre-processor definitions

```python
target_compile_definitions(root PUBLIC "ROOT_VERSION=42")
```

• Root will have ROOT_VERSION defined and anything that links to it will also
INTERFACE Libraries

• An INTERFACE library target does not directly create build output, though it may have properties set on it and it may be installed, exported, and imported.

add_library(root INTERFACE)
target_compile_features(root INTERFACE cxx_std_11)
IMPORTING / EXPORTING
Imported Targets

- Logical name for an outside library
- Reference like any other target

```cpp
add_library(math STATIC IMPORTED)
set_property(TARGET math PROPERTY IMPORTED_LOCATION /usr/lib/libm.a)
target_link_libraries(trunk PUBLIC math)
```
Imported Targets

- Per-configuration import rules
- Better than optimized/debug keywords

```cpp
find_library(math_REL NAMES m)
find_library(math_DBG NAMES md)
add_library(math STATIC IMPORTED)
set_target_properties(math
  PROPERTIES
    IMPORTED_LOCATION "${math_REL}"
    IMPORTED_LOCATION_DEBUG "${math_DBG}"
)

target_link_libraries(trunk PUBLIC math)
```
Exporting Targets

• Install rules can generate imported targets

```cpp
add_library(parasite STATIC eat_leaf.cxx)
install(TARGETS parasite root trunk leaf
       DESTINATION lib
       EXPORT tree-targets)
install(EXPORT tree-targets
       DESTINATION lib/tree)
```

• Installs library and target import rules
  – `<prefix>/lib/tree/libparasite.a`
  – `<prefix>/lib/tree/tree-targets.cmake`
Conditional Includes

- Able to specify include directories based on if we are building a library or using the installed version

```cmake
target_include_directories(trunk PUBLIC
  $<BUILD_INTERFACE:
    ${CMAKE_CURRENT_SOURCE_DIR}/path/in/src/tree>
  $<INSTALL_INTERFACE:
    $<INSTALL_PREFIX>/include/package/>
)
```
Generating Export Package

- This is constructing components needed for the CMake-aware config package
- CMakePackageConfigHelpers can help with the generation of the `<Name>Config.cmake` file
- Exporting of find package calls has to be replicated inside `<Name>Config.cmake`, but CMakeFindDependencyMacro helps simplify this
Generating Export Package

```cpp
#include(CMakePackageConfigHelpers)
# generate the config file that is includes the exports
configure_package_config_file(Config.cmake.in
  "${CMAKE_CURRENT_BINARY_DIR}/TreeConfig.cmake"
  INSTALL_DESTINATION "lib/cmake/example"
)

#include(CMakeFindDependencyMacro)
find_dependency(PNG REQUIRED)

#include("${CMAKE_CURRENT_LIST_DIR}/TreeTargets.cmake")
```
# Create imported target root
add_library(root INTERFACE IMPORTED)

set_target_properties(root PROPERTIES
    INTERFACE_COMPILE_DEFINITIONS "ROOT_VERSION=42"
    INTERFACE_COMPILE_FEATURES "cxx_std_11"
    INTERFACE_COMPILE_OPTIONS "\$<\$<NOT:\$<CONFIG:DEBUG>>:;>"
)

# Create imported target trunk
add_library(trunk SHARED IMPORTED)

set_target_properties(trunk PROPERTIES
    INTERFACE_INCLUDE_DIRECTORIES "${_IMPORT_PREFIX}/include/parent"
)

# Create imported target leaf
add_library(leaf SHARED IMPORTED)

set_target_properties(leaf PROPERTIES
    INTERFACE_LINK_LIBRARIES "trunk"
)
add_library(support STATIC support_functions.cu)
set_target_properties(support PROPERTIES
    CUDA_SEPARABLE_COMPILATION ON
    POSITION_INDEPENDENT_CODE ON)
target_link_libraries(support PRIVATE compiler_info)

add_library(black_scholes
    black_scholes/Serial.cpp
    black_scholes/Parallel.cu
)
target_link_libraries(black_scholes PUBLIC compiler_info support)

[20%] Building CUDA object CMakeFiles/support_functions.cu.o
[40%] Linking CUD...
INSTALL RULES
Install Rules

• Specify rules to run at install time
• Can install targets, files, or directories

```cpp
add_library(leaf SHARED leaf.cxx)
install(TARGETS root trunk leaf parasite
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib)
```
Install Rules

• To install files:

```bash
install(FILES
    trunk.h
    leaf.h
    DESTINATION include
)
```
Using Config Modules

• `find_package` also supports config modules
• Config modules are generated by CMake export command
• Automatically generate import targets with all information, removing the need for consuming projects to write a find module
CMake Scripts

- **cmake –E command**
  - Cross platform command line utility for:
  - Copy file, Remove file, Compare and conditionally copy, time, others

- **cmake –P script.cmake**
  - Cross platform scripting utility
  - Does not generate CMakeCache.txt
  - Ignores commands specific to generating build environment
add_library(root OBJECT root.cxx)
add_library(trunk OBJECT trunk.cxx)
add_library(leaf SHARED leaf.cxx)
target_link_libraries(leaf root trunk)

[100%] Linking CXX shared library libleaf.so
/usr/bin/c++ -fPIC -shared -Wl,-soname,libleaf.so
-o libleaf.so leaf.cxx.o root.cxx.o trunk.cxx.o
CTEST
Automatic Testing Benefits

“Automated Software Testing,”
1999, Dustin, et al. Addison Wesley
Video of ParaView Nightly Testing
Testing with CMake

• Testing needs to be enabled by calling include(CTest) or enable_testing()

```cpp
add_test(NAME testname
COMMAND exename arg1 arg2 ...)
```

– Executable should return 0 for a test that passes

• ctest – an executable that is distributed with cmake that can run tests in a project.

• CDash – Web based dashboard to show testing results.
CTest

• Run ctest at the top of a binary directory to run all tests

$ ctest
Test project /tmp/example/bin
  Start 1: case1
  1/1 Test #1: case1 ................................ Passed 0.00 sec
  Start 2: case2
  2/2 Test #2: case2 ........................................ Passed 0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) = 0.01 sec
CTest

• -j option allows you to run tests in parallel
• -R option allows you to choose a test
• Running tests from Makefiles or projects
  – make test
  – Build RUN_TESTS project
• ctest --help for more information
GoogleTest integration

```cpp
include(GoogleTest)
add_executable(tests tests.cpp)
target_link_libraries(tests GTest::GTest)
```

- **gtest_discover_tests**: new in CMake 3.10.
  - CMake asks the test executable to list its tests. Finds new tests without rerunning CMake.

```
gtest_discover_tests(tests TEST_PREFIX new:)
```
Static Analysis

• Supported tools include:
  – include-what-you-use
  – link-what-you-use
  – clang-tidy
  – cpplint
  – cppcheck

• Setup instructions available here:
CDash
Software Process Dashboards

- Automated cross-platform testing is triggered
- Results of testing are stored on a dashboard
- Software developer commits changes to source code / data repository
- Software developer is notified of any issues that occurred during testing
<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Update Files</th>
<th>Update Error</th>
<th>Update Warn</th>
<th>Configure Files</th>
<th>Configure Error</th>
<th>Configure Warn</th>
<th>Build Files</th>
<th>Build Error</th>
<th>Build Warn</th>
<th>Test Not Run</th>
<th>Test Fail</th>
<th>Test Pass</th>
<th>Build Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>dashmacmini5.kitware</td>
<td>KWStyle</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20 hours ago</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash2win64-windows.kitware</td>
<td>Windows-VS9-ninja</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>309</td>
<td>7 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash2win64.kitware</td>
<td>Windows-icl-11.1-64</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>314</td>
<td>10 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash2win64.kitware</td>
<td>Windows-icl-11.1-32</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>314</td>
<td>12 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vs8.elementtech</td>
<td>Win64-VS80</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>325</td>
<td>8 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>amber12.kitware</td>
<td>Win64-vs10-WINSDK-7.1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>314</td>
<td>19 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash2win64.kitware</td>
<td>Win64-vs10-Tv90</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>320</td>
<td>15 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash2win64.kitware</td>
<td>Win64-vs10</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>317</td>
<td>17 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vs8.elementtech</td>
<td>Win64-nmake80</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>318</td>
<td>8 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dash2win64.kitware</td>
<td>Win64-nmake10</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>312</td>
<td>8 hours ago</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CDash works with other CI tools

- Jenkins
- Buildbot
- Gitlab/CI
- ctest scripts and cronjobs
- CircleCI
- Travis
Search for relevant results

### Filters

*Site*: contains *microsoft*  
*Group*: is *Nightly Expected*  
*Tests Failed*: is greater than *0*

### Nightly Expected

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Update</th>
<th>Configure</th>
<th>Build</th>
<th>Not Run</th>
<th>Fail</th>
<th>Pass</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>gillesk.microsoft</td>
<td>VS2017 x86.rel</td>
<td>602d4c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4+</td>
<td>4+</td>
<td>10 hours ago</td>
</tr>
<tr>
<td>gillesk.microsoft</td>
<td>VS2015 x64.rel</td>
<td>602d4c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4+</td>
<td>4+</td>
<td>10 hours ago</td>
</tr>
<tr>
<td>gillesk.microsoft</td>
<td>VS2012 x86.rel</td>
<td>602d4c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3+</td>
<td>3+</td>
<td>5 hours ago</td>
</tr>
<tr>
<td>gillesk.microsoft</td>
<td>VS2012 x64.rel</td>
<td>602d4c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3+</td>
<td>3+</td>
<td>5 hours ago</td>
</tr>
<tr>
<td>gillesk.microsoft</td>
<td>VS2017 x64.rel</td>
<td>602d4c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3+</td>
<td>3+</td>
<td>10 hours ago</td>
</tr>
<tr>
<td>gillesk.microsoft</td>
<td>VS2015 x86.rel</td>
<td>602d4c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3+</td>
<td>3+</td>
<td>10 hours ago</td>
</tr>
</tbody>
</table>
Compare results across systems

Testing summary for `kwsys.testConsoleBuf` performed between 2018-09-13T01:00:00 and 2018-09-14T01:00:00

98% passed, 2 failed out of 104.

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Build Stamp</th>
<th>Status</th>
<th>Time (s)</th>
<th>Build Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>gillesk.microsoft</td>
<td>VS2017 x86.rel</td>
<td>20180913-0100-Nightly</td>
<td>Failed</td>
<td>11.02</td>
<td>602d4c6e06673b9864ad2f8bb3d706d5bd440c1a</td>
</tr>
<tr>
<td>trinsic.kitware</td>
<td>vs14-64-ninja</td>
<td>20180913-0100-Nightly</td>
<td>Failed</td>
<td>13.66</td>
<td>602d4c6e06673b9864ad2f8bb3d706d5bd440c1a</td>
</tr>
<tr>
<td>aaargh.kitware.com</td>
<td>Linux-EL7-Intel-16.0.0</td>
<td>20180913-0100-Nightly</td>
<td>Passed</td>
<td>0.02</td>
<td>602d4c6e06673b9864ad2f8bb3d706d5bd440c1a</td>
</tr>
<tr>
<td>aaargh.kitware.com</td>
<td>Linux-EL7-Intel-16.0.1</td>
<td>20180913-0100-Nightly</td>
<td>Passed</td>
<td>0.02</td>
<td>602d4c6e06673b9864ad2f8bb3d706d5bd440c1a</td>
</tr>
<tr>
<td>aaargh.kitware.com</td>
<td>Linux-EL7-Intel-16.0.2</td>
<td>20180913-0100-Nightly</td>
<td>Passed</td>
<td>0.02</td>
<td>602d4c6e06673b9864ad2f8bb3d706d5bd440c1a</td>
</tr>
</tbody>
</table>
Test output

WaitForSingleObject returned unexpected status 0x102
In function testConsole, line 718: WaitForSingleObject#2 failed!
Failed with error: 0x2!
Error message: The system cannot find the file specified.
# CDash Subproject Support

The CDash interface displays a dashboard for tracking the status of various projects and subprojects. The main project is highlighted, along with subprojects such as Trilinos, Teuchos, and Kokkos. The dashboard provides a comprehensive view of the error, warning, and pass statuses for configuration and build processes, along with test results and last submission dates.
CDash Queries

Show the HEAVY builds for the last two weeks:

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Update Files</th>
<th>Error Files</th>
<th>Warn Files</th>
<th>Configure Error</th>
<th>Error Files</th>
<th>Warn Files</th>
<th>Not Run Files</th>
<th>Fail Files</th>
<th>Pass Files</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>21 hours ago</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Jun 07, 2016 - 01:10 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>Jun 06, 2016 - 01:10 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Jun 05, 2016 - 01:10 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Jun 04, 2016 - 01:10 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>Jun 03, 2016 - 01:10 EDT</td>
</tr>
</tbody>
</table>
CDash Queries

Show most expensive tests yesterday:

<table>
<thead>
<tr>
<th>Site</th>
<th>Build Name</th>
<th>Test Name</th>
<th>Status</th>
<th>Time</th>
<th>Details</th>
<th>Build Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>MPACT_exe_testProgression_Problems_1-min</td>
<td>Passed</td>
<td>13111.8</td>
<td>Completed</td>
<td>2016-06-07T03:10:34 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>MPACT_exe_testProgression_Problems_2-min</td>
<td>Passed</td>
<td>12943.4</td>
<td>Completed</td>
<td>2016-06-07T03:10:34 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>VeraAPImpact_p6a_mpact_dep</td>
<td>Passed</td>
<td>5739.74</td>
<td>Completed</td>
<td>2016-06-07T12:48:23 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>MPACT_exe_testIVS_ap1000_IFBAOnly</td>
<td>Passed</td>
<td>4886.6</td>
<td>Completed</td>
<td>2016-06-07T03:10:34 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>MPACT_exe_testIVS_ap1000_REGION4</td>
<td>Passed</td>
<td>4106.07</td>
<td>Completed</td>
<td>2016-06-07T03:10:34 EDT</td>
</tr>
<tr>
<td>james007.ornl.gov</td>
<td>Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY</td>
<td>MPACT_exe_testIVS_ap1000_REGION5</td>
<td>Passed</td>
<td>4012.66</td>
<td>Completed</td>
<td>2016-06-07T03:10:34 EDT</td>
</tr>
</tbody>
</table>
CTest Command Wrappers Output

Build Time: 2009-05-04T01:53:37 MDT
Found 1 Warnings

Errors are here.

Warning while building cpp object file "CMakeFiles/Kokkos_BaseSparseSolve.dir/cxx_main.cpp.o" in target Kokkos_BaseSparseSolve.

Source File: packages/kokkos/test/BaseSparseSolve/cxx_main.cpp
Label: Kokkos

Command:
"/Users/bmpersc/bin/gcc-4.3.3/bin/gcc" -ansi -pedantic -Wall -Wno-long-long -Wwrite-strings
"-g" "-00" "-D_GLIBCXX_DEBUG" "$" -I/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/BUILD/packages/kokkos/src
"-I/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/src"
"-I/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/test/BaseSparseSolve.../BaseSparseMultiply"
"-f" "CMakeFiles/Kokkos_BaseSparseSolve.dir/cxx_main.cpp.o" "$" "$" -I/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/test/BaseSparseSolve/cxx_main.cpp"
"-I/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/test/BaseSparseSolve/cxx_main.cpp"
"-I/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/test/BaseSparseSolve/cxx_main.cpp"

Exit Condition: 0

Standard Output:
/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/src/Kokkos_BasesparseSolve.hpp: In member function
/Users/bmpersc/nightly/Trilinos.base/SERIAL_DEBUG/Trilinos/packages/kokkos/src/Kokkos_BasesparseSolve.hpp:666: instanti
## Coverage Display: `Gcov/Bullseye`

<table>
<thead>
<tr>
<th>FileName</th>
<th>Coverage</th>
<th>Function Calls</th>
<th>Instructions</th>
<th>Line Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/Source/makefiles/BuildTargetGenerator.cxx</code></td>
<td>68.31%</td>
<td>36</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><code>/Source/makefiles/ExecutableTargetGenerator.cxx</code></td>
<td>68.33%</td>
<td>371</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td><code>/Source/makefiles/TargetGenerator.cxx</code></td>
<td>68.83%</td>
<td>70</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><code>/Source/makefiles/ExecutableTargetGenerator.cxx</code></td>
<td>70.00%</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><code>/Source/makefiles/TargetGenerator.cxx</code></td>
<td>70.83%</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>/Source/makefiles/TargetGenerator.cxx</code></td>
<td>70.83%</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><code>/Source/makefiles/BuildTool.cxx</code></td>
<td>71.01%</td>
<td>44</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>/Source/makefiles/MakeFiles.cxx</code></td>
<td>71.74%</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><code>/Source/makefiles/auto_ptr.hxx</code></td>
<td>71.88%</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>/Source/makefiles/CommandLineArguments.cxx</code></td>
<td>71.68%</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><code>/Source/makefiles/Command.cxx</code></td>
<td>72.07%</td>
<td>57</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><code>/Source/makefiles/Target.cxx</code></td>
<td>72.24%</td>
<td>20</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Coverage produced by Bullseye command:**
```
/home/bullseye/build/Bullseye.mak
```

**Coverage produced by Bullseye command:**
```
/home/bullseye/build/Bullseye.mak
```
Valgrind / Purify

Dynamic analysis started on 2009-05-03 03:38:06

Site Name: dash17.bitware
Build Name: Linux-g++4.0

Dynamic analysis started on 2009-05-04 03:37:17

Site Name: dash17.bitware
Build Name: Linux-g++4.0

Valgrind / Purify
CDash Image Difference
CPack
What is CPack

- CPack is bundled with CMake
- Creates professional platform specific installers
CPack Features

• Supports CMake-based and non-CMake-based projects

• Unix
  – TGZ and self-extracting TGZ (STGZ)

• Windows
  – WiX – MSI installers
  – NullSoft Scriptable Install System (NSIS / NSIS64)

• Mac OSX
  – DragNDrop
  – PackageMaker

• Deb
  – Debian packages

• RPM
  – RPM package manager
Using CPack

- On Windows install command line ZIP program, NSIS and WiX
- Setup your project to work with cpack
  - Get make install to work
    - install(...)
    - make sure your executables work with relative paths and can work from any directory
  - Set cpack option variables if needed
  - include(CPack)
Now that you are inspired

• Read “how to write a CMake buildsystem”
• Explore the CMake documentation
  – [https://www.cmake.org/cmake/help/v3.8/](https://www.cmake.org/cmake/help/v3.8/)