FindGTest

Locate the Google C++ Testing Framework.

Imported targets

This module defines the following IMPORTED targets:

GTest::GTest
: The Google Test gtest library, if found; adds Thread::Thread automatically

GTest::Main
: The Google Test gtest_main library, if found

Result variables

This module will set the following variables in your project:

GTEST_FOUND
: Found the Google Testing framework

GTEST_INCLUDE_DIRS
: the directory containing the Google Test headers

The library variables below are set as normal variables. These contain debug/optimized keywords when a debugging library is found.

GTEST_LIBRARIES
: The Google Test gtest library; note it also requires linking with an appropriate thread library

GTEST_MAIN_LIBRARIES
: The Google Test gtest_main library

GTEST_BOTH_LIBRARIES
: Both gtest and gtest_main

Cache variables

The following cache variables may also be set:

GTEST_ROOT
: The root directory of the Google Test installation (may also be set as an environment variable)

GTEST_MSVC_SEARCH
: If compiling with MSVC, this variable can be set to MD or MT (the default) to enable searching a GTest build tree

Example usage

```
enable_testing()
find_package(GTest REQUIRED)

add_executable(foo foo.cc)
target_link_libraries(foo GTest::GTest GTest::Main)

add_test(AllTestsInFoo foo)
```

Deeper integration with CTest

If you would like each Google test to show up in CTest as a test you may use the following macro:

```
GTEST_ADD_TESTS(executable extra_args files...)
```

executable
: the path to the test executable

extra_args
: a list of extra arguments to be passed to executable enclosed in quotes (or "" for none)

files...
: a list of source files to search for tests and test fixtures. Or AUTO to find them from executable target

However, note that this macro will slow down your tests by running an executable for each test and test fixture.

Example usage:

```
set(FooTestArgs --foo 1 --bar 2)
add_executable(FooTest FooUnitTest.cc)
GTEST_ADD_TESTS(FooTest "${FooTestArgs}" AUTO)
```