TestNG++

Installation & Configuration Guide For MSVC Users

Arthur Yuan

ABSTRACT

This document guides you how to configure, install and use TestNG++ on Windows, along with Microsoft Visual Studio.

1. Configuration, Building & Installation

1.1 Preparation

Firstly, go to the download page of TestNG++ web site (http://code.google.com/p/test-ng-pp), choose a certain version of source archive and download it. Alternatively, you can check out the latest source code directly from SVN repository (http://test-ng-pp.google.com/svn/truck). However, it is usually not as stable as the formal released version.

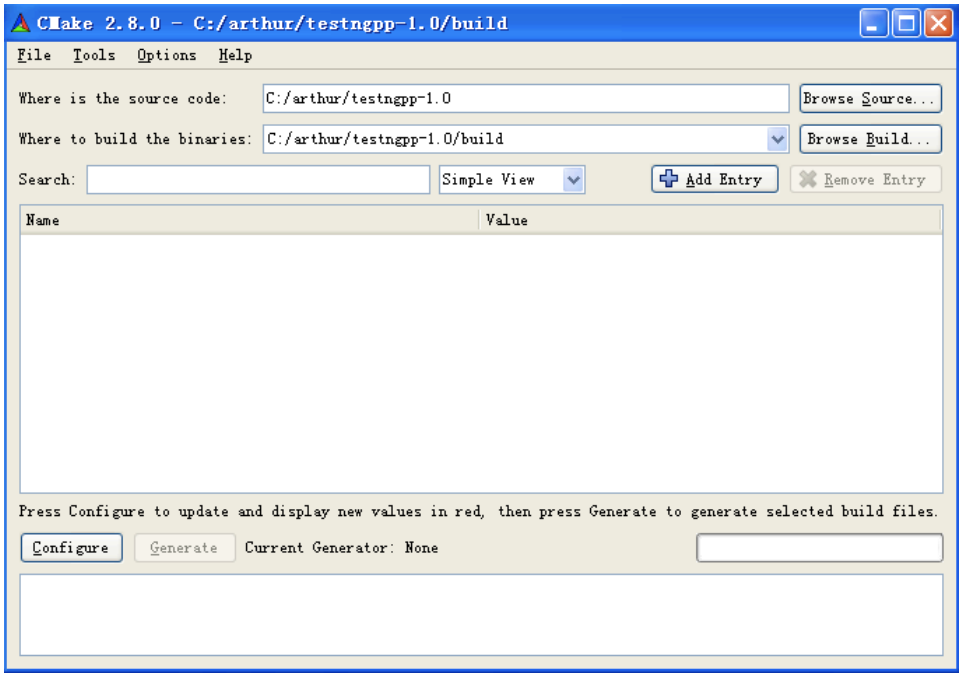
TestNG++ uses **cmake** (http://www.cmake.org) as the building system. Besides, the ***Test Generator*** of TestNG++ is written in python. Therefore, before you are able to build TestNG++, you need to make sure you have **cmake** and **python** installed on your system.

If you are going to use TestNG++ with **Microsoft Visual Studio**, you also need to have it installed on your system.

After all these get ready, uncompress the source code tar ball to somewhere, ***C:\arthur***, for instance. Then you get ***C:\arthur\testngpp-1.0***. Afterwards, create a new folder in which all **cmake** generated files & build outputs will be placed. In this example, the folder we create is ***C:\arthur\testngpp-1.0\build***.

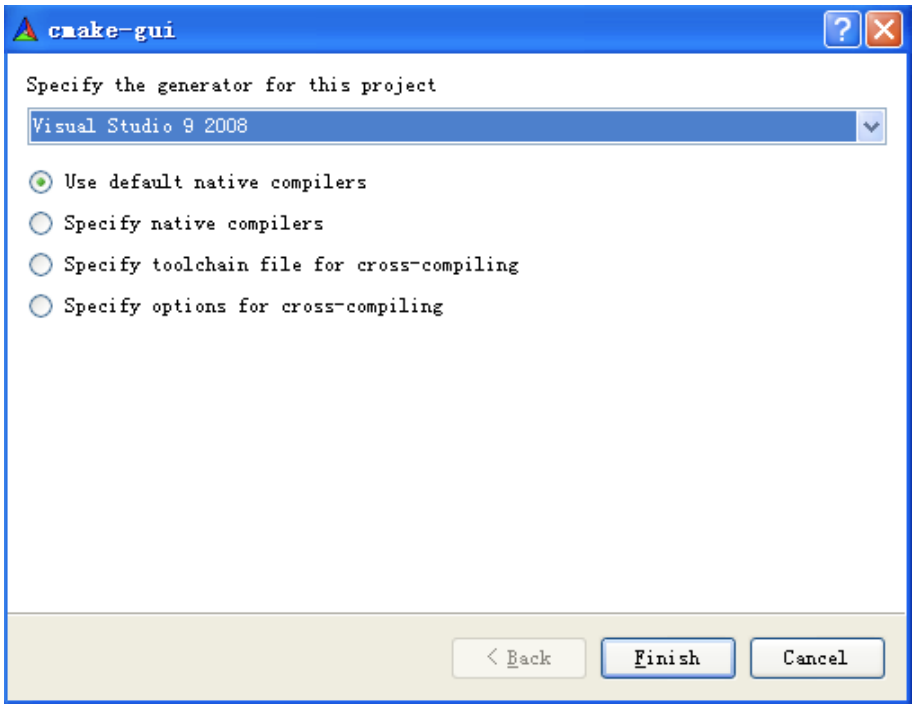
1.2 Configuration

Now, launch the **cmake-gui**[1]. Set the source code path in ***"Where is the source code"***, and the build path in ***"Where to build the binaries"***.



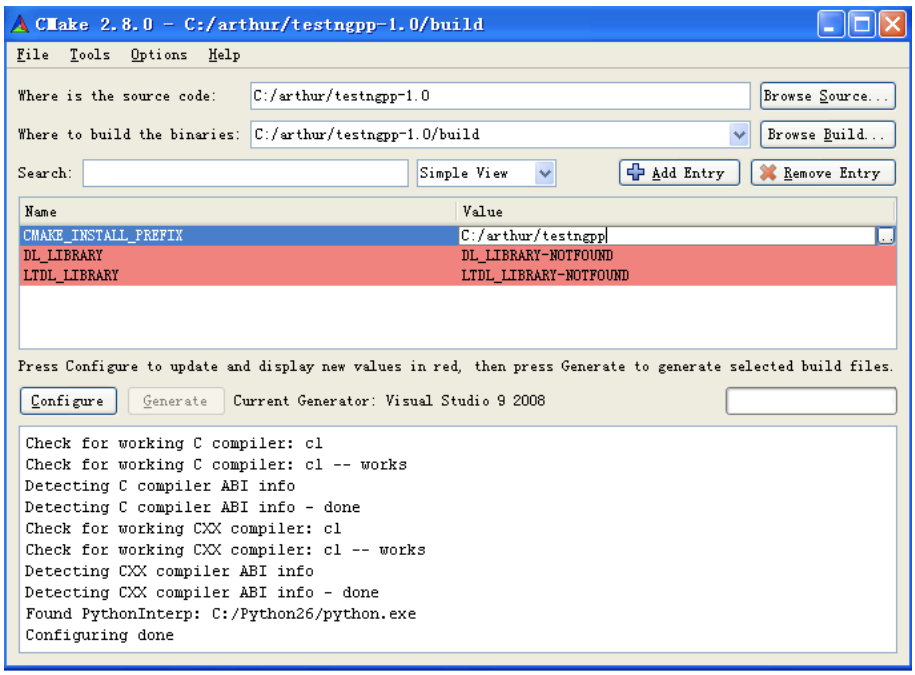
[1] You can also choose to use command line of **cmake**, here we use **cmake-gui** for the sake of demonstration.

Then, click the button ***"Configure"***, you will see a pop-up window like this:

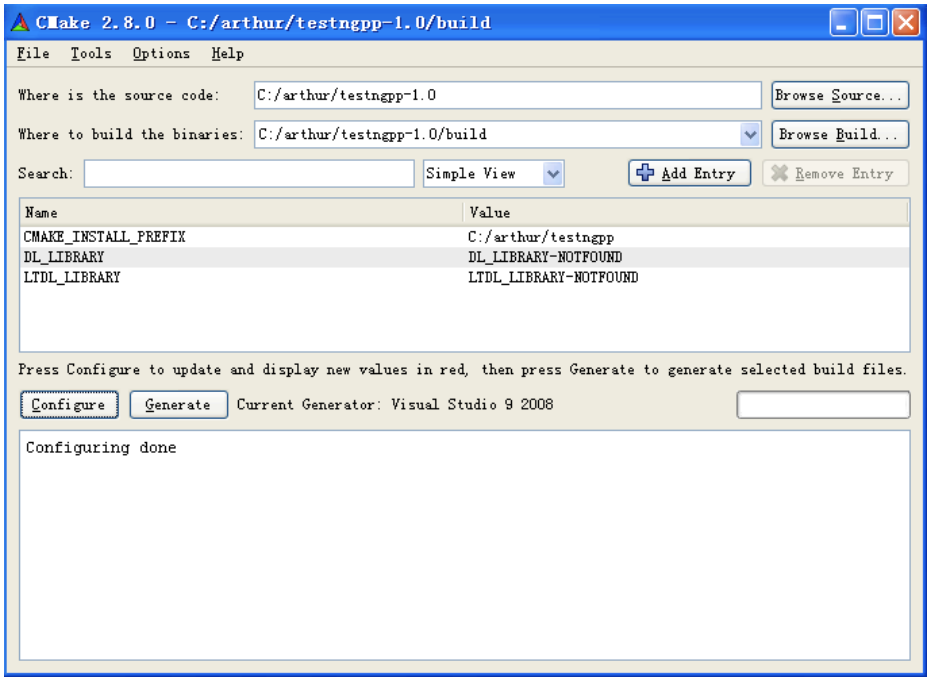


Choose a generator according to your **Visual Studio** version, and select ***"Use default native compilers"***, and then click the button ***"Finish"***, which will start the configuration process.

After the configuration finishes successfully, modify the value of the variable **"CMAKE_INSTALL_PREFIX"** to the path where you are going to install TestNG++. On Windows, its default value is ***C:\Program Files\testngpp***.



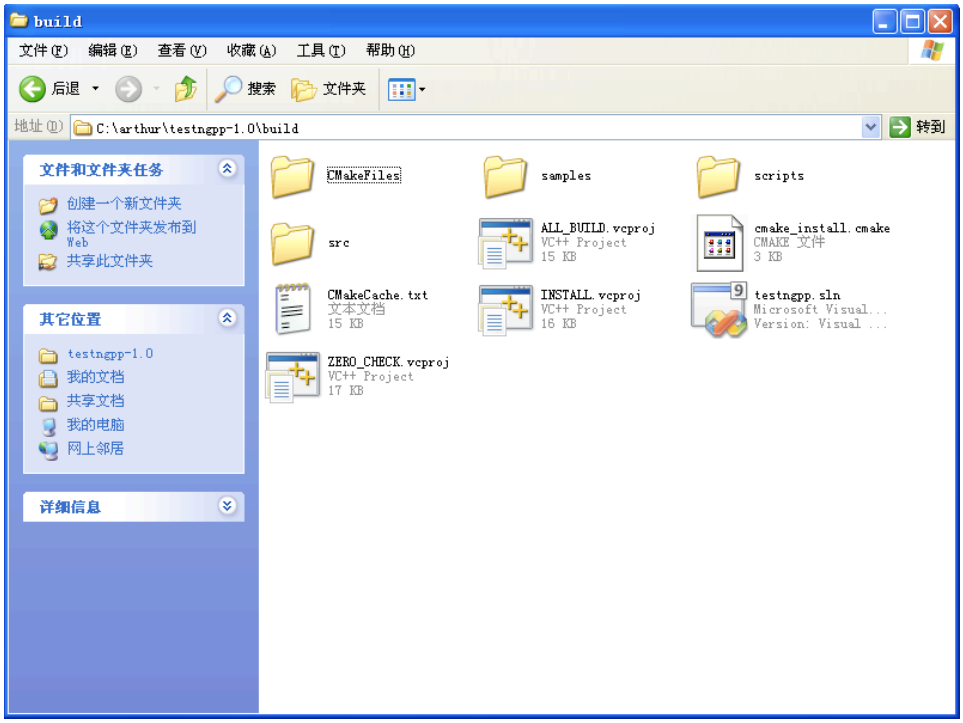
After setting the installation path, click the **"Configure"** button again, then you'll get:



Click the **"Generate"** button to generate the **Visual Studio** project files.

1.3 Building & Installation

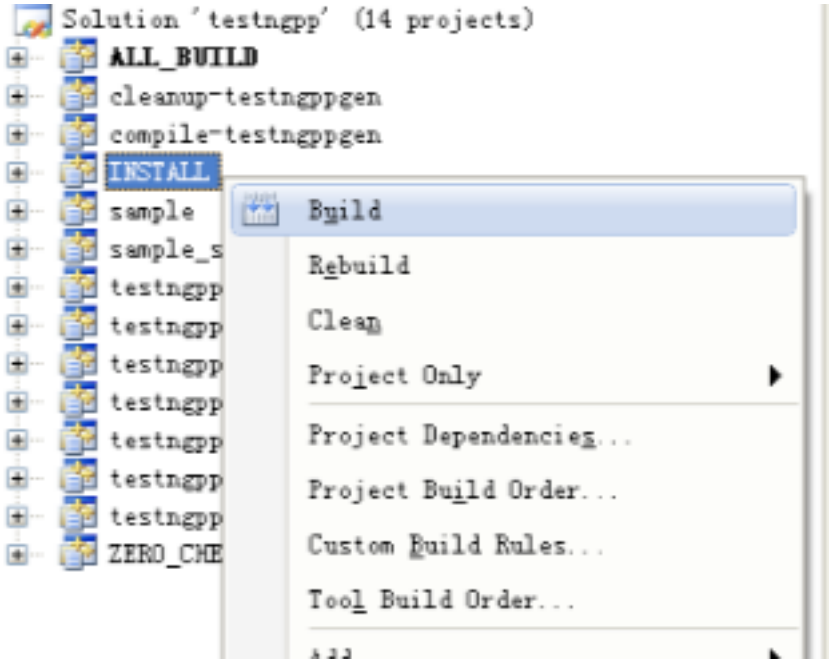
Now, close the **"cmake-gui"** window, open the folder **c:\arthur\testngpp-1.0\build**. Here you can see the solution and project files, as well as others.



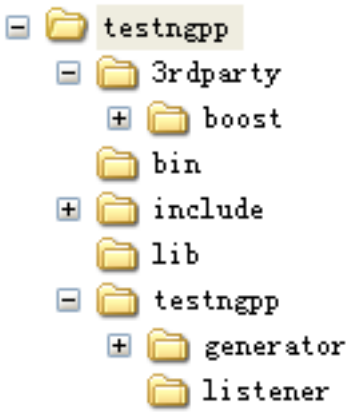
Double click the solution file ***"testngpp.sln"*** to open it in **Visual Studio**.

In Visual Studio, press **F7** to build the solution. After TestNG++ is built successfully, right click the project **"INSTALL"** and select **"build"** to install

TestNG++ to the folder "***C:\arthur\testngpp***", which was set in previous configuration.



After the installation process finishes successfully, open the folder "***C:\arthur\testngpp***". You can see the following directory structure:



While using **MSVC**, TestNG++ uses the **"typeof"** library of **boost**; all related libraries of which were installed in ***"3rdparty\boost"***.

In ***"bin"***, **testngpp-runner** is placed.

The header files of TestNG++ are installed in ***"include"***, you are going to need to include them when you start to write test.

Library **testngpp.lib** locates in ***"lib"***. When you build your tests, you need to link it into your test modules.

Test Generator is written in **python**, the compiled python byte code files are put in ***"testngpp\generator"***.

Although users can develop their own **Test Listener** with the mechanisms provided by TestNG++, TestNG++ offers two **Test Listener**s for immediate use.

- **Stdout Listener**
- **XML Listener**.

The module files are **testngpplistener.dll** and **testngppxmllistener.dll** respectively, which were installed in ***"testngpp\listener"***.

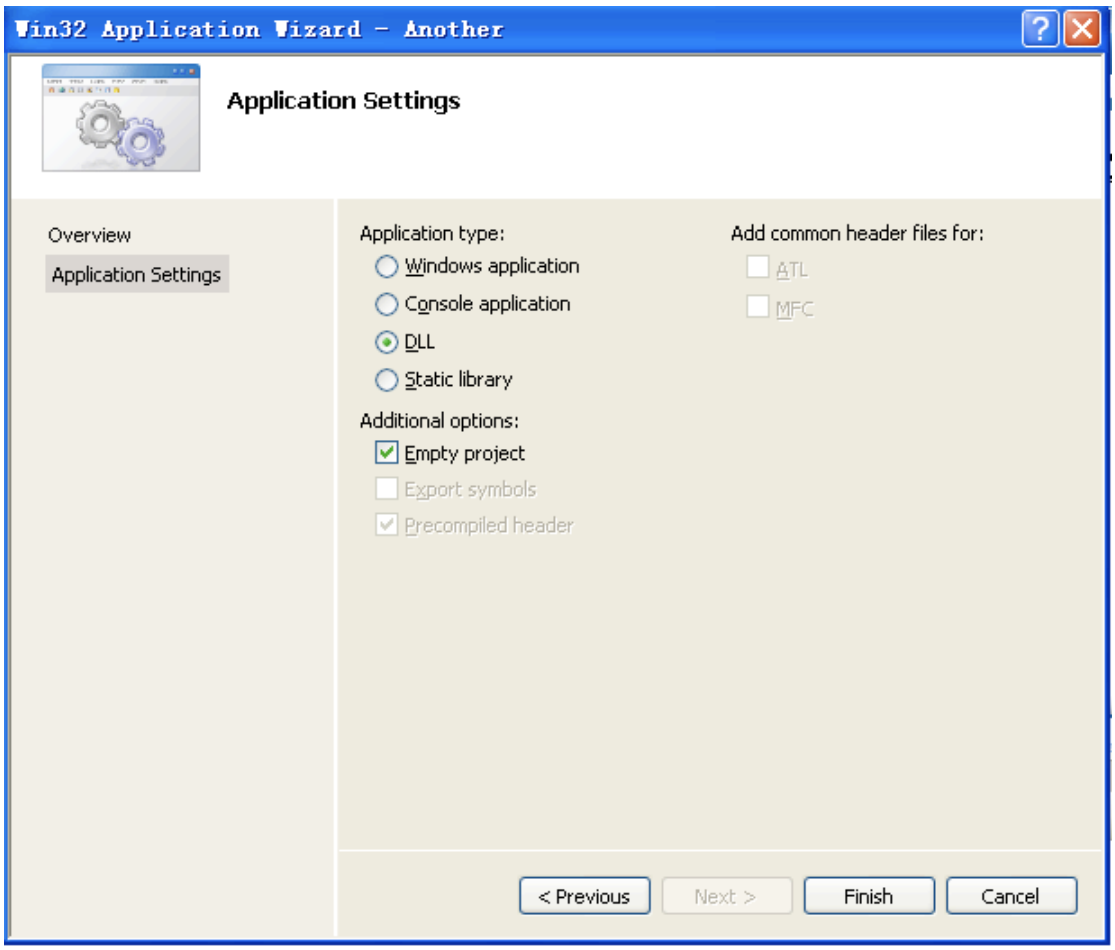
2. Using TestNG++

There are many patterns of using TestNG++ in the environment of **Microsoft Visual Studio**. Here gives one possible pattern to demonstrate the usage of TestNG++.

2.1 Create Projects

Create your **SUT** (System Under Test) project as a **win32 static library** one[2]; and create a **Test** project, which should be a **win32 dynamic library** project.

Please note that when you create **Test** project, you should make sure **"Empty project"** is selected.



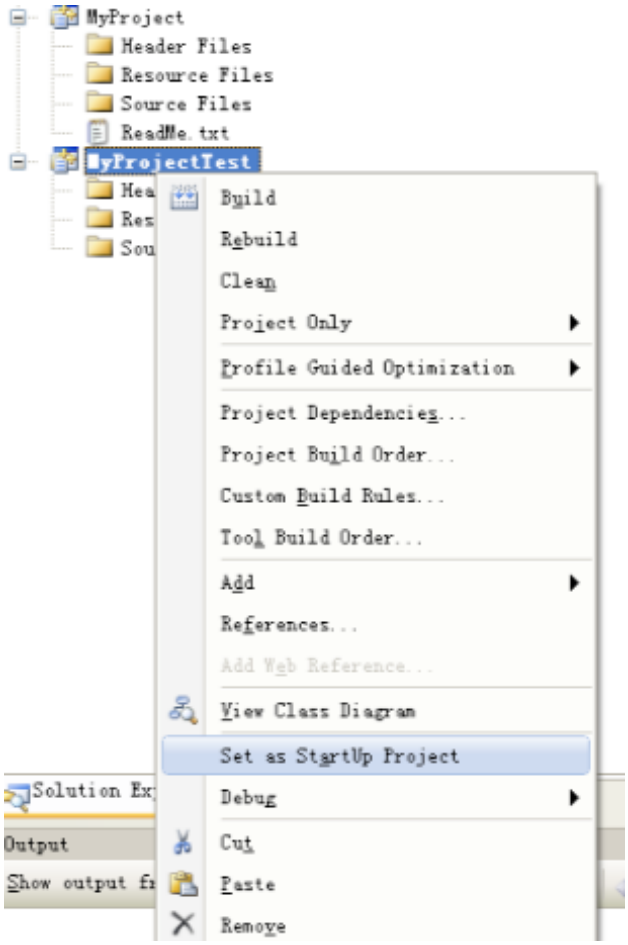
For instance, we created a **SUT** project named ***MyProject***, and the **Test** project is ***MyProjectTest***.

---

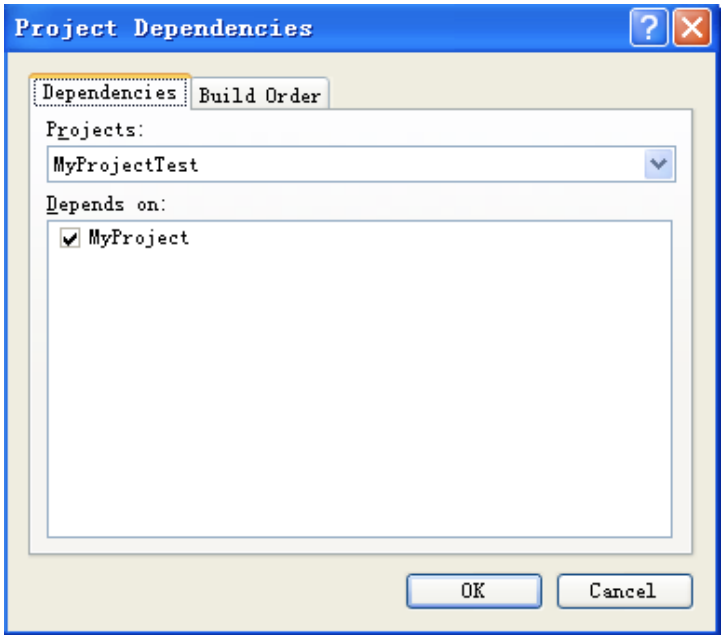
[2] Although in reality, you project is an executable or a dynamic library, but still, you can create another static library project as the **SUT** project, and let your real project depends on it.

2.2 Set Dependencies

Then we set ***MyProjectTest*** as the startup project:

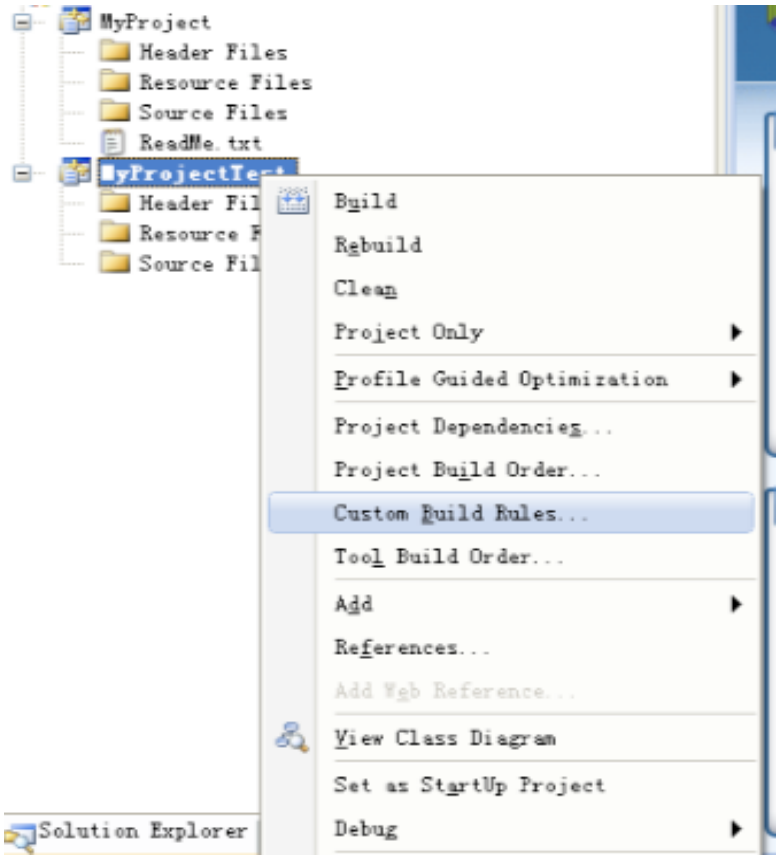


And by **"Project Dependencies"**, set ***MyProject*** as the dependency of ***MyProjectTest***.

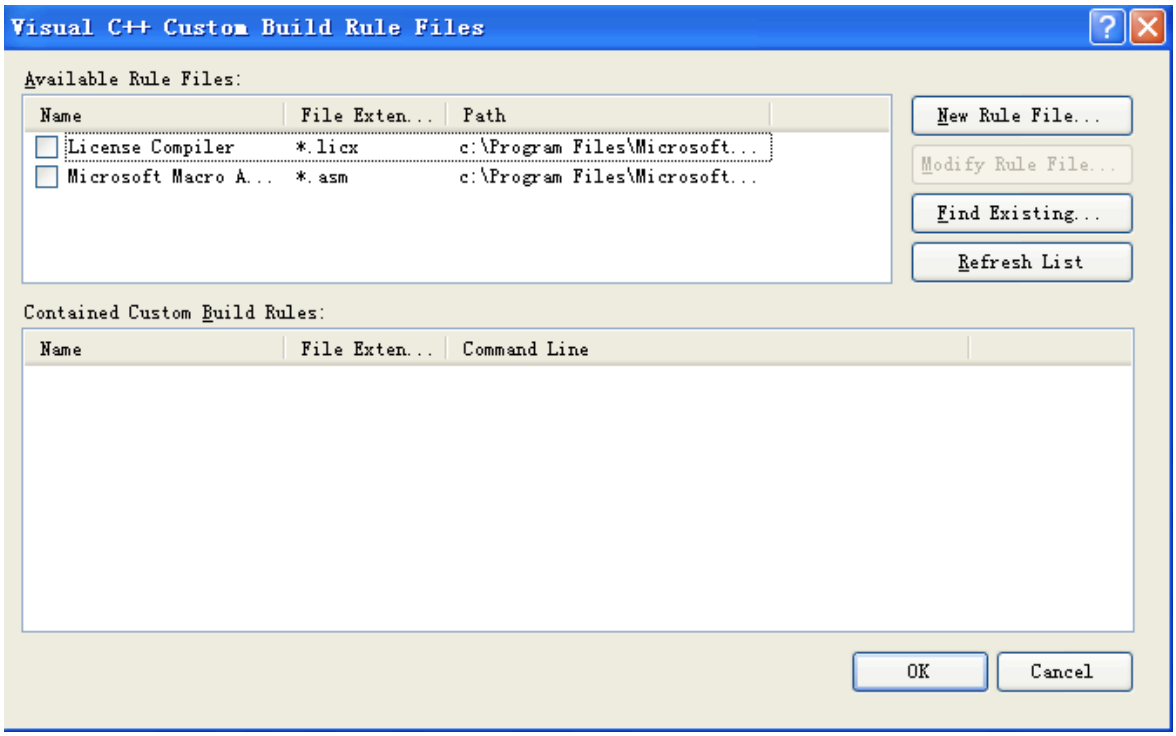


2.3 Setup Test Generator

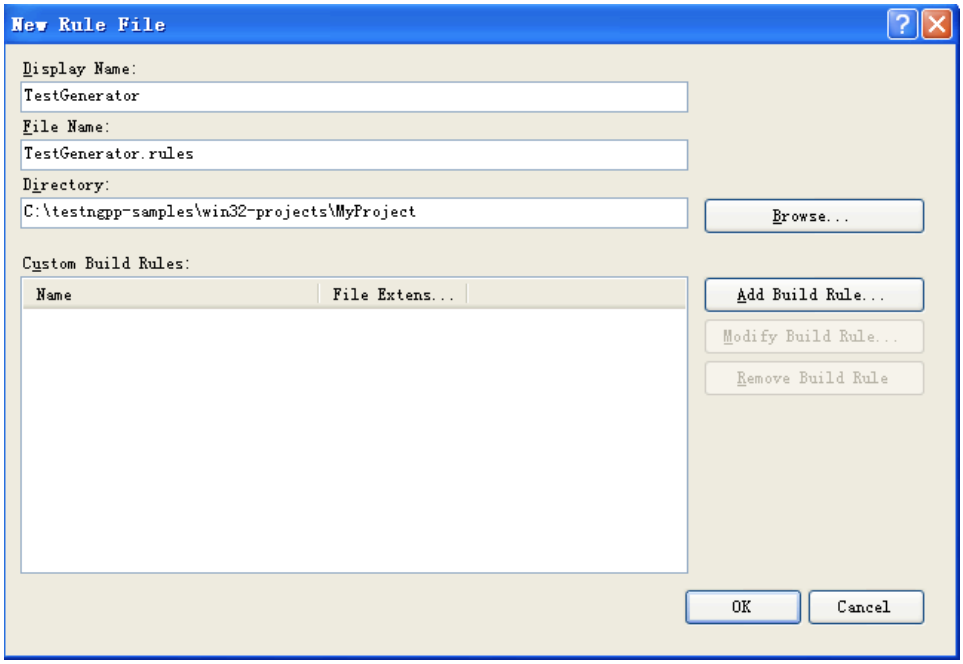
Now setup the **"Custom Build Rules"** of ***MyProjectTest***.



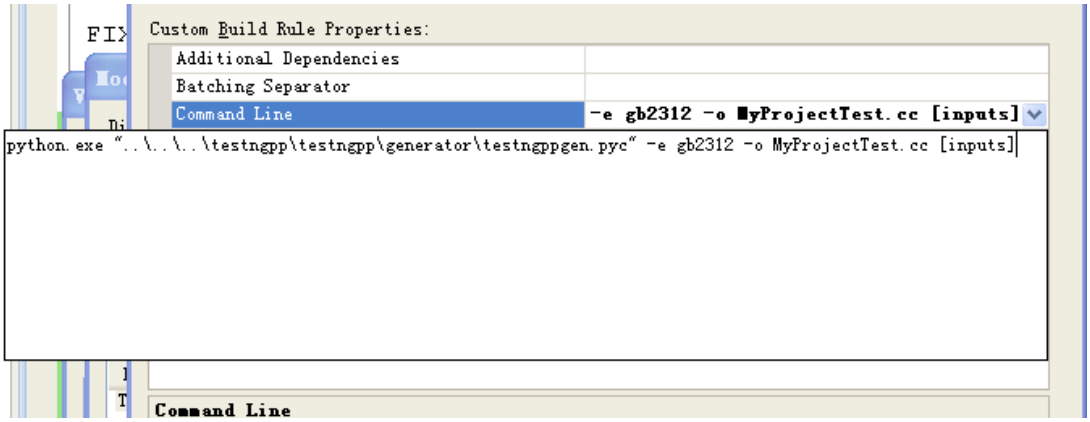
In the popup window, click the **"New Rule File..."** button.



Set the rule name, rule file name and location; then click the **"Add Build Rule..."** button:

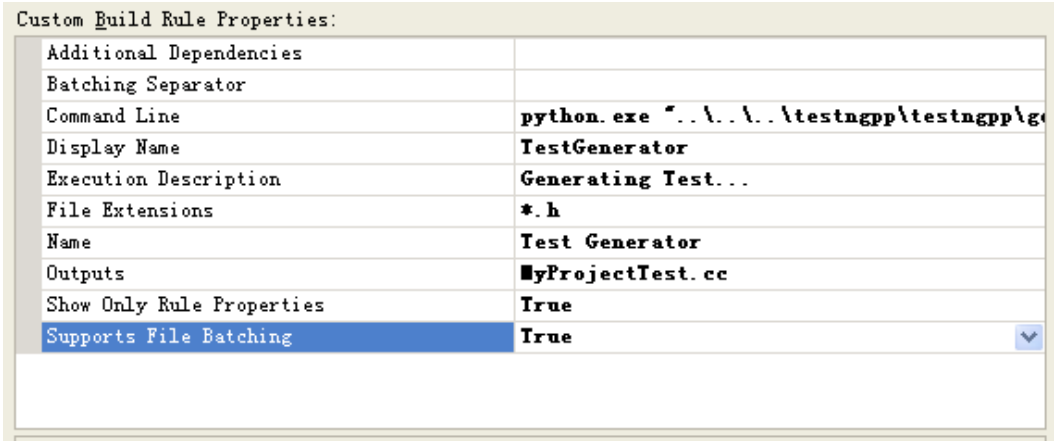


In the popup windows, write the command in the "command line", like this:



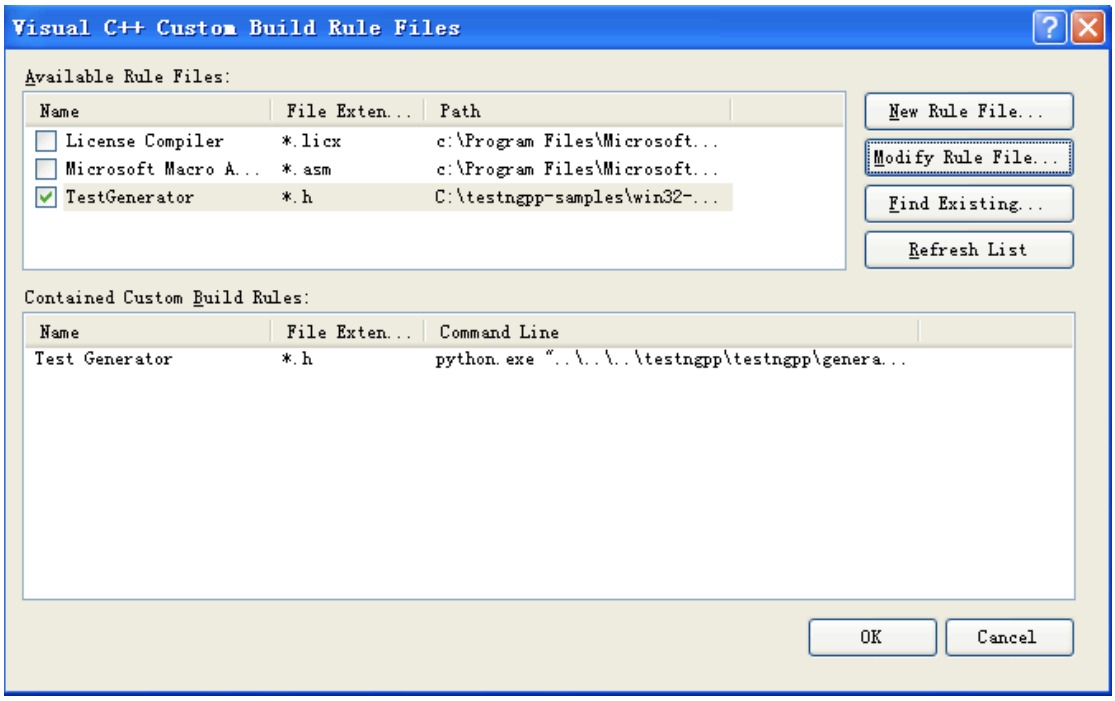
This command line is actually an invocation to the **Test Generator**. The option **"-o"** is used to specify the generated output file, and the option **"-e"** is for specifying the encoding of test source files. If this option is not specified, by default, the encoding is **"utf-8"**. In this example, we set it as **"gb2312"**.

Then we set other fields as following:



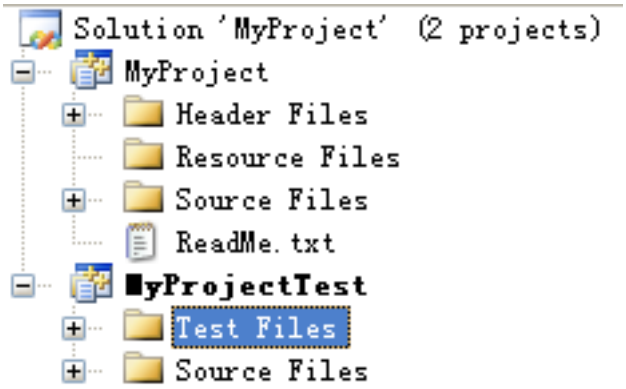
You may have noticed that we set the **"File Extensions"** as **"*.h"**, which means, in ***MyProjectTest***, all headers with the extension **"*.h"** are thought as test source files, you should no longer use this extension for any non-test file. If you intent to use **"*.h"** as the extension of normal headers, you should set **"File Extensions"** as different value, **"*.hpp"**, for instance.

Now click **"OK"** button to go back to the window, here you must make sure the new rules we created are selected.



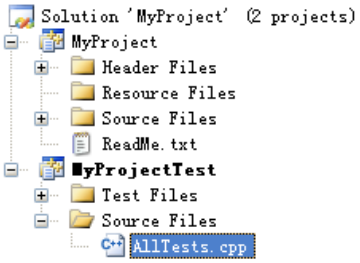
2.4 Create Source File

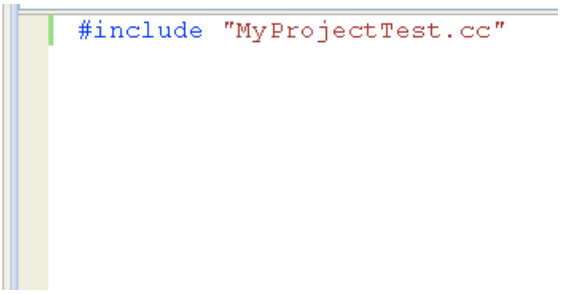
Change the name of **"Header Files"** filter to **"Test Files"**. It's simply for clarity. Actually, it does not matter at all whatever name it is.



Then create a source file **"AllTests.cpp"** or whatever names you prefer in project **MyProjectTest**.

Then open **"AllTests.cpp"** and edit it. It only has one line source code.





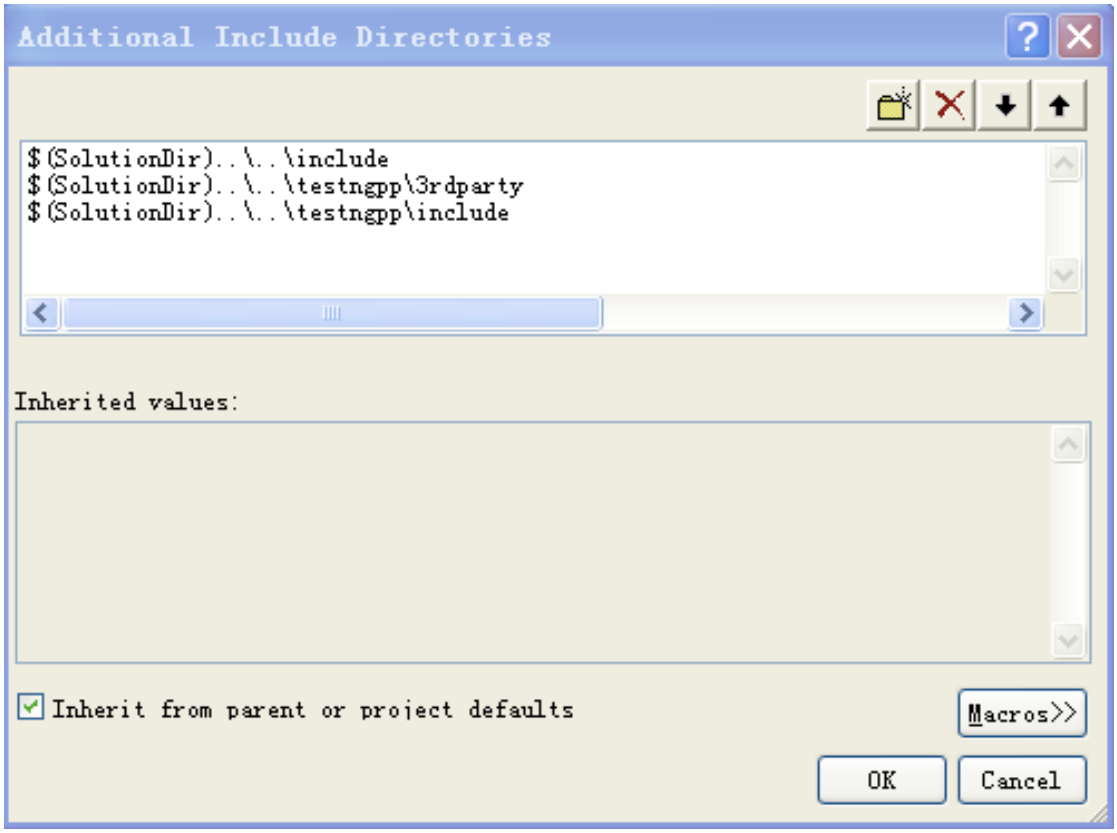
Yes, the included file is the one generated by **Test Generator.** We adopt this weird way for two reasons:

1. We have to have a source code file in project ***MyProjectTest***, otherwise, we can neither set the compiling option easily nor compile the project to get the DLL we need.
2. If we add **"MyProjectTest.cc"** into project directly, because its content is generated according the contents of test files and changes always, you will get annoyed constantly by the message of reminding you to reload this file because MSVC has detected the change of it content.

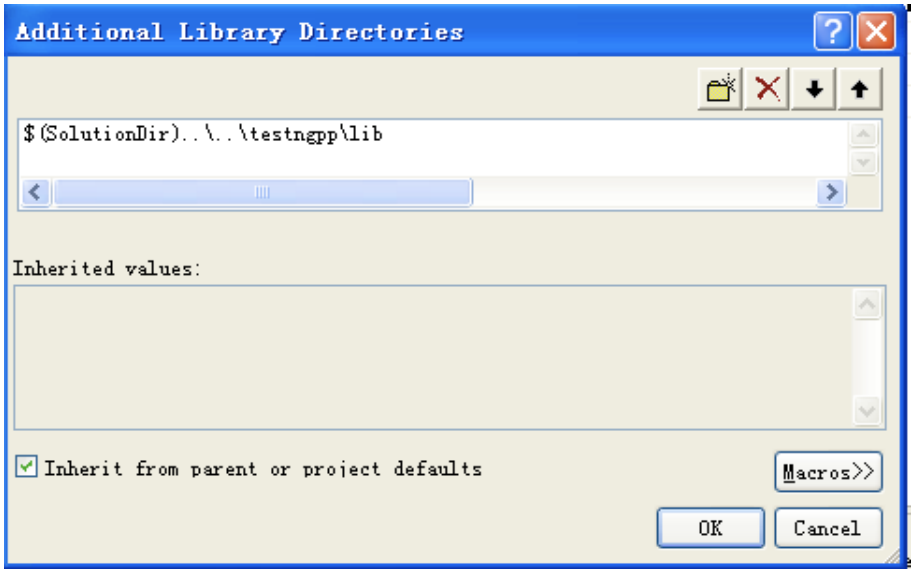
Actually, it's merely a cheap way to workaround these trouble.

2.5 Setup Include Path & Linking Libraries

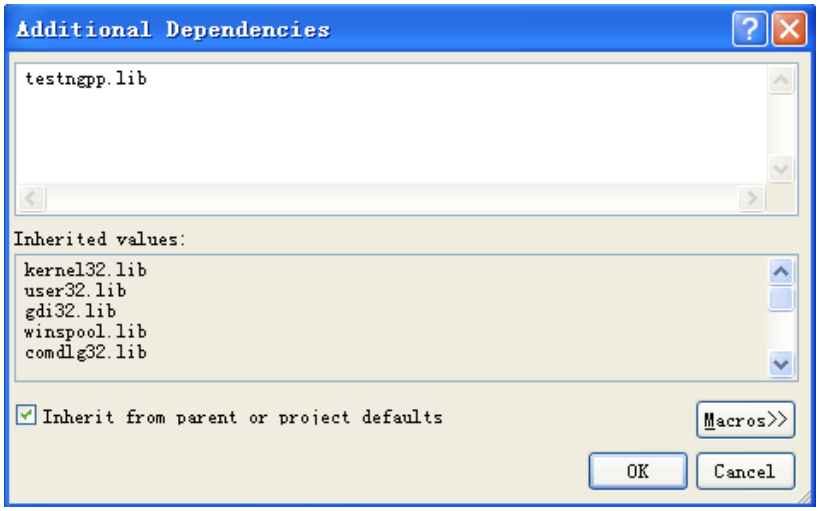
Now add the paths of TestNG++ headers and **boost**, as well as the path of headers of project ***MyProject***, to the **"Additional Include Directories"** of project ***MyProjectTest***.



Also, add the library path of TestNG++ to the **"Additional Library Directories"**.

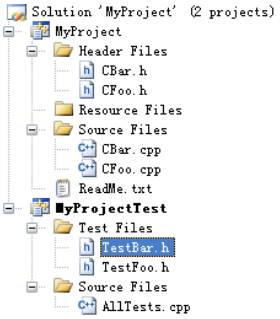


And the TestNG++ library should be added to **"Additional Dependencies"** as well.



2.6 Create & Writing Test

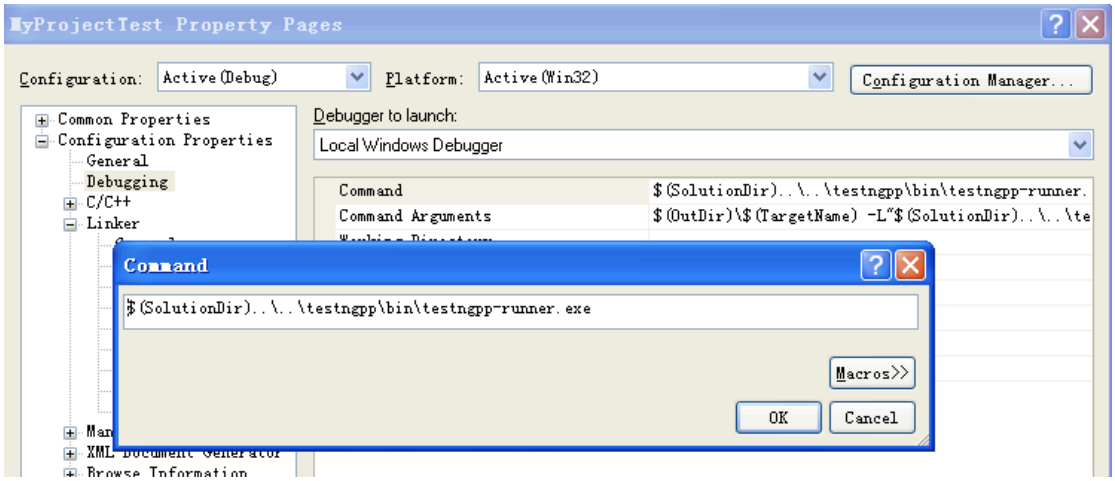
Now create test sources, write test, and implement the SUT.



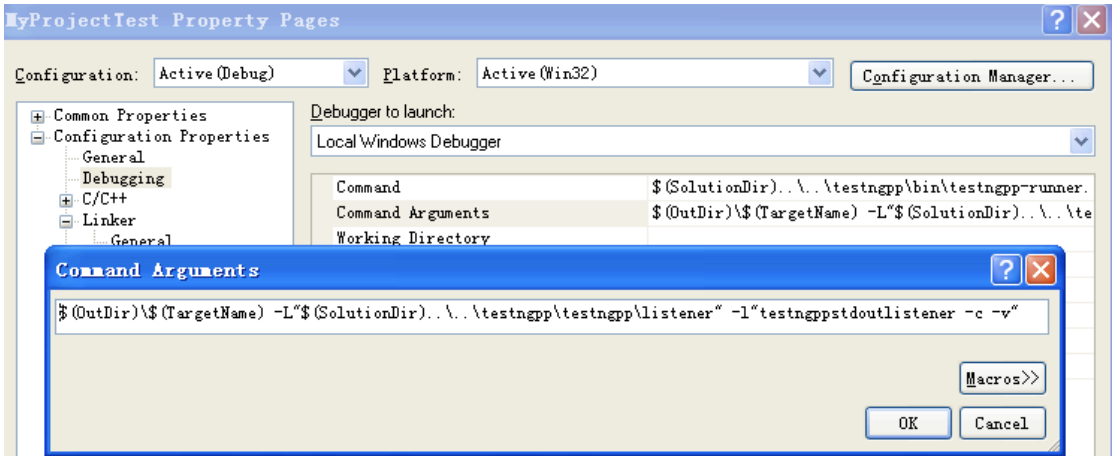

```
#include <testngpp/testngpp.hpp>

#include <CBar.h>

FIXTURE(CBar)
{
    TEST(should be able to multiply 2 integers)
    {
        ASSERT_EQ(6, CBar::multiply(2, 3));
    }

    TEST(should be able to div 2 integers)
    {
        INFO("the result should be double type");
        ASSERT_EQ(3, CBar::divide(6, 2));
    }

    TEST(if devident is 0 => throw exception)
    {
        WARN("not implemented yet");
    }
};
```

2.7 Setup The Debugging Command Line

Now set the debugging command line of project ***MyProjectTest***. The command is TestNG++ **Test Runner**.



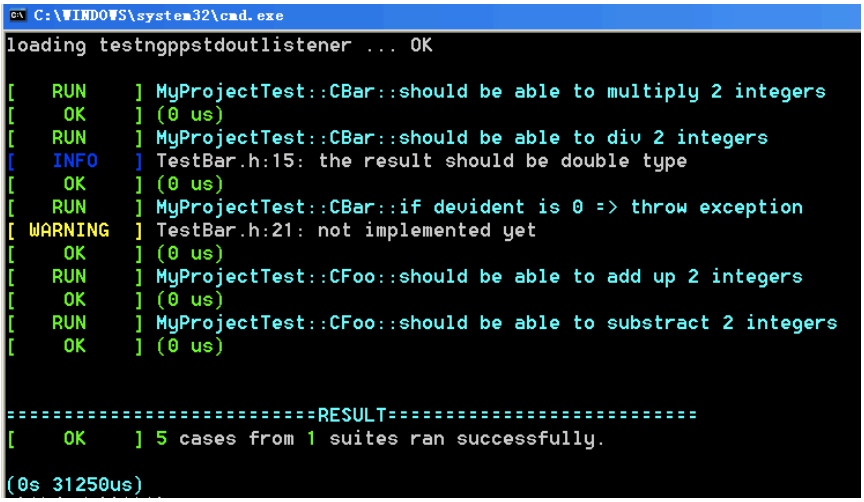
And the argument could be set like this:



About the options and arguments of Test Runner, please refer ***TestNG++ User Manual***.

2.8 Building & Running Test

Press **"Ctl + F5"** to build tests and run them, if everything goes well, the TestNG++ Stdout Listener will report the test result.



2.9 Get The Example

The whole example could be downloaded from **testngpp-samples** (http://code.google.com/p/testngpp-samples)