///////////////////////////////////////////////////////////////////////////////

/// GtFixCallsTests.h : Google Unit tests for FixCalls

/// 8/18/2017

/// Inspired by TestFixCalls.cpp

#ifndef \_\_GTFIXCALLSTESTS\_H\_\_

#define \_\_GTFIXCALLSTESTS\_H\_\_

#include "stdafx.h"

#pragma warning( disable : 4251 4996 )

#include "UnityFixFunctions.h"

namespace fixcallstests

{

///////////////////////////////////////////////////////////////////////////////

/// Helper macros

#define FIXTEST(val1, val2) TEST(val1, val2) { val2(); }

///////////////////////////////////////////////////////////////////////////////

} // namespace fixcallstests

#endif // \_\_GTFIXCALLSTESTS\_H\_\_

///////////////////////////////////////////////////////////////////////////////

/// GtFixCallsTestApi.h : Google Unit tests for FixCalls

/// 8/18/2017

/// Inspired by TestFixCalls.cpp

#ifndef \_\_GTFIXCALLSTESTAPI\_H\_\_

#define \_\_GTFIXCALLSTESTAPI\_H\_\_

#include "stdafx.h"

#pragma warning( disable : 4251 4996 )

#include <iostream>

#include "gtest\gtest.h"

#include "FixReferences.h"

namespace fixcallstestapi

{

///////////////////////////////////////////////////////////////////////////////

/// Globals

extern bool isAssert;

///////////////////////////////////////////////////////////////////////////////

/// Helper macros

#define DUMMY\_ARC\_ARGV int ac = 1; int\* pac = &ac; char av = 'T'; \

char\* pav = &av; char\*\* ppav = &pav;

#define FIXCONDITIONASSERT(type, isStrict, condition, expr) if(isStrict) \

{ ASSERT\_##type((condition), (expr)); } \

else { EXPECT\_##type((condition), (expr)); }

#define FIXASSERT(type, condition, expr) if(fixcallstestapi::isAssert) \

{ ASSERT\_##type((condition), (expr)); } \

else { EXPECT\_##type((condition), (expr)); }

///////////////////////////////////////////////////////////////////////////////

/// Helper functions

/\* TODO My be not neccessary just modeling function

void getFIRateCurve();

\*/

FixYcFullBase getFIRateCurve();

FixIRConventions getIRConventions(const std::string& objectName);

FixYcFullBase getFixYcBaseMake(const std::string& objectName);

///////////////////////////////////////////////////////////////////////////////

/// Tests

void runFixVersion();

void runFixDaycount();

void runFixDate();

void runFixVariantSize();

void runFixVariantInt();

void runFixVariantDouble();

void runFixVariantLong();

void runFixVariantBool();

void runFixVariantString();

void runFixVariantDate();

void runFixVariant();

void runFixAddTenor();

void runFixRefereceFixVaraint();

void runFixRefereceReset();

void runFixVolCalendarMake();

void runBonds();

void runFixYcDiscreteMake();

void runFixFastCalls();

void runBlackModel();

void runEnum();

void runFixInterpLiborDates();

void runDefaultFixAddTenor();

void runFixDF();

///////////////////////////////////////////////////////////////////////////////

} // namespace fixcallstestapi

#endif // \_\_GTFIXCALLSTESTAPI\_H\_\_

///////////////////////////////////////////////////////////////////////////////

/// GtFixCallsMemTests.h : Google Unit tests for testing memory routine

/// 8/18/2017

/// Inspired by TestFixCalls.cpp

#ifndef \_\_GTFIXCALLSMEMTESTS\_H\_\_

#define \_\_GTFIXCALLSMEMTESTS\_H\_\_

#include "stdafx.h"

#pragma warning( disable : 4251 4996 )

#ifdef \_DEBUG

#ifndef DBG\_NEW

#define DBG\_NEW new( \_NORMAL\_BLOCK , \_\_FILE\_\_ , \_\_LINE\_\_ )

#define new DBG\_NEW

#endif

#endif // \_DEBUG

namespace fixcallsmemtests

{

///////////////////////////////////////////////////////////////////////////////

/// Helper macros

#ifdef \_DEBUG

#define MEMTEST\_SET\_FLAGS \_CrtSetDbgFlag ( \_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF ); \

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE); \

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT); \

\_CrtSetReportMode(\_CRT\_ERROR, \_CRTDBG\_MODE\_FILE); \

\_CrtSetReportFile(\_CRT\_ERROR, \_CRTDBG\_FILE\_STDOUT); \

\_CrtSetReportMode(\_CRT\_ASSERT, \_CRTDBG\_MODE\_FILE); \

\_CrtSetReportFile(\_CRT\_ASSERT, \_CRTDBG\_FILE\_STDOUT);

#define MEMTEST\_VARS \_CrtMemState s1, s2, s3;

#define MEMTEST\_RUN(func) std::cout << std::string(80, '-') << std::endl \

<< "=> Memory test function: " << #func << std::endl; \

\_CrtMemCheckpoint((&s1)); func##(); \_CrtMemCheckpoint((&s2)); \

if ( \_CrtMemDifference( &s3, &s1, &s2) ) \

{ \

std::cout << "=> Memory leaks for the function: " << #func << std::endl \

<< "file name: " << \_\_FILE\_\_ << " line: " << \_\_LINE\_\_ << std::endl << std::endl; \

\_CrtMemDumpStatistics( &s3 ); \

} \

else \

{ \

std::cout << "=> Test Ok" << std::endl; \

} \

memset(&s1, 0, sizeof(\_CrtMemState)); \

memset(&s2, 0, sizeof(\_CrtMemState)); \

memset(&s3, 0, sizeof(\_CrtMemState));

#endif

///////////////////////////////////////////////////////////////////////////////

/// Functions

void runMemTests();

///////////////////////////////////////////////////////////////////////////////

} // namespace fixcallsmemtests

#endif // \_\_GTFIXCALLSMEMTESTS\_H\_\_

///////////////////////////////////////////////////////////////////////////////

/// GtFixCalls.cpp : Google Unit tests for FixCalls

/// 8/17/2017

/// Inspired by TestFixCalls.cpp

#include "stdafx.h"

#include "GtFixCallsTests.h"

#include "GtFixCallsMemTests.h"

#include "GtFixCallsTestApi.h"

using namespace fixcallsmemtests;

using namespace fixcallstests;

///////////////////////////////////////////////////////////////////////////////

/// MAIN

int \_tmain(int argc, \_TCHAR\* argv[])

{

fixcallstestapi::isAssert = false; //< use EXPECT

::testing::GTEST\_FLAG(output) = "xml:gtestlog\_SampleGtest.xml";

DUMMY\_ARC\_ARGV

::testing::InitGoogleTest(pac, ppav);

RUN\_ALL\_TESTS();

std::cout << std::string(80, '=') << std::endl

<< "| Begin memory tests" << std::endl

<< std::string(80, '=') << std::endl;

fixcallstestapi::isAssert = true; //< use ASSERT

runMemTests();

std::cout << std::string(80, '=') << std::endl

<< "| Begin memory leaks report" << std::endl

<< std::string(80, '=') << std::endl;

\_CrtDumpMemoryLeaks();

return 0;

}

///////////////////////////////////////////////////////////////////////////////

/// GtFixCallsMemTests.cpp : Google Unit tests for testing memory routine

/// 8/18/2017

/// Inspired by TestFixCalls.cpp

#include "stdafx.h"

#include <crtdbg.h>

#include <iostream>

#include "gtest\gtest.h"

#include "GtFixCallsTestApi.h"

#include "GtFixCallsMemTests.h"

namespace fixcallsmemtests

{

using namespace fixcallstestapi;

///////////////////////////////////////////////////////////////////////////////

///

void runMemTests()

{

MEMTEST\_SET\_FLAGS

MEMTEST\_VARS

MEMTEST\_RUN(runFixVersion)

MEMTEST\_RUN(runFixDaycount)

MEMTEST\_RUN(runFixDate)

MEMTEST\_RUN(runFixVariantSize)

MEMTEST\_RUN(runFixVariantInt)

MEMTEST\_RUN(runFixVariantDouble)

MEMTEST\_RUN(runFixVariantLong)

MEMTEST\_RUN(runFixVariantBool)

MEMTEST\_RUN(runFixVariantString)

MEMTEST\_RUN(runFixVariantDate)

MEMTEST\_RUN(runFixVariant)

MEMTEST\_RUN(runFixAddTenor)

MEMTEST\_RUN(runFixRefereceFixVaraint)

MEMTEST\_RUN(runFixRefereceReset)

MEMTEST\_RUN(runFixYcDiscreteMake)

MEMTEST\_RUN(runFixVolCalendarMake)

MEMTEST\_RUN(runBonds)

MEMTEST\_RUN(runFixYcDiscreteMake)

MEMTEST\_RUN(runFixFastCalls)

MEMTEST\_RUN(runBlackModel)

MEMTEST\_RUN(runEnum)

MEMTEST\_RUN(runFixInterpLiborDates)

MEMTEST\_RUN(runDefaultFixAddTenor)

MEMTEST\_RUN(runFixDF)

}

///////////////////////////////////////////////////////////////////////////////

} // namespace fixcallsmemtests

///////////////////////////////////////////////////////////////////////////////

/// GtFixCallsTests.cpp : Google Unit tests for FixCalls

/// 8/18/2017

/// Inspired by TestFixCalls.cpp

#include "stdafx.h"

#include "gtest\gtest.h"

#include "GtFixCallsTests.h"

#include "GtFixCallsTestApi.h"

namespace fixcallstests

{

using namespace fixcallstestapi;

///////////////////////////////////////////////////////////////////////////////

/// TESTS

FIXTEST(FixCalls, runFixVersion)

FIXTEST(FixCalls, runFixDaycount)

FIXTEST(FixCalls, runFixDate)

FIXTEST(FixCalls, runFixVariantSize)

FIXTEST(FixCalls, runFixVariantInt)

FIXTEST(FixCalls, runFixVariantDouble)

FIXTEST(FixCalls, runFixVariantLong)

FIXTEST(FixCalls, runFixVariantBool)

FIXTEST(FixCalls, runFixVariantString)

FIXTEST(FixCalls, runFixVariantDate)

FIXTEST(FixCalls, runFixVariant)

FIXTEST(FixCalls, runFixAddTenor)

FIXTEST(FixCalls, runFixRefereceFixVaraint)

FIXTEST(FixCalls, runFixRefereceReset)

FIXTEST(FixCalls, runFixVolCalendarMake)

FIXTEST(FixCalls, runBonds)

/\* TODO My be not neccessary just modeling function

FIXTEST(FixCalls, getFIRateCurve)

\*/

FIXTEST(FixCalls, runFixYcDiscreteMake)

FIXTEST(FixCalls, runFixFastCalls)

FIXTEST(FixCalls, runBlackModel)

FIXTEST(FixCalls, runEnum)

FIXTEST(FixCalls, runFixInterpLiborDates)

FIXTEST(FixCalls, runDefaultFixAddTenor)

FIXTEST(FixCalls, runFixDF)

///////////////////////////////////////////////////////////////////////////////

} // namespace fixcallstests

// unityfixgtest.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include "gtest\gtest.h"

#define DUMMY\_ARC\_ARGV int ac = 1; int\* pac = &ac; char av = 'T'; \

char\* pav = &av; char\*\* ppav = &pav;

int \_tmain(int argc, \_TCHAR\* argv[])

{

#ifdef GTEST\_RUN

::testing::GTEST\_FLAG(output) = "xml:gtestlog\_SampleGtest.xml";

DUMMY\_ARC\_ARGV

::testing::InitGoogleTest(pac, ppav);

RUN\_ALL\_TESTS();

std::cout << "All tests are completed" << std::endl;

return 0;

#endif

std::cout << "Regular program run" << std::endl;

return 0;

}

#include "stdafx.h"

#include <string>

#include <vector>

#include "FixBaseFunctions.h"

#include "gtest\gtest.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SHORT MEMO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ASSERTIONS

https://github.com/google/googletest/blob/master/googletest/docs/Primer.md

Fatal assertion

ASSERT\_TRUE( condition ) ; condition is true

ASSERT\_FALSE( condition ) ; condition is false

Nonfatal assertion

EXPECT\_TRUE( condition ) ; condition is true

EXPECT\_FALSE( condition ) ; condition is false

---------------------------------------------------------------

BINARY COMPARISION

Fatal assertion

ASSERT\_EQ( val1 , val2 ); val1 == val2

ASSERT\_NE( val1 , val2 ); val1 != val2

ASSERT\_LT( val1 , val2 ); val1 < val2

ASSERT\_LE( val1 , val2 ); val1 <= val2

ASSERT\_GT( val1 , val2 ); val1 > val2

ASSERT\_GE( val1 , val2 ); val1 >= val2

Nonfatal assertion

EXPECT\_EQ( val1 , val2 ); val1 == val2

EXPECT\_NE( val1 , val2 ); val1 != val2

EXPECT\_LT( val1 , val2 ); val1 < val2

EXPECT\_LE( val1 , val2 ); val1 <= val2

EXPECT\_GT( val1 , val2 ); val1 > val2

EXPECT\_GE( val1 , val2 ); val1 >= val2

---------------------------------------------------------------

STRING COMPARISON

The assertions in this group compare two C strings.

If you want to compare two string objects,

use EXPECT\_EQ , EXPECT\_NE , and etc instead.

Fatal assertion

ASSERT\_STREQ( str1 , str2 ); the two C strings have the same content

ASSERT\_STRNE( str1 , str2 ); the two C strings have different content

ASSERT\_STRCASEEQ( str1 , str2 ); the two C strings have the same content, ignoring case

ASSERT\_STRCASENE( str1 , str2 ); the two C strings have different content, ignoring case

Nonfatal assertion

EXPECT\_STREQ( str1 , \_str\_2 ); the two C strings have the same content

EXPECT\_STRNE( str1 , str2 ); the two C strings have different content

EXPECT\_STRCASEEQ( str1 , str2 ); the two C strings have the same content, ignoring case

EXPECT\_STRCASENE( str1 , str2 ); the two C strings have different content, ignoring case

Note that "CASE" in an assertion name means that case is ignored.

\*STREQ\* and \*STRNE\* also accept wide C strings ( wchar\_t\* ).

If a comparison of two wide strings fails, their values will be printed as UTF-8 narrow strings.

A NULL pointer and an empty string are considered different.

\*/

///////////////////////////////////////////////////////////////////////////////

// TESTS

TEST(getNameList, Zero)

{

std::string objName = "objName";

std::string clsName = "clsName";

// This function to be tested

//bool fixRepositoryFree(const std::string& objName,const std::string& clsName);

ASSERT\_FALSE(fixRepositoryFree(objName, clsName));

EXPECT\_EQ(true, fixRepositoryFree(objName, clsName));

}

[Building Google Test and Google Mock for native C++](http://wfs-confluence.wellsfargo.com:8090/pages/viewpage.action?pageId=270500480)

[Skip to end of metadata](http://wfs-confluence.wellsfargo.com:8090/pages/viewpage.action?pageId=270500480#page-metadata-end)

* Created by [Budantsev, Maxim](http://wfs-confluence.wellsfargo.com:8090/display/~U595365), last modified on [Sep 12, 2017](http://wfs-confluence.wellsfargo.com:8090/pages/diffpagesbyversion.action?pageId=270500480&selectedPageVersions=6&selectedPageVersions=7)

[Go to start of metadata](http://wfs-confluence.wellsfargo.com:8090/pages/viewpage.action?pageId=270500480#page-metadata-start)

This guide describe how to build Google Test and Google Mock libraries to use it for C++ native code unit tests.

Google's framework for writing C++ tests on a variety of platforms (Linux, Mac OS X, Windows, Cygwin, Windows CE, and Symbian). Based on the xUnit architecture. Supports automatic test discovery, a rich set of assertions, user-defined assertions, death tests, fatal and non-fatal failures, value- and type-parameterized tests, various options for running the tests, and XML test report generation.

Getting StartedAfter downloading Google Test, unpack it, read the README file and the documentation wiki pages (listed on the right side of this front page). Who Is Using Google Test?In addition to many internal projects at Google, Google Test is also used by the following notable projects: The Chromium projects (behind the Chrome browser and Chrome OS) The LLVM compiler Protocol Buffers (Google's data interchange format)

If you know of a project that's using Google Test and want it to be listed here, please let [googletestframework@googlegroups.com](mailto:googletestframework@googlegroups.com) know. Google Test-related open source projectsGoogle Test UI is test runner that runs your test binary, allows you to track its progress via a progress bar, and displays a list of test failures. Clicking on one shows failure text. Google Test UI is written in C#. GTest TAP Listener is an event listener for Google Test that implements the TAP protocol for test result output. If your test runner understands TAP, you may find it useful.

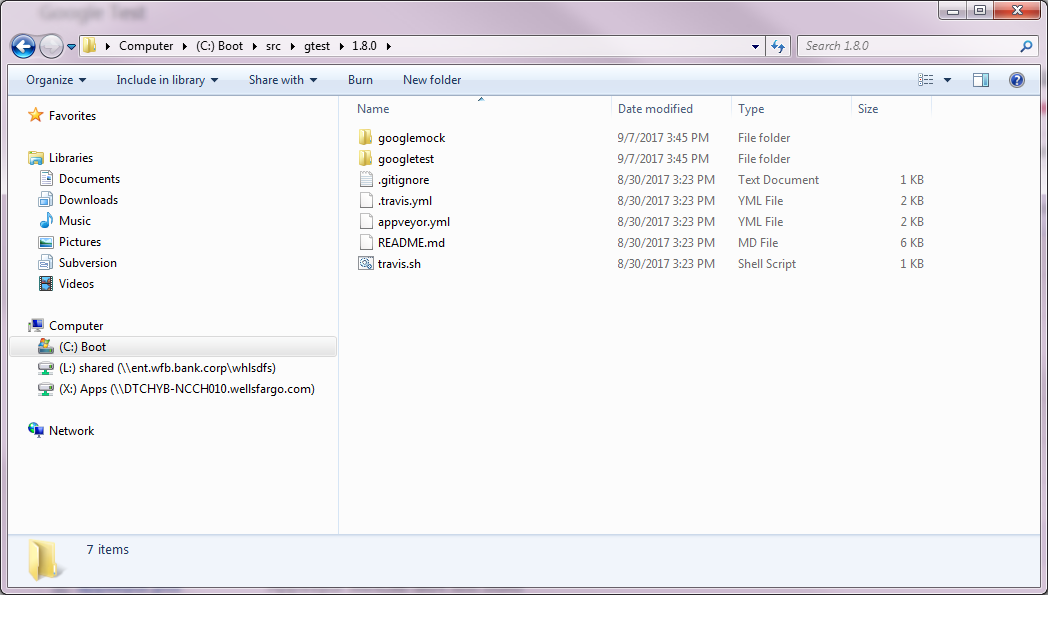
Step-by-step guide for Windows 7 64bit

Download the source code from

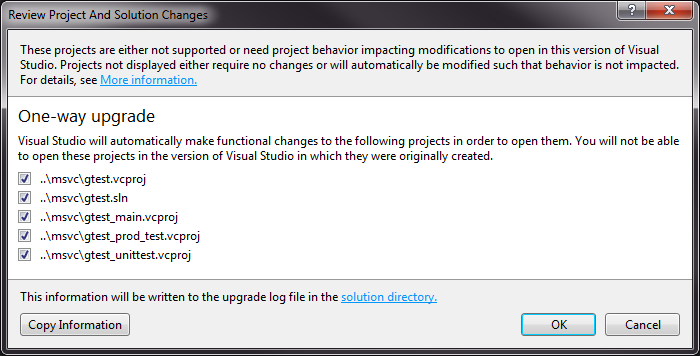
<https://cppra00a0114.wellsfargo.com:8112/oer/index.jsp> (09/07/2017 version 1.8.0 is available)

or from original source <https://github.com/google/googletest/> in local directory (e.g. C:\src\gtest\)

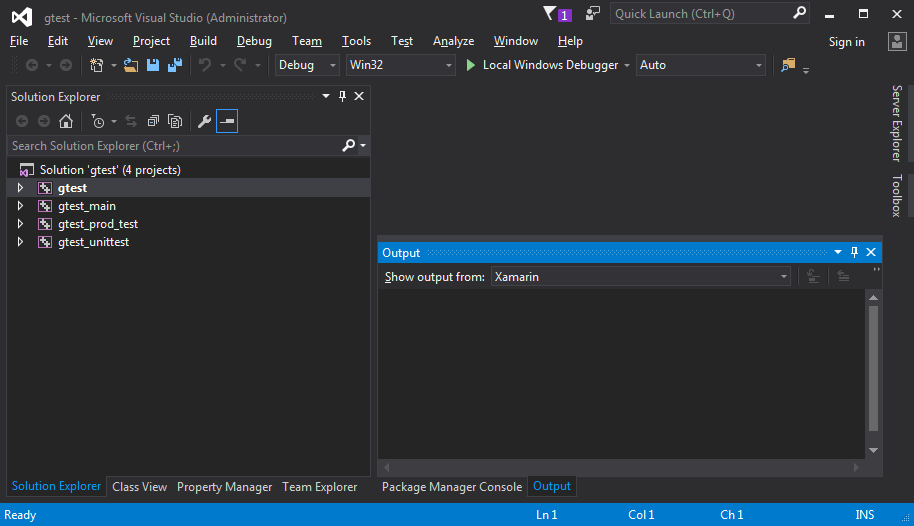
Place the file googletest-release-1.8.0.zip into the folder c:\src\gtest and unpack zip archive. Delete the .zip file. After that rename the directory googletest-release-1.8.0 to 1.8.0. Such a source tree should be on the local computer after all of these manipulations.



Go to the C:\src\gtest\1.8.0\googletest\msvc, right click on the gtest.sln and Open with Visual studio 2015. VS2015/VS2012 asks to one-way upgrade  - press OK.



This is a source tree in the Visual Studio 2015 IDE



Right click on the gtest and choose Build. On the VS2015 it should be built ok but two warnings are issued

1>------ Build started: Project: gtest, Configuration: Debug Win32 ------

1>  [gtest-all.cc](http://gtest-all.cc/)

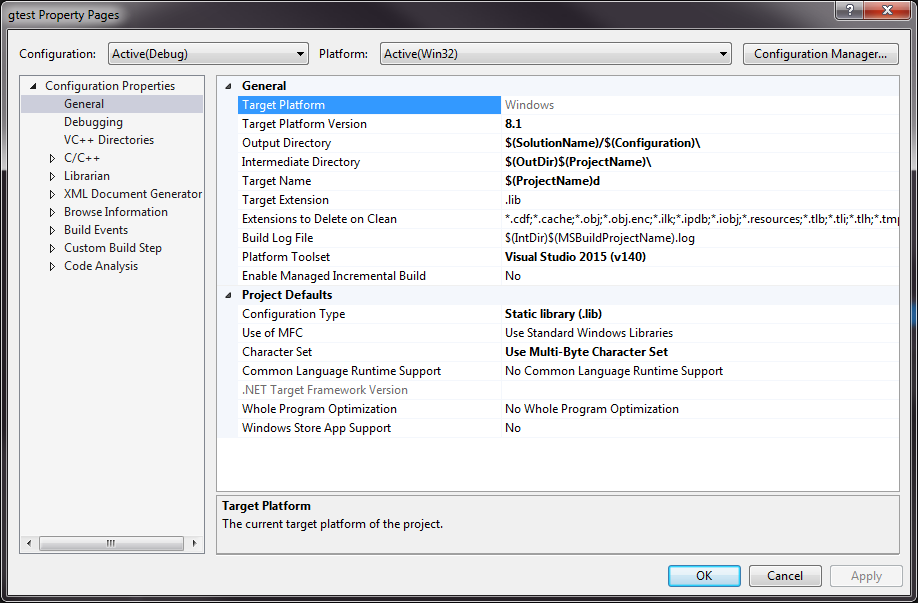
1>C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140\Microsoft.CppBuild.targets(1357,5): warning MSB8012: TargetPath(C:\src\gtest\1.8.0\googletest\msvc\gtest/Debug\gtest.lib) does not match the Library's OutputFile property value (C:\src\gtest\1.8.0\googletest\msvc\gtest\Debug\gtestd.lib). This may cause your project to build incorrectly. To correct this, please make sure that $(OutDir), $(TargetName) and $(TargetExt) property values match the value specified in %(Lib.OutputFile).

1>C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140\Microsoft.CppBuild.targets(1359,5): warning MSB8012: TargetName(gtest) does not match the Library's OutputFile property value (gtestd). This may cause your project to build incorrectly. To correct this, please make sure that $(OutDir), $(TargetName) and $(TargetExt) property values match the value specified in %(Lib.OutputFile).

1>  gtest.vcxproj -> C:\src\gtest\1.8.0\googletest\msvc\gtest/Debug\gtest.lib

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

To fix it right click on the gtest and choose Properties and add **d** symbol to the General->Target Name



After that rebuild the library and no warnings should be issued.

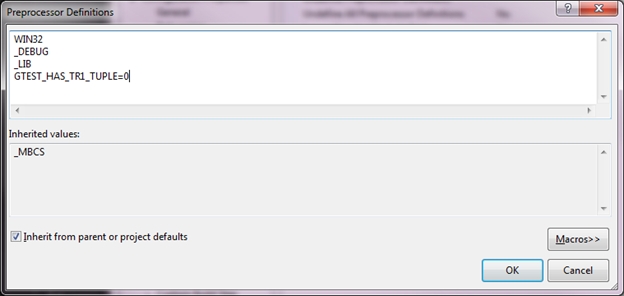
As  a target you get C:\src\gtest\1.8.0\googletest\msvc\gtest/Debug\gtestd.lib static library.

NOTE for VS2012

If you use VS2012 you should add GTEST\_HAS\_TR1\_TUPLE=0 definition to all of these 4 projects

* gtest,
* gtest\_main,
* gtest\_prod\_test,
* gtest\_unittest

In properties of the each project go to the C++->Preprocesor->Preprocessor definitions



FYI: Choosing a TR1 Tuple Library

Some Google Test features require the C++ Technical Report 1 (TR1) tuple library, which is not yet available with all compilers. The good news is that Google Test implements a subset of TR1 tuple that's enough for its own need, and will automatically use this when the compiler doesn't provide TR1 tuple.

Usually you don't need to care about which tuple library Google Test uses. However, if your project already uses TR1 tuple, you need to tell Google Test 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 and your tests. If you want to force Google Test to use its own tuple library, just add

-DGTEST\_USE\_OWN\_TR1\_TUPLE=1

to the compiler flags instead.

If you don't want Google Test to use tuple at all, add

-DGTEST\_HAS\_TR1\_TUPLE=0 (set this for the compilation for VS2012)

and all features using tuple will be disabled.

See

<https://github.com/appveyor/ci/issues/742>

<https://msdn.microsoft.com/en-us/library/bdscwf1c.aspx>

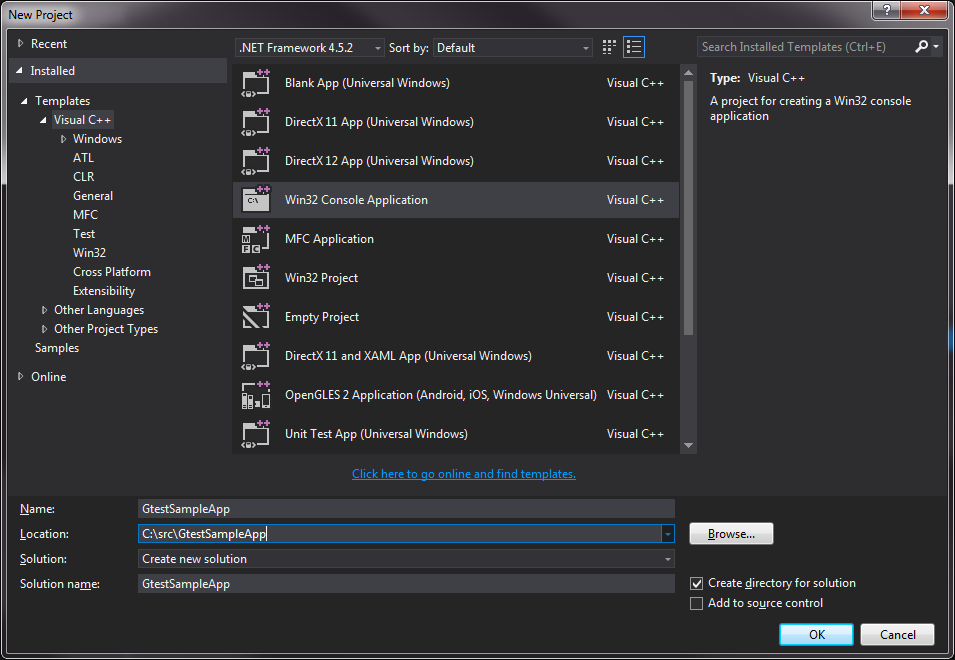
Also in C/C++ ->Code generation->Runtime library set /MTd option

Create test application

Now let's create test application

In VS2015\VS2012 File->New->Project->Win32 Console Application

Name: GtestSampleApp



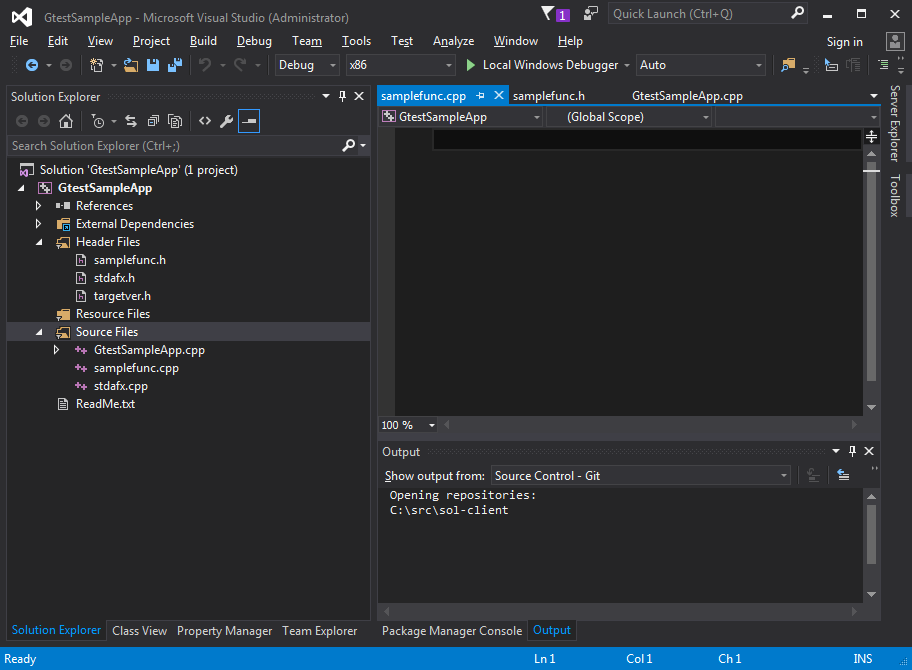
Click Next, remove the mark from Security Development Lifecycle (SDL) checks option, click Finish

Let's create simplest function to test - make separate files to emulate larger project.

Click on the subfolder Header Files inside GtestSampleApp solution and right click. Choose Add->New Item and in new dialog box Header File(,h). Change name of the file in the Name: field to samplefunc.h.

After add the same way a file samplefunc.cpp to the Source Files subfolder.

The source tree looks now:



The source of the samplefunc.h

#ifndef \_\_SAMPLEFUNC\_H\_\_

#define \_\_SAMPLEFUNC\_H\_\_

int multiplyByTwo(int k);

#endif

// #ifndef \_\_SAMPLEFUNC\_H\_\_

The source of the samplefunc.cpp

#include "samplefunc.h"

int multiplyByTwo(int k)

{

return k \* 2;

}

Right on the GtestSampleApp and go to the Properties

Then create two more files for the create tests. Despite the fact all of these things may be placed to one file with main functions I try to show the important concept of the Google Tests - tests may (and should.. and must) be undependable  from main code.

So create files tests.h and tests.cpp respectively.