Google Mock

The Google C++ mocking framework.

Overview

Google's framework for writing and using C++ mock classes. It can help you derive better designs of your system and write better tests.

It is inspired by:

- jMock,
- EasyMock, and
- Hamcrest,

and designed with C++'s specifics in mind.

Google mock:

- lets you create mock classes trivially using simple macros.
- supports a rich set of matchers and actions.
- handles unordered, partially ordered, or completely ordered expectations.
- is extensible by users.

We hope you find it useful!

Features

- Provides a declarative syntax for defining mocks.
- Can easily define partial (hybrid) mocks, which are a cross of real and mock objects.
- Handles functions of arbitrary types and overloaded functions.
- Comes with a rich set of matchers for validating function arguments.
- Uses an intuitive syntax for controlling the behavior of a mock.
- Does automatic verification of expectations (no record-and-replay needed).
- Allows arbitrary (partial) ordering constraints on function calls to be expressed,.
- Lets an user extend it by defining new matchers and actions.
- Does not use exceptions.
- Is easy to learn and use.

Please see the project page above for more information as well as the mailing list for questions, discussions, and development. There is also an IRC channel on OFTC (irc.oftc.net) #gtest available. Please join us!

Please note that code under scripts/generator is from cppclean and released under the Apache License, which is different from Google Mock's license.

Getting Started

If you are new to the project, we suggest that you read the user documentation in the following order:

- Learn the basics of Google Test, if you choose to use Google Mock with it (recommended).
- Read Google Mock for Dummies.
- Read the instructions below on how to build Google Mock.

You can also watch Zhanyong's talk on Google Mock's usage and implementation.

Once you understand the basics, check out the rest of the docs:

- CheatSheet - all the commonly used stuff at a glance.
- CookBook - recipes for getting things done, including advanced techniques.

If you need help, please check the KnownIssues and FrequentlyAskedQuestions before posting a question on the discussion group.

Using Google Mock Without Google Test

Google Mock is not a testing framework itself. Instead, it needs a testing framework for writing tests. Google Mock works seamlessly with Google Test, but you can also use it with any C++ testing framework.

Requirements for End Users

Google Mock is implemented on top of Google Test, and depends on it. You must use the bundled version of Google Test when using Google Mock.

You can also easily configure Google Mock to work with another testing framework, although it will still need Google Test. Please read "Using_Google_Mock_with_Any_Testing_Framework" for instructions.

Google Mock depends on advanced C++ features and thus requires a more modern compiler. The following are needed to use Google Mock:

Linux Requirements

- GNU-compatible Make or "gmake"
- POSIX-standard shell
- POSIX(-2) Regular Expressions (regex.h)
- C++98-standard-compliant compiler (e.g. GCC 3.4 or newer)

Windows Requirements

- Microsoft Visual C++ 8.0 SP1 or newer

Mac OS X Requirements

- Mac OS X 10.4 Tiger or newer
- Developer Tools Installed

Requirements for Contributors

We welcome patches. If you plan to contribute a patch, you need to build Google Mock and its tests, which has further requirements:

- Automake version 1.9 or newer
- Autoconf version 2.59 or newer
- Libtool / Libtoolize
- Python version 2.3 or newer (for running some of the tests and re-generating certain source files from templates)

Building Google Mock

Using CMake

If you have CMake available, it is recommended that you follow the build instructions as described for Google Test.

If are using Google Mock with an existing CMake project, the section Incorporating Into An Existing CMake Project may be of particular interest. To make it work for Google Mock you will need to change

```
target_link_libraries(example gtest_main)
```

to

```
target_link_libraries(example gmock_main)
```

This works because `gmock_main` library is compiled with Google Test. However, it does not automatically add Google Test includes. Therefore you will also have to change

```
if (CMAKE_VERSION VERSION_LESS 2.8.11)
  include_directories("${gtest_SOURCE_DIR}/include")
endif()
```

to

```
if (CMAKE_VERSION VERSION_LESS 2.8.11)
  include_directories(BEFORE SYSTEM
    "${gtest_SOURCE_DIR}/include" "${gmock_SOURCE_DIR}/include")
else()
  target_include_directories(gmock_main SYSTEM BEFORE INTERFACE
    "${gtest_SOURCE_DIR}/include" "${gmock_SOURCE_DIR}/include")
endif()
```

This will addtionally mark Google Mock includes as system, which will silence compiler warnings when compiling your tests using clang with `-Wpedantic -Wall -Wextra -Wconversion`.

Preparing to Build (Unix only)

If you are using a Unix system and plan to use the GNU Autotools build system to build Google Mock (described below), you'll need to configure it now.

To prepare the Autotools build system:

```
cd googlemock
autoreconf -fvi
```

To build Google Mock and your tests that use it, you need to tell your build system where to find its headers and source files. The exact way to do it depends on which build system you use, and is usually straightforward.

This section shows how you can integrate Google Mock into your existing build system.

Suppose you put Google Mock in directory `${GMOCK_DIR}` and Google Test in `${GTEST_DIR}` (the latter is `${GMOCK_DIR}/gtest` by default). To build Google Mock, create a library build target (or a project as called by Visual Studio and Xcode) to compile

```
${GTEST_DIR}/src/gtest-all.cc and ${GMOCK_DIR}/src/gmock-all.cc
```

with

```
${GTEST_DIR}/include and ${GMOCK_DIR}/include
```

in the system header search path, and

```
${GTEST_DIR} and ${GMOCK_DIR}
```

in the normal header search path. Assuming a Linux-like system and gcc, something like the following will do:

```
g++ -isystem ${GTEST_DIR}/include -I${GTEST_DIR} \
    -isystem ${GMOCK_DIR}/include -I${GMOCK_DIR} \
    -pthread -c ${GTEST_DIR}/src/gtest-all.cc
g++ -isystem ${GTEST_DIR}/include -I${GTEST_DIR} \
    -isystem ${GMOCK_DIR}/include -I${GMOCK_DIR} \
    -pthread -c ${GMOCK_DIR}/src/gmock-all.cc
ar -rv libgmock.a gtest-all.o gmock-all.o
```

(We need -pthread as Google Test and Google Mock use threads.)

Next, you should compile your test source file with ${GTEST_DIR}/include and ${GMOCK_DIR}/include in the header search path, and link it with gmock and any other necessary libraries:

```
g++ -isystem ${GTEST_DIR}/include -isystem ${GMOCK_DIR}/include \
    -pthread path/to/your_test.cc libgmock.a -o your_test
```

As an example, the make/ directory contains a Makefile that you can use to build Google Mock on systems where GNU make is available (e.g. Linux, Mac OS X, and Cygwin). It doesn't try to build Google Mock's own tests. Instead, it just builds the Google Mock library and a sample test. You can use it as a starting point for your own build script.

If the default settings are correct for your environment, the following commands should succeed:

```
cd ${GMOCK_DIR}/make
make
./gmock_test
```

If you see errors, try to tweak the contents of make/Makefile to make them go away.

Windows

The msvc/2005 directory contains VC++ 2005 projects and the msvc/2010 directory contains VC++ 2010 projects for building Google Mock and selected tests.

Change to the appropriate directory and run "msbuild gmock.sln" to build the library and tests (or open the gmock.sln in the MSVC IDE). If you want to create your own project to use with Google Mock, you'll have to configure it to use the `gmock_config` propety sheet. For that:

- Open the Property Manager window (View | Other Windows | Property Manager)
- Right-click on your project and select "Add Existing Property Sheet..."
- Navigate to `gmock_config.vsprops` or `gmock_config.props` and select it.
- In Project Properties | Configuration Properties | General | Additional Include Directories, type /include.

Tweaking Google Mock

Google Mock can be used in diverse environments. The default configuration may not work (or may not work well) out of the box in some environments. However, you can easily tweak Google Mock by defining control macros on the compiler command line. Generally, these macros are named like `GTEST_XYZ` and you define them to either 1 or 0 to enable or disable a certain feature.

We list the most frequently used macros below. For a complete list, see file ${GTEST_DIR}/include/gtest/internal/gtest-port.h.

Choosing a TR1 Tuple Library

Google Mock uses the C++ Technical Report 1 (TR1) tuple library heavily. Unfortunately TR1 tuple is not yet widely available with all compilers. The good news is that Google Test 1.4.0+ implements a subset of TR1 tuple that's enough for Google Mock's need. Google Mock will automatically use that implementation when the compiler doesn't provide TR1 tuple.

Usually you don't need to care about which tuple library Google Test and Google Mock use. However, if your project already uses TR1 tuple, you need to tell Google Test and Google Mock to use the same TR1 tuple library the rest of your project uses, or the two tuple implementations will clash. To do that, add

```
-DGTEST_USE_OWN_TR1_TUPLE=0
```

to the compiler flags while compiling Google Test, Google Mock, and your tests. If you want to force Google Test and Google Mock to use their own tuple library, just add

```
-DGTEST_USE_OWN_TR1_TUPLE=1
```

to the compiler flags instead.

If you want to use Boost's TR1 tuple library with Google Mock, please refer to the Boost website (http://www.boost.org/) for how to obtain it and set it up.

As a Shared Library (DLL)

Google Mock is compact, so most users can build and link it as a static library for the simplicity. Google Mock can be used as a DLL, but the same DLL must contain Google Test as well. See Google Test's README for instructions on how to set up necessary compiler settings.

Tweaking Google Mock

Most of Google Test's control macros apply to Google Mock as well. Please see Google Test's README for how to tweak them.

Upgrading from an Earlier Version

We strive to keep Google Mock releases backward compatible. Sometimes, though, we have to make some breaking changes for the users' long-term benefits. This section describes what you'll need to do if you are upgrading from an earlier version of Google Mock.

Upgrading from 1.1.0 or Earlier

You may need to explicitly enable or disable Google Test's own TR1 tuple library. See the instructions in section "Choosing a TR1 Tuple Library".

Upgrading from 1.4.0 or Earlier

On platforms where the pthread library is available, Google Test and Google Mock use it in order to be thread-safe. For this to work, you may need to tweak your compiler and/or linker flags. Please see the "Multi-threaded Tests" section in file Google Test's README for what you may need to do.

If you have custom matchers defined using `MatcherInterface` or `MakePolymorphicMatcher()`, you'll need to update their definitions to use the new matcher API (monomorphic, polymorphic). Matchers defined using `MATCHER()` or `MATCHER_P*()` aren't affected.

Developing Google Mock

This section discusses how to make your own changes to Google Mock.

Testing Google Mock Itself

To make sure your changes work as intended and don't break existing functionality, you'll want to compile and run Google Test's own tests. For that you'll need Autotools. First, make sure you have followed the instructions above to configure Google Mock. Then, create a build output directory and enter it. Next,

```
${GMOCK_DIR}/configure  # try --help for more info
```

Once you have successfully configured Google Mock, the build steps are standard for GNU-style OSS packages.

```
make        # Standard makefile following GNU conventions
make check  # Builds and runs all tests - all should pass.
```

Note that when building your project against Google Mock, you are building against Google Test as well. There is no need to configure Google Test separately.

Contributing a Patch

We welcome patches. Please read the Developer's Guide for how you can contribute. In particular, make sure you have signed the Contributor License Agreement, or we won't be able to accept the patch.

Happy testing!