MAN Scania 2020

Unit Testing Documentation

Unit Testing

Version 1.0

Release Date: 21-Dec-2016

Harman Confidential

© 2016 Harman. All rights reserved.

Revision History

Version	Author	Remarks
1.0	Kathiravan Manickavasagam	Initial Version
2.0	Tanvir Haveri	GTest and Gcov on Linux for MMT2020 HMI

© 2016 Harman. All rights reserved.

© 2016 Harman. All rights reserved.

1. Introduction

The purpose of this document is to provide inputs/ideas to prepare a HMI unit test cases using Gtest and Gmock. This document clearly explains where to start and implement the test cases. Pictorial representations are only examples and do not pertain to any design activity.

1.1 Document Scope

This document clearly explains how to create and implement HMI unit test cases. This document describes the procedure to follow to implement unit test cases using Gtest and Gmock.

1.2 Intended Audience

This document is meant for the developers and all relevant stake holders of the MAN Scania 2020 program who are involved in developing HMI.

1.3 Terminologies

Abbreviation	**Description**
MMT	Multi-Media Truck
HMI	Human Machine Interface
GTest	Google Test
GMock	Google Mock
TBD	To Be Decided

1.4 Main Author

Kathiravan Manickavasagam

© 2016 Harman. All rights reserved.

kathiravan.manickavasagam@harman.com

2. GTest

2.1 Overview

Google C++ Testing Framework helps you write better C++ tests. Why GTest?

- Tests should be independent and repeatable
- Tests should be well organized and reflect the structure of the tested code
- Tests should be portable and reusable
- When tests fail, they should provide as much information about the problem as possible
- The testing framework should liberate test writers from housekeeping chores and let them focus on the test content
- Tests should be fast

3. GMock

3.1 Overview

When you write a prototype or test, often it's not feasible or wise to rely on real objects entirely. A **mock object** implements the same interface as a real object (so it can be used as one), but lets you specify at run time how it will be used and what it should do. We create a mock object to test the behavior of some other object. An object that we want to mock has the following characteristics:

- Supplies non-deterministic results (e.g. the current time)
- Has states that are difficult to create or reproduce (e.g. a network error)
- Is slow (e.g. a complete database, which would have to be initialized before the test)
- Does not yet exist or may change behavior
- Would have to include information and methods exclusively for testing purposes (and not for its actual task)

3.2 Why Gmock?

While mock objects help you remove unnecessary dependencies in tests and make them fast and reliable, using mocks manually in C++ is hard:

- Someone has to implement the mocks. The job is usually tedious and error-prone. No wonder people go great distance to avoid it

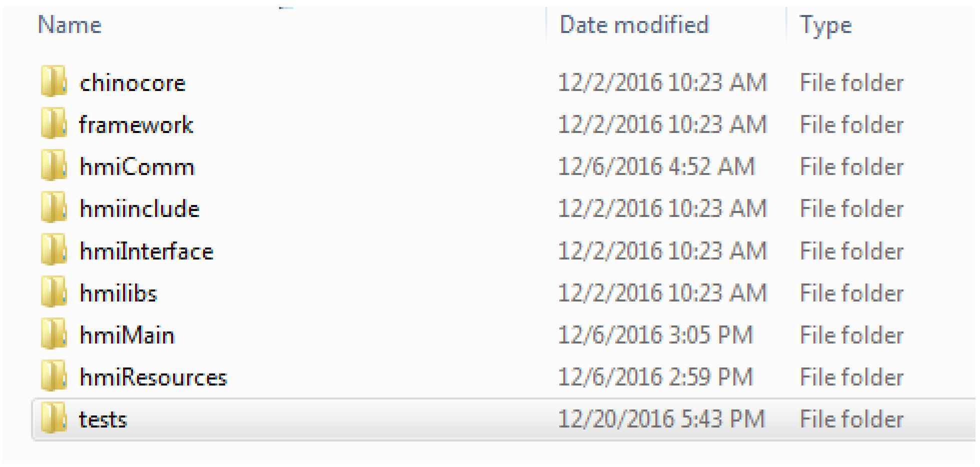
© 2016 Harman. All rights reserved.

- The quality of those manually written mocks is a bit, uh, unpredictable. You may see some really polished ones, but you may also see some that were hacked up in a hurry and have all sorts of ad hoc restrictions
- The knowledge you gained from using one mock doesn't transfer to the next

4. Gtest & Gmock in MMT

4.1 Folder Structure

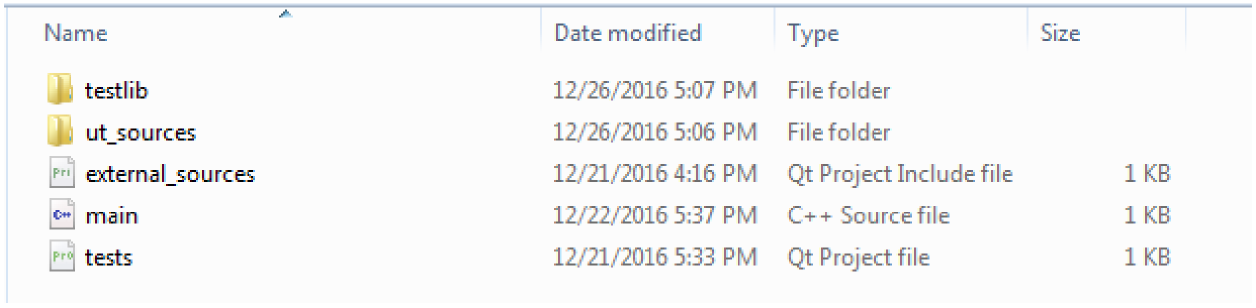
'tests' folder contains the unit test source files and gtest/gmock libraries.



Name	Date modified	Type
chinocore	12/2/2016 10:23 AM	File folder
framework	12/2/2016 10:23 AM	File folder
hmiComm	12/6/2016 4:52 AM	File folder
hmiinclude	12/2/2016 10:23 AM	File folder
hmiInterface	12/2/2016 10:23 AM	File folder
hmilibs	12/2/2016 10:23 AM	File folder
hmiMain	12/6/2016 3:05 PM	File folder
hmiResources	12/6/2016 2:59 PM	File folder
tests	12/20/2016 5:43 PM	File folder

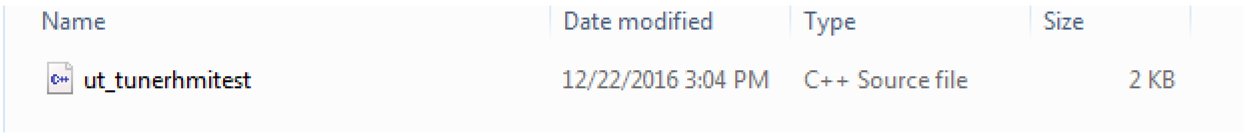
4.2 Inside 'tests' Folder

'testlib' folder contains the Gtest & Gmock libraries. 'external_sources.pri' is the project include file which has the details of source files which is included for testing.



Name	Date modified	Type	Size
testlib	12/26/2016 5:07 PM	File folder	
ut_sources	12/26/2016 5:06 PM	File folder	
external_sources	12/21/2016 4:16 PM	Qt Project Include file	1 KB
main	12/22/2016 5:37 PM	C++ Source file	1 KB
tests	12/21/2016 5:33 PM	Qt Project file	1 KB

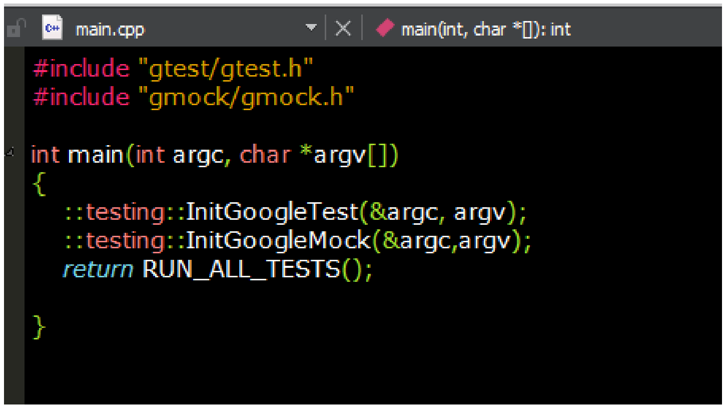
Unit test source files were placed inside 'ut_sources' folder. Each testable file will have a respective unit test file.



Name	Date modified	Type	Size
ut_tunerhmitest	12/22/2016 3:04 PM	C++ Source file	2 KB

4.3 main.cpp

This is the file initiates the Gtest and Gmock. This is used to execute all the test cases.

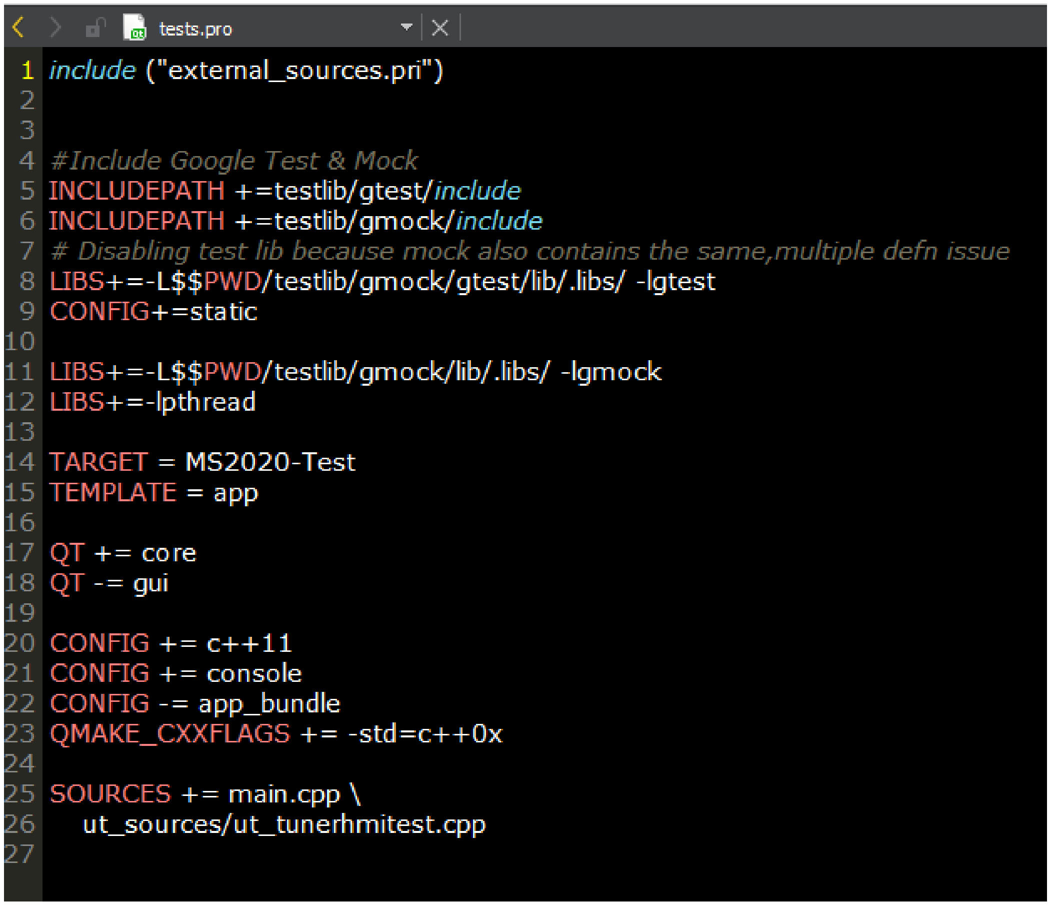


```
#include "gtest/gtest.h"
#include "gmock/gmock.h"

int main(int argc, char *argv[])
{
   ::testing::InitGoogleTest(&argc, argv);
   ::testing::InitGoogleMock(&argc,argv);
   return RUN_ALL_TESTS();

}
```

4.4 tests.pro

This is the test project file. This contains the include, configuration and other details/stuffs.

© 2016 Harman. All rights reserved.



tests.pro

```
include ("external_sources.pri")


#Include Google Test & Mock
INCLUDEPATH +=testlib/gtest/include
INCLUDEPATH +=testlib/gmock/include
# Disabling test lib because mock also contains the same,multiple defn issue
LIBS+=-L$$PWD/testlib/gmock/gtest/lib/.libs/ -lgtest
CONFIG+=static

LIBS+=-L$$PWD/testlib/gmock/lib/.libs/ -lgmock
LIBS+=-lpthread

TARGET = MS2020-Test
TEMPLATE = app

QT += core
QT -= gui

CONFIG += c++11
CONFIG += console
CONFIG -= app_bundle
QMAKE_CXXFLAGS += -std=c++0x

SOURCES += main.cpp \
    ut_sources/ut_tunerhmitest.cpp

```

4.5 Changes for running the project in Linux

The .pro and .pri file need to be changed for executing the hmiTest Project on the target hardware. This is needed since earlier gtest was being executed on windows for which the current project does not build.

- The gtest and gtest_main libraries are picked up from the elina project.
- /opt/elina/2.0.2015143A/sysroots/cortexa15hf-vfp-neon-elina-linux-gnueabi/usr/lib/libgtest.a
- Similar for gtest_main
- /opt/elina/2.0.2015143A/sysroots/cortexa15hf-vfp-neon-elina-linux-gnueabi/usr/lib/libgtest_main.a
- Both the libraries are statically linked and hence there is no need to have the libraries configured on the target for execution of the test binary.
- Note that the current elina project does not provide the gmock library which is cross compiled for execution on the target hardware.

The chino_framework.pri file must be changed to run the project in Linux. The _d should be removed so that it points to binaries that are build for linux and not windows. Below is an example:

© 2016 Harman. All rights reserved.

```
LIBS += -L$$PWD/../deploy/linux/release/lib/ -lchinocore
```

Please build the chino_core, chino_framework, hmi_comms and hmiMain Projects for the Target before building the hmiTests Project. These required changes are already done in the project uploaded on perforce in the below path:

//MAN_SCANIA_MMT_AUS5/Documentation/hmi/

4.6 Writing test cases for the Mediators

It must be noted that only mediators can be tested using Gtest. Domain files cannot be tested since the Chino:Domain class requires a chino::domain parent as its parent and this leads to a recursive dependency.

For writing test cases it is mandatory that there is some data which is saved as class members otherwise Gtest will not be able to test the resulting functionality. Also these class members should either be public or protected. This is necessary since for testing the fucnctionality of the class we need to inherit the class and then make testcases on the inherited functions.

If there are no class members which save the state after the execution of a function then the functionality may be untestable. A work around for this is to check if the class method is modifying any property that is part of the common component which is maintained as a chino::sharedptr. If this is so then we can get the property saved in that component and check whether it is as per expectation.

The creation of a component however can be tested always.

4.7 Executing the test case binary for Gtest

After writing the ut_* test files you can build them for the target after copying the hmiTest.pro and chino_framework.pri files from Perforce.

//MAN_SCANIA_MMT_AUS5/Documentation/hmi/

Once the build is successful follow the below steps:

Copy the binary(MMT2020) to the target using the below command:

```
scp MMT2020 root@<TargetIPAddress>:/opt/hmi
```

Login to the target through ssh:

```
Ssh root@<TargetIPAddress>
```

Now run the below commands to set the proper environment variables:

© 2016 Harman. All rights reserved.

export LD_LIBRARY_PATH=/opt/hmi/deploy/linux/debug/lib/:$LD_LIBRARY_PATH

export XDG_RUNTIME_DIR=/var/run/root/1000

export QT_QPA_PLATFORM=wayland

export QT_WAYLAND_DISABLE_WINDOWDECORATION=1

export FONTCONFIG_FILE=/etc/fonts/fonts.conf

export PROJECTION_TOUCH_CAPTURE=1

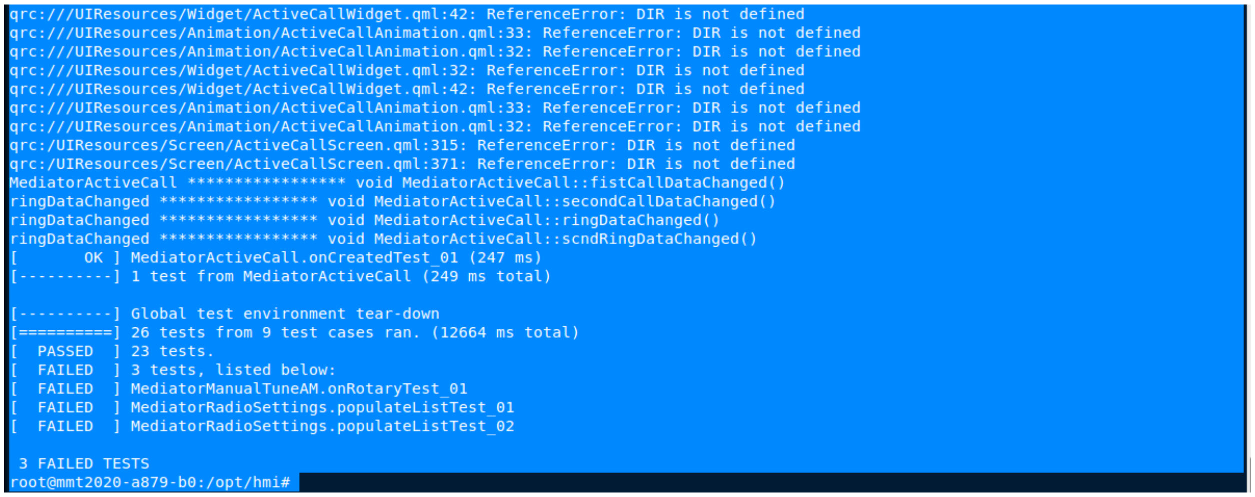
export QT_QPA_FONTDIR=/usr/share/fonts/ttf

export QT_WAYLAND_HIDE_CURSOR=0

Now run the test binary using ./:

/opt/hmi/./MMT2020

This will start executing all the test cases that were written in the ut_* files and the output can be viewed on the terminal itself as below:



```
qrc:///UIResources/Widget/ActiveCallWidget.qml:42: ReferenceError: DIR is not defined
qrc:///UIResources/Animation/ActiveCallAnimation.qml:33: ReferenceError: DIR is not defined
qrc:///UIResources/Animation/ActiveCallAnimation.qml:32: ReferenceError: DIR is not defined
qrc:///UIResources/Widget/ActiveCallWidget.qml:32: ReferenceError: DIR is not defined
qrc:///UIResources/Widget/ActiveCallWidget.qml:42: ReferenceError: DIR is not defined
qrc:///UIResources/Animation/ActiveCallAnimation.qml:33: ReferenceError: DIR is not defined
qrc:///UIResources/Animation/ActiveCallAnimation.qml:32: ReferenceError: DIR is not defined
qrc:/UIResources/Screen/ActiveCallScreen.qml:315: ReferenceError: DIR is not defined
qrc:/UIResources/Screen/ActiveCallScreen.qml:371: ReferenceError: DIR is not defined
MediatorActiveCall ***************** void MediatorActiveCall::fistCallDataChanged()
ringDataChanged ***************** void MediatorActiveCall::secondCallDataChanged()
ringDataChanged ***************** void MediatorActiveCall::ringDataChanged()
ringDataChanged ***************** void MediatorActiveCall::scndRingDataChanged()
[       OK ] MediatorActiveCall.onCreatedTest_01 (247 ms)
[----------] 1 test from MediatorActiveCall (249 ms total)

[----------] Global test environment tear-down
[==========] 26 tests from 9 test cases ran. (12664 ms total)
[  PASSED  ] 23 tests.
[  FAILED  ] 3 tests, listed below:
[  FAILED  ] MediatorManualTuneAM.onRotaryTest_01
[  FAILED  ] MediatorRadioSettings.populateListTest_01
[  FAILED  ] MediatorRadioSettings.populateListTest_02

 3 FAILED TESTS
root@mmt2020-a879-b0:/opt/hmi#
```

5. GCov Overview

Gcov is a source code test coverage analysis tool. Gcov generates exact counts of the number of times each statement in a program is executed. Gcov comes as a standard utility with the GNU Compiler Collection (GCC) suite. The gcov utility gives information on how often a program executes segments of code. The gcov utility does not produce any time-based data and works only on code compiled with the GCC suite. The options -fprofile-arcs -ftest-coverage should be used to compile the program for coverage analysis (first option to record branch statistics and second to save line execution count); -

© 2016 Harman. All rights reserved.

fprofile-arcs should also be used to link the program. After running such program will create files with .gcno extension which can be analysed by gcov.

5.1 Changes for running Gcov

The changes done for executing the Gcov to get the coverage data of the test cases is as below. The gcov is run on the code files after executing the test binary on the target and collecting the .gcda files from the target. Note that it is necessary to copy these files in the exact same directories as the .o files.

hmiTests.pro file need below changes for gCov:

```
#for gcov

LIBS+=-lgcov

#end of for gcov

#for gcov

QMAKE_CXXFLAGS += -g -Wall -fprofile-arcs -ftest-coverage

QMAKE_LFLAGS += -g -Wall -fprofile-arcs -ftest-coverage

#end of for gcov
```

These flags enable extra information to be stored in the test binary that enables the Gcov mechanism.

Compile it for target using the MMT2020 kit.

Whenever a new unit test file is added make sure that it is also added in the hmiTests.pro file in the below tag:

```
SOURCES += main.cpp \
```

5.2 Executing the test case binary for Gcov

Please ensure that you have logged in as root while executing the test executable on the target otherwise the .gcda files may not be created.

Use same steps as mentioned in section 4.4.5, the additional steps are as below:

Copy the .gcda files to the locations of the .o files. For example:

If radiomediatorsettings.o is in /home/yoctoadm/Project/MMT2020/hmi/source/build-hmiTests-MMT2020-Release/home/yoctoadm/Project/MMT2020/hmi/source/hmiMain/domains/radio then the radiomediatorsettings.gcda file should also be in the same directory.

© 2016 Harman. All rights reserved.

Also note that the .gcov files will be generated in the same directory from where the gcov command is being run which is usually the same directory as the code.

Ensure that the .gcno files are also generated during the build and their location will be in the same directory as the .o file of the respective .cpp file

For example if the .o file for radiomediatorsettings.cpp is in

/home/yoctoadm/Project/MMT2020/hmi/source/build-hmiTests-MMT2020-Release/home/yoctoadm/Project/MMT2020/hmi/source/hmiMain/domains/radio/radiomediatorsettings.o

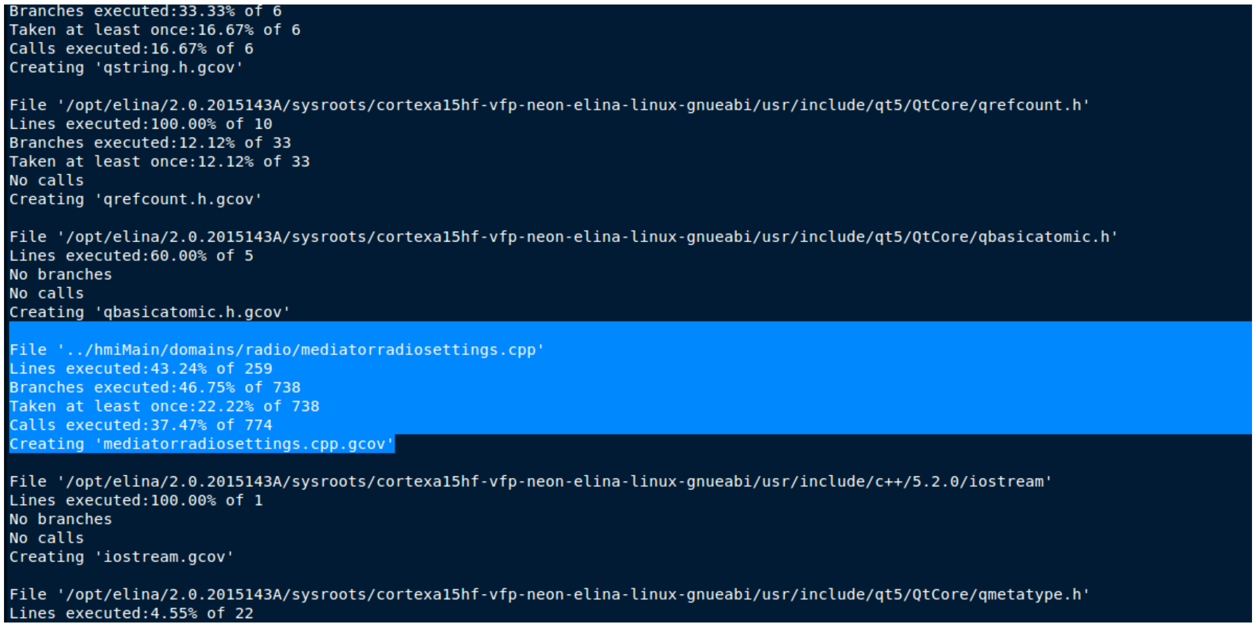
Then the radiomediatorsettings.gcno file will also be in the same location.

Now we can run the gcov command on the individual code files to check how much part of the code was executed during the test binary execution.

The example is as below:

```
yoctoadm@kickseed:~/Project/MMT2020/hmi/source/hmiMain$ gcov -a -c -b -u -f -o
/home/yoctoadm/Project/MMT2020/hmi/source/build-hmiTests-MMT2020-
Release/home/yoctoadm/Project/MMT2020/hmi/source/hmiMain/domains/radio
mediatorradiosettings.cpp
```

This will give all the details of the coverage. Example of some part is as below:



```
Branches executed:33.33% of 6
Taken at least once:16.67% of 6
Calls executed:16.67% of 6
Creating 'qstring.h.gcov'

File '/opt/elina/2.0.2015143A/sysroots/cortexa15hf-vfp-neon-elina-linux-gnueabi/usr/include/qt5/QtCore/qrefcount.h'
Lines executed:100.00% of 10
Branches executed:12.12% of 33
Taken at least once:12.12% of 33
No calls
Creating 'qrefcount.h.gcov'

File '/opt/elina/2.0.2015143A/sysroots/cortexa15hf-vfp-neon-elina-linux-gnueabi/usr/include/qt5/QtCore/qbasicatomic.h'
Lines executed:60.00% of 5
No branches
No calls
Creating 'qbasicatomic.h.gcov'

File '../hmiMain/domains/radio/mediatorradiosettings.cpp'
Lines executed:43.24% of 259
Branches executed:46.75% of 738
Taken at least once:22.22% of 738
Calls executed:37.47% of 774
Creating 'mediatorradiosettings.cpp.gcov'

File '/opt/elina/2.0.2015143A/sysroots/cortexa15hf-vfp-neon-elina-linux-gnueabi/usr/include/c++/5.2.0/iostream'
Lines executed:100.00% of 1
No branches
No calls
Creating 'iostream.gcov'

File '/opt/elina/2.0.2015143A/sysroots/cortexa15hf-vfp-neon-elina-linux-gnueabi/usr/include/qt5/QtCore/qmetatype.h'
Lines executed:4.55% of 22
```

© 2016 Harman. All rights reserved.

Ensure that the .gcda, .gcno and .o files should be in one location. Then run gcov from the location of the source files by giving the -o option to specify the location of the .o, .gcda and .gcno files.

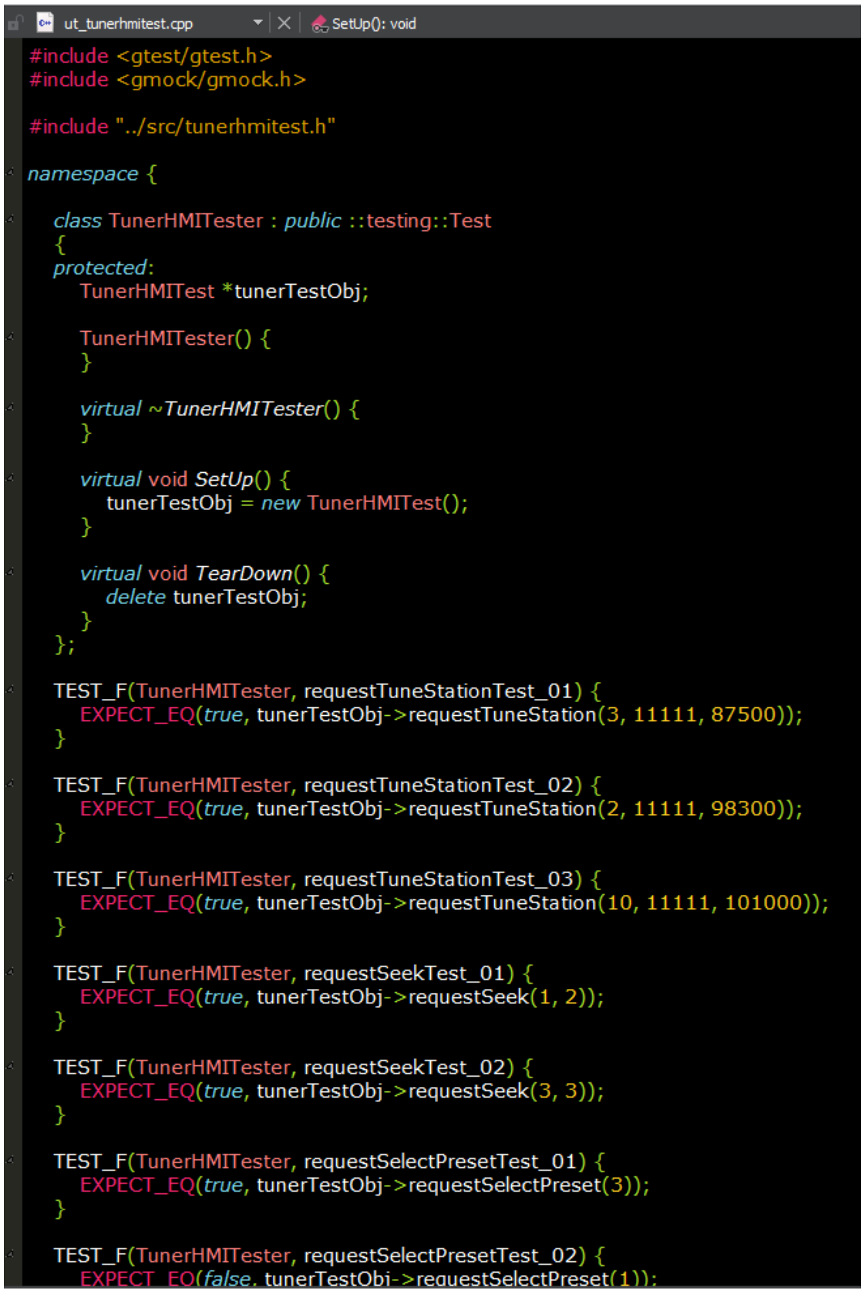
6. Sample Testing

This is sample test executed on tuner HMI API calls. This contains few functions like station tune, preset tune, tune seek, etc.,

6.1 ut_tunerhmitest.cpp

Below is the sample code with few test cases.

© 2016 Harman. All rights reserved.

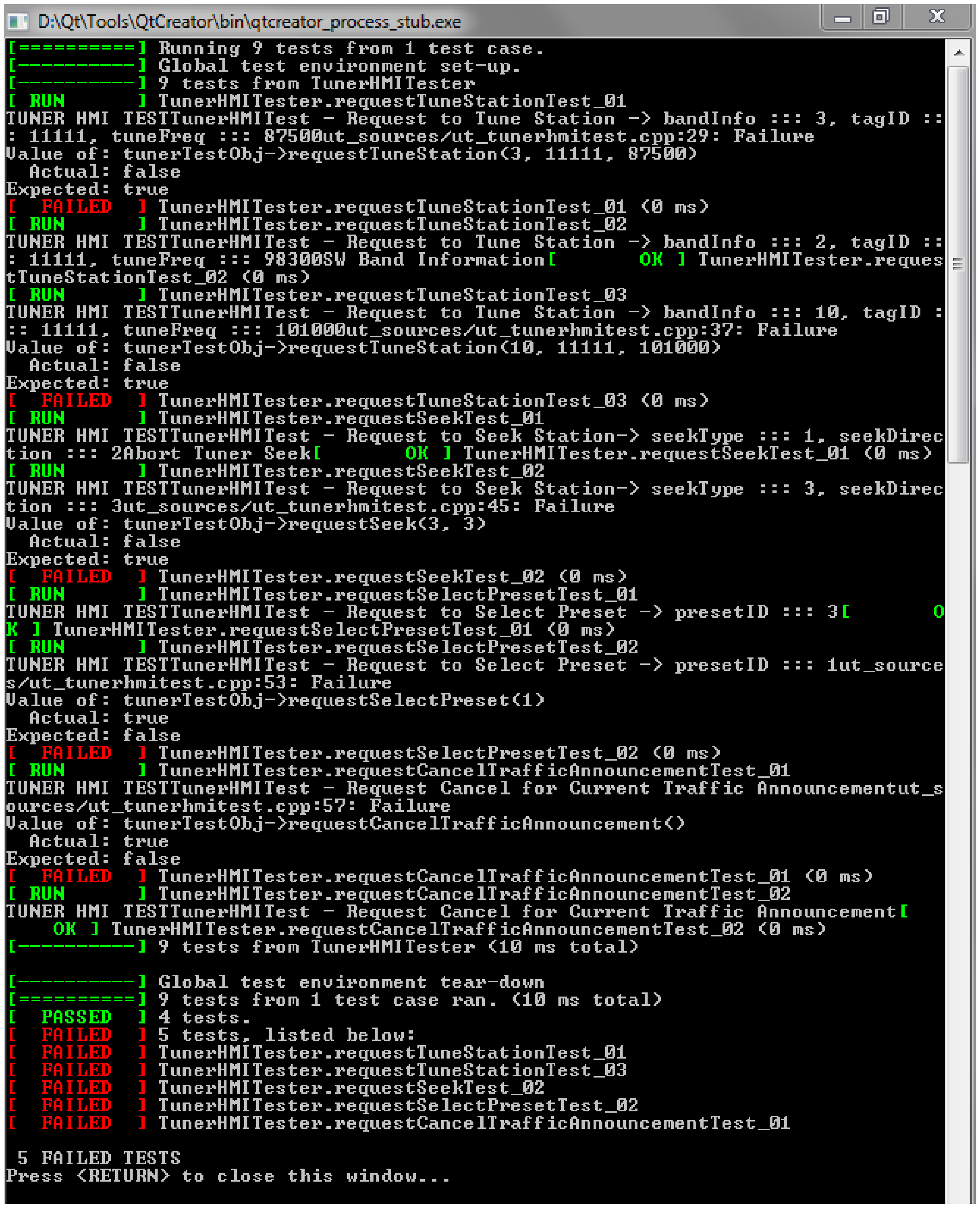


```
#include <gtest/gtest.h>
#include <gmock/gmock.h>

#include "../src/tunerhmitest.h"

namespace {

  class TunerHMITester : public ::testing::Test
  {
  protected:
    TunerHMITest *tunerTestObj;

    TunerHMITester() {
    }

    virtual ~TunerHMITester() {
    }

    virtual void SetUp() {
      tunerTestObj = new TunerHMITest();
    }

    virtual void TearDown() {
      delete tunerTestObj;
    }
  };

  TEST_F(TunerHMITester, requestTuneStationTest_01) {
    EXPECT_EQ(true, tunerTestObj->requestTuneStation(3, 11111, 87500));
  }

  TEST_F(TunerHMITester, requestTuneStationTest_02) {
    EXPECT_EQ(true, tunerTestObj->requestTuneStation(2, 11111, 98300));
  }

  TEST_F(TunerHMITester, requestTuneStationTest_03) {
    EXPECT_EQ(true, tunerTestObj->requestTuneStation(10, 11111, 101000));
  }

  TEST_F(TunerHMITester, requestSeekTest_01) {
    EXPECT_EQ(true, tunerTestObj->requestSeek(1, 2));
  }

  TEST_F(TunerHMITester, requestSeekTest_02) {
    EXPECT_EQ(true, tunerTestObj->requestSeek(3, 3));
  }

  TEST_F(TunerHMITester, requestSelectPresetTest_01) {
    EXPECT_EQ(true, tunerTestObj->requestSelectPreset(3));
  }

  TEST_F(TunerHMITester, requestSelectPresetTest_02) {
    EXPECT_EQ(false, tunerTestObj->requestSelectPreset(1));
```

6.2 Results

The results are seen in the console as below. The passed and failed test cases are captured.

© 2016 Harman. All rights reserved.


D:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
[==========] Running 9 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 9 tests from TunerHMITester
[ RUN      ] TunerHMITester.requestTuneStationTest_01
TUNER HMI TESTTunerHMITest - Request to Tune Station -> bandInfo ::: 3, tagID ::
: 11111, tuneFreq ::: 87500ut_sources/ut_tunerhmitest.cpp:29: Failure
Value of: tunerTestObj->requestTuneStation(3, 11111, 87500)
  Actual: false
Expected: true
[  FAILED  ] TunerHMITester.requestTuneStationTest_01 (0 ms)
[ RUN      ] TunerHMITester.requestTuneStationTest_02
TUNER HMI TESTTunerHMITest - Request to Tune Station -> bandInfo ::: 2, tagID ::
: 11111, tuneFreq ::: 98300SW Band Information[       OK ] TunerHMITester.reques
tTuneStationTest_02 (0 ms)
[ RUN      ] TunerHMITester.requestTuneStationTest_03
TUNER HMI TESTTunerHMITest - Request to Tune Station -> bandInfo ::: 10, tagID :
:: 11111, tuneFreq ::: 101000ut_sources/ut_tunerhmitest.cpp:37: Failure
Value of: tunerTestObj->requestTuneStation(10, 11111, 101000)
  Actual: false
Expected: true
[  FAILED  ] TunerHMITester.requestTuneStationTest_03 (0 ms)
[ RUN      ] TunerHMITester.requestSeekTest_01
TUNER HMI TESTTunerHMITest - Request to Seek Station-> seekType ::: 1, seekDirec
tion ::: 2Abort Tuner Seek[       OK ] TunerHMITester.requestSeekTest_01 (0 ms)
[ RUN      ] TunerHMITester.requestSeekTest_02
TUNER HMI TESTTunerHMITest - Request to Seek Station-> seekType ::: 3, seekDirec
tion ::: 3ut_sources/ut_tunerhmitest.cpp:45: Failure
Value of: tunerTestObj->requestSeek(3, 3)
  Actual: false
Expected: true
[  FAILED  ] TunerHMITester.requestSeekTest_02 (0 ms)
[ RUN      ] TunerHMITester.requestSelectPresetTest_01
TUNER HMI TESTTunerHMITest - Request to Select Preset -> presetID ::: 3[       O
K ] TunerHMITester.requestSelectPresetTest_01 (0 ms)
[ RUN      ] TunerHMITester.requestSelectPresetTest_02
TUNER HMI TESTTunerHMITest - Request to Select Preset -> presetID ::: 1ut_source
s/ut_tunerhmitest.cpp:53: Failure
Value of: tunerTestObj->requestSelectPreset(1)
  Actual: true
Expected: false
[  FAILED  ] TunerHMITester.requestSelectPresetTest_02 (0 ms)
[ RUN      ] TunerHMITester.requestCancelTrafficAnnouncementTest_01
TUNER HMI TESTTunerHMITest - Request Cancel for Current Traffic Announcementut_s
ources/ut_tunerhmitest.cpp:57: Failure
Value of: tunerTestObj->requestCancelTrafficAnnouncement()
  Actual: true
Expected: false
[  FAILED  ] TunerHMITester.requestCancelTrafficAnnouncementTest_01 (0 ms)
[ RUN      ] TunerHMITester.requestCancelTrafficAnnouncementTest_02
TUNER HMI TESTTunerHMITest - Request Cancel for Current Traffic Announcement[
    OK ] TunerHMITester.requestCancelTrafficAnnouncementTest_02 (0 ms)
[----------] 9 tests from TunerHMITester (10 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 1 test case ran. (10 ms total)
[  PASSED  ] 4 tests.
[  FAILED  ] 5 tests, listed below:
[  FAILED  ] TunerHMITester.requestTuneStationTest_01
[  FAILED  ] TunerHMITester.requestTuneStationTest_03
[  FAILED  ] TunerHMITester.requestSeekTest_02
[  FAILED  ] TunerHMITester.requestSelectPresetTest_02
[  FAILED  ] TunerHMITester.requestCancelTrafficAnnouncementTest_01

 5 FAILED TESTS
Press <RETURN> to close this window...
```

© 2016 Harman. All rights reserved.