Tribits User Quickstart Documentation

Release 0.1

Roscoe A. Bartlett

June 11, 2013

Author Roscoe A. Bartlett

Contact bartlett.roscoe@gmail.com

Abstract This document is a generic guide for users of a TriBITS project who need to configure, build, install, and test software that uses the CMake-based TriBITS system.

Contents

CHAPTER

ONE

1.1 INTRODUCTION

This document is a generic guide for users of a TriBITS project who need to configure, build, install, and test software that uses the CMake-based TriBITS system. The generic <Project>_ and <PROJECT>_ prefix is used for regular-case and upper-case project names.

CHAPTER

TWO

1.2 GETTING SET UP TO USE CMAKE

2.1 1.2.1 Installing a binary release of CMake [casual Trilinos users]

Download and install the binary (currently version 2.8 is required) from:

http://www.cmake.org/cmake/resources/software.html

2.2 1.2.2 Installing CMake from source [Trilinos developers and experienced users]

If you have access to the Trilinos CVS repository, then install CMake with:

```
$ $TRILINOS_HOME/cmake/python/install-cmake.py \
  --install-dir=INSTALL_BASE_DIR
```

This will result in cmake and related CMake tools being installed in INSTALL_BASE_DIR/bin.

Getting help for installing CMake with this script:

```
$ $TRILINOS_HOME/cmake/python/install-cmake.py --help
```

NOTE: you will want to read the help message about how to use sudo to install in a privileged location (like the default /usr/local/bin).

CHAPTER

THREE

1.3 GETTING HELP

3.1 1.3.1 Finding CMake help at the website

http://www.cmake.org

3.2 1.3.2 Building CMake help locally

```
$ cmake --help-full cmake.help.html
```

(Open your web browser to the file cmake.help.html)

CHAPTER

FOUR

1.4 CONFIGURING (MAKEFILE GENERATOR)

4.1 1.4.1 Setting up a build directory

```
$ mkdir SOME_BUILD_DIR
$ cd SOME_BUILD_DIR
```

NOTE: You can create a build directory from any location you would like. It can be a sub-directory of the Trilinos base source directory or anywhere else.

NOTE: If you mistakenly try to configure for an in-source build (e.g. with 'cmake .') you will get an error message and instructions on how to resolve the problem by deleting the generated CMakeCache.txt file (and other generated files) and then follow directions on how to create a different build directory as shown above.

4.2 1.4.2 Basic configuration of Trilinos

1. [Recommended] Create a 'do-configure' script such as:

   ```
   EXTRA_ARGS=$@

   cmake \
     -D CMAKE_BUILD_TYPE:STRING=DEBUG \
     -D Trilinos_ENABLE_TESTS:BOOL=ON \
     $EXTRA_ARGS \
     ${TRILINOS_HOME}
   ```

   and then run it with:

   ```
   ./do-configure [OTHER OPTIONS] -DTrilinos_ENABLE_<TRIBITS_PACKAGE>=ON
   ```

   where <TRIBITS_PACKAGE> is Epetra, AztecOO, etc. and TRILINOS_HOME is et to the Trilinos source base directory (or your can just give it explicitly).

   See *Trilinos/sampleScripts/*cmake* for real examples.

2. [Recommended] Create a CMake file fragment and point to it.

   Create a do-configure script like:

   ```
   EXTRA_ARGS=$@

   cmake \
     -D Trilinos_CONFIGURE_OPTIONS_FILE:FILEPATH=MyConfigureOptions.cmake \
     -D Trilinos_ENABLE_TESTS:BOOL=ON \
   ```

```
  $EXTRA_ARGS \
  ${TRILINOS_HOME}
```

where MyConfigureOptions.cmake might look like:

```
SET(CMAKE_BUILD_TYPE DEBUG CACHE STRING "" FORCE)
SET(Trilinos_ENABLE_CHECKED_STL ON CACHE BOOL "" FORCE)
SET(BUILD_SHARED_LIBS ON CACHE BOOL "" FORCE)
...
```

Using a configuration fragment file allows for better reuse of configure options across different configure scripts and better version control of configure options.

NOTE: You can actually pass in a list of configuration fragment files which will be read in the order they are given.

NOTE: If you do not use 'FORCE' shown above, then the option can be overridden on the cmake command line with -D options. Also, if you don't use 'FORCE' then the option will not be set if it is already set in the case (e.g. by another configuration fragment file prior in the list).

3. Using ccmake to configure

```
$ ccmake $TRILINOS_HOME
```

4. Using the QT CMake configuration GUI:

   On systems where the QT CMake GUI is installed (e.g. Windows) the CMake GUI can be a nice way to configure Trilinos if you are a user. To make your configuration easily repeatable, you might want to create a fragment file and just load it by setting Trilinos_CONFIGURE_OPTIONS_FILE (see above) in the GUI.

4.3 1.4.3 Selecting the list of packages to enable

1. Configuring a package(s) along with all of the packages it can use:

```
$ ./do-configure \
   -D Trilinos_ENABLE_<TRIBITS_PACKAGE>:BOOL=ON \
   -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
   -D Trilinos_ENABLE_TESTS:BOOL=ON
```

   NOTE: This set of arguments allows a user to turn on <TRIBITS_PACKAGE> as well as all packages that <TRIBITS_PACKAGE> can use. However, tests and examples will only be turned on for <TRIBITS_PACKAGE> (or any other packages specifically enabled).

2. Configuring Trilinos to test all effects of changing a given package(s):

```
$ ./do-configure \
   -D Trilinos_ENABLE_<TRIBITS_PACKAGE>:BOOL=ON \
   -D Trilinos_ENABLE_ALL_FORWARD_DEP_PACKAGES:BOOL=ON \
   -D Trilinos_ENABLE_TESTS:BOOL=ON
```

   NOTE: The above set of arguments will result in package <TRIBITS_PACKAGE> and all packages that depend on <TRIBITS_PACKAGE> to be enabled and have all of their tests turned on. Tests will not be enabled in packages that do not depend on <TRIBITS_PACKAGE> in this case. This speeds up and robustifies pre-checkin testing.

3. Configuring to build all stable packages with tests and examples:

```
$ ./do-configure \
   -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=ON \
   -D Trilinos_ENABLE_TESTS:BOOL=ON
```

NOTE: Specific packages can be disabled with Trilinos_ENABLE_<TRIBITS_PACKAGE>:BOOL=OFF. This will also disable all packages that depend on <TRIBITS_PACKAGE>.

NOTE: All examples are enabled by default when setting Trilinos_ENABLE_TESTS:BOOL=ON.

NOTE: By default, setting Trilinos_ENABLE_ALL_PACKAGES=ON only enables Primary Stable Code. To have this also enable all secondary stable code, you must also you must set Trilinos_ENABLE_SECONDARY_STABLE_CODE=ON.

4. Disable a package and all its dependencies:

```
$ ./do-configure \
   -D Trilinos_ENABLE_<PACKAGE_A>:BOOL=ON \
   -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
   -D Trilinos_ENABLE_<PACKAGE_B>:BOOL=OFF
```

Above, this will enable <PACKAGE_A> and all of the packages that it depends on except for <PACKAGE_B> and all of its forward dependencies. For example, if you run:

```
$ ./do-configure \
   -D Trilinos_ENABLE_Thyra:BOOL=ON \
   -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
   -D Trilinos_ENABLE_Epetra:BOOL=OFF
```

The packages Thyra, RTOp, and Teuchos will be enabled, but the packages Epetra, EpetraExt will be disabled.

5. Removing all package enables in the Cache

```
$ ./-do-confiugre -D Trilinos_UNENABLE_ENABLED_PACKAGES:BOOL=TRUE
```

This option will set to empty '' all package enables, leaving all other cache variables as they are. You can then reconfigure with a new set of package enables for a different set of packages. This allows you to avoid more expensive configure time checks and to preserve other cache variables that you have set and don't want to loose.

4.4 1.4.4 Selecting compiler and linker options

NOTE: The Trilinos CMake build system will set up default compile options for GCC ('GNU') in development mode on order to help produce portable code.

1. Configuring to build with default debug or release compiler flags:

   To build a debug version, pass into 'cmake':

   ```
   -D CMAKE_BUILD_TYPE:STRING=DEBUG
   ```

   This will result in default debug flags getting passed to the compiler.

   To build a release (optimized) version, pass into 'cmake':

   ```
   -D CMAKE_BUILD_TYPE:STRING=RELEASE
   ```

   This will result in optimized flags getting passed to the compiler.

2. Adding arbitrary compiler flags but keeping other default flags:

To append arbitrary compiler flags that apply to all build types, configure with:

```
-DCMAKE_<LANG>_FLAGS:STRING="<EXTRA_COMPILER_OPTIONS>"
```

where <LANG> = C, CXX, Fortran and <EXTRA_COMPILER_OPTIONS> are your extra compiler options like "-DSOME_MACRO_TO_DEFINE -funroll-loops". These options will get appended to other internally defined compiler option and therefore override them.

NOTES:

1) Setting CMAKE_<LANG>_FLAGS will override but will not replace any other internally set flags in CMAKE_<LANG>_FLAGS defined by the Trilinos CMake system because these flags will come after those set internally. To get rid of these default flags, see below.

2) For each compiler type (e.g. C, C++ (CXX), Fortran), CMake passes compiler options to the compiler in the order:

```
CMAKE_<LANG>_FLAGS    CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE>
```

where <LANG> = C, CXX, or Fortran and <CMAKE_BUILD_TYPE> = DEBUG or RELEASE. THEREFORE: The options in CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE> come after and override those in CMAKE_<LANG>_FLAGS!.

3) CMake defines default CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE> values that are overridden by the Trilinos CMake build system for GCC ("GNU") compilers in development mode (e.g. Trilinos_ENABLE_DEVELOPMENT_MODE=ON). This is mostly to provide greater control over the Trilinos development environment. This means that users setting the CMAKE_<LANG>_FLAGS will *not* override the internally set debug or release flags in CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE> which come after on the compile line. Therefore, setting CMAKE_<LANG>_FLAGS should only be used for options that will not get overridden by the internally-set debug or release compiler flags in CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE>. However, setting CMAKE_<LANG>_FLAGS will work well for adding extra compiler defines (e.g. -DSOMETHING) for example.

WARNING: Any options that you set through the cache variable CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE> (where <CMAKE_BUILD_TYPE> = DEBUG or RELEASE) will get overridden in the Trilinos CMake system for GNU compilers in development mode so don't try to manually set CMAKE_<LANG>_FLAGS_<CMAKE_BUILD_TYPE>!

3. Overriding debug/release compiler options:

   To pass in compiler options that override the default debug options use:

   ```
   -D CMAKE_C_FLAGS_DEBUG_OVERRIDE:STRING="-g -O1" \
   -D CMAKE_CXX_FLAGS_DEBUG_OVERRIDE:STRING="-g -O1"
   ```

   and to override default release options use:

   ```
   -D CMAKE_C_FLAGS_RELEASE_OVERRIDE:STRING="-04 -funroll-loops" \
   -D CMAKE_CXX_FLAGS_RELEASE_OVERRIDE:STRING="-03 -fexceptions"
   ```

   NOTES: The new CMake variable CMAKE_${LANG}_FLAGS_${BUILDTYPE}_OVERRIDE is used and not CMAKE_${LANG}_FLAGS_${BUILDTYPE} because the Trilinos CMake wrappers redefine CMAKE_${LANG}_FLAGS_${BUILDTYPE} and it is impossible to determine if the value defined is determined by a user or by CMake.

4. Appending arbitrary link flags to every executable:

   In order to append any set of arbitrary link flags to your executables use:

```
-D Trilinos_EXTRA_LINK_FLAGS:STRING="$EXTRA_LINK_FLAGS"
```

Above, you can pass any type of library and they will always be the last libraries listed, even after all of the TPL.

NOTE: This is how you must set extra libraries like Fortran libraries and MPI libraries (when using raw compilers). Please only use this variable as a last resort.

NOTE: You must only pass in libraries in Trilinos_EXTRA_LINK_FLAGS and *not* arbitrary linker flags. To pass in extra linker flags that are not libraries, use the built-in CMake variable CMAKE_EXE_LINKER_FLAGS instead.

5. Turning off strong warnings for individual packages:

   To turn off strong warnings (for all langauges) for a given TriBITS package, set:

   ```
   -D <TRIBITS_PACKAGE>_DISABLE_STRONG_WARNINGS:BOOL=ON
   ```

   This will only affect the compilation of the sources for <TRIBITS_PACKAGES>, not warnings generated from the header files in downstream packages or client code.

6. Overriding all (strong warnings and debug/release) compiler options:

   To override all compiler options, including both strong warning options and debug/release options, configure with:

   ```
   -D CMAKE_C_FLAGS:STRING="-04 -funroll-loops" \
   -D CMAKE_CXX_FLAGS:STRING="-03 -fexceptions" \
   -D CMAKE_BUILD_TYPE:STRING=NONE \
   -D Trilinos_ENABLE_STRONG_C_COMPILE_WARNINGS:BOOL=OFF \
   -D Trilinos_ENABLE_STRONG_CXX_COMPILE_WARNINGS:BOOL=OFF \
   -D Trilinos_ENABLE_SHADOW_WARNINGS:BOOL=OFF \
   -D Trilinos_ENABLE_COVERAGE_TESTING:BOOL=OFF \
   -D Trilinos_ENABLE_CHECKED_STL:BOOL=OFF \
   ```

   NOTE: Options like Trilinos_ENABLE_SHADOW_WARNINGS, Trilinos_ENABLE_COVERAGE_TESTING, and Trilinos_ENABLE_CHECKED_STL do not need to be turned off by default but they are shown above to make it clear what other CMake cache variables can add compiler and link arguments.

7. Enable and disable shadowing warnings for all Trilinos packages:

   To enable shadowing warnings for all Trilinos packages (that don't already have them turned on) then use:

   ```
   -D Trilinos_ENABLE_SHADOW_WARNINGS:BOOL=ON
   ```

   To disable shadowing warnings for all Trilinos packages then use:

   ```
   -D Trilinos_ENABLE_SHADOW_WARNINGS:BOOL=OFF
   ```

   NOTE: The default value is empty '' which lets each Trilinos package decide for itself if shadowing warnings will be turned on or off for that package.

8. Removing warnings as errors for CLEANED packages:

   To remove the -Werror flag (or some other flag that is set) from being applied to compile CLEANED packages like Teuchos, set the following when configuring:

   ```
   -D Trilinos_WARNINGS_AS_ERRORS_FLAGS:STRING=""
   ```

CHAPTER

FIVE

1.5 BUILDING (MAKEFILE GENERATOR)

5.1 1.5.1 Building all targets

```
$ make [-jN]
```

(where N is the number of processes to use)

5.2 1.5.2 Discovering what targets are available to build after configuration

```
$ make help
```

5.3 1.5.3 See all of the targets to build for a package

```
$ make help | grep <TRIBITS_PACKAGE>_
```

(where <TRIBITS_PACKAGE> = Teuchos, Epetra, NOX, etc.)

or:

```
$ cd packages/<TRIBITS_PACKAGE>
$ make help
```

5.4 1.5.4 Building all of the targets for a package

```
$ make <TRIBITS_PACKAGE>_all
```

(where <TRIBITS_PACKAGE> = Teuchos, Epetra, NOX, etc.)

or:

```
$ cd packages/<TRIBITS_PACKAGE>
$ make
```

CHAPTER

SIX

1.6 TESTING WITH CTEST

6.1 1.6.1 [Recommended] Testing using *ctest*

```
$ ctest -j4
```

(see output in Testing/Temporary/LastTest.log)

NOTE: The -jN argument allows CTest to use more processes to run tests but will not exceed the max number of processes specified at configure time.

See detailed test output with:

```
$ ctest -j4 -VV
```

6.2 1.6.2 Only running tests for a single package

Running a single package test:

```
$ ctest -j4 -R '^<TRIBITS_PACKAGE>_'
```

(e.g. TRIBITS_PACKAGE = Teuchos, Epetra, etc.) (see output in Testing/Temporary/LastTest.log)

or:

```
$ cd packages/<TRIBITS_PACKAGE>
$ ctest -j4
```

Running a single test with full output to the console:

```
$ ctest -R ^FULL_TEST_NAME$ -VV
```

(e.g. FULL_TEST_NAME = Teuchos_Comm_test, Epetra_MultiVector_test, etc.)

6.3 1.6.3 Running memory checking

To run the memory tests for just a single package, from the *base* build directory, run:

```
$ ctest -R '^<TRIBITS_PACKAGE>_' -T memcheck
```

(where <TRIBITS_PACKAGE> = Epetra, NOX etc.).

(see the detailed output in ./Testing/Temporary/LastDynamicAnalysis_DATE_TIME.log)

NOTE: If you try to run memory tests from any subdirectories, that does not seem to work. You have to run them from the base build directory and then use -R '^<TRIBITS_PACKAGE>_' with ctest in order to run your packages tests.

6.4 1.6.4 Testing using `make test`

```
$ make test
```

NOTE: This is equivalent to just running 'ctest' without any arguments.

CHAPTER

SEVEN

1.7 INSTALLING

7.1 1.7.1 Setting the install prefix at configure time

```
$ ./do-configure \
   -D CMAKE_INSTALL_PREFIX:PATH=$HOME/PROJECTS/install/trilinos/mpi/opt
```

NOTE: The script 'do-configure' is just a simple shell script that calls CMake as shown above.

7.2 1.7.2 Installing after configuration

```
$ make install
```

(will build all of the targets needed before the install)

7.3 1.7.3 Uninstall

```
$ make uninstall
```

CHAPTER

EIGHT

1.8 PACKAGING

8.1 1.8.1 Creating a tarball of the source tree

```
$ make package_source
```

NOTE: The above command will tar up *everything* in the source tree (except for files explicitly excluded in the CMakeLists.txt files) so make sure that you start with a totally clean source tree before you do this. Or, you could build Doxygen documentation first and then tar up Trilinos and that would give you the source with Doxygen documentation.

NOTE: You can control what gets put into the tarball by setting the cache variable CPACK_SOURCE_IGNORE_FILES when configuring with CMake.

CHAPTER

NINE

1.9 DASHBOARD SUBMISSIONS

You can use the extended CTest scripting system in Trilinos to submit package-by-package build, test, coverage, memcheck results to the dashboard.

First, configure as normal but add the build and test parallel levels with:

```
$ ./do-configure -DCTEST_BUILD_FLAGS:STRING=-j4 \
  -DCTEST_PARALLEL_LEVEL:STRING=4 \
  [OTHER OPTIONS]
```

Then, invoke the build, test and submit with:

```
$ make dashboard
```

This invokes the advanced CTest script Trilinos/cmake/ctest/experimental_build_test.cmake to do an experimental build for all of the packages that you have explicitly enabled. The packages that are implicitly enabled due to package dependencies are not directly processed by the experimental_build_test.cmake script.

There are a number of options that you can set in the environment to control what this script does. This set of options can be found by doing:

```
$ grep 'SET_DEFAULT_AND_FROM_ENV(' \
    Trilinos/cmake/tribits/ctest/TribitsCTestDriverCore.cmake
```

Currently, this options includes:

```
Blah blah blah ...
```