

## C++ Test Tools including FIT and CppUnitLite

This package supports acceptance testing and unit testing. Currently only the Microsoft Visual C++V6 and MS Visual Studio.NET are supported. It was developed using VC++ V6, ported to VS.NET. It has been developed and tested with fitnessse20040408.

Included in the CppTestTools zip file are:

- TestTools.sln – VS.NET solution file
- TestTools.dsw – VC++V6 workspace
- HomeGuard.zip – a sample application with unit and acceptance tests
- HomeGuardTests.zip – HomeGuard Fitnessse test pages

### Installation

Unzip CppTestTools into your fitnessse\cpp directory, for example

C:\tools\fitnessse\cpp

Unzip the contained HomeGuard zip file where ever you might put a project, for example

C:\Projects\HomeGuard

Unzip HomeGuardTests into FitNesseRoot, for example

C:\tools\fitnessse\FitNesseRoot

Define these environment variables using the correct paths on your machine:

Environement variable	Example value
CPP_PLATFORM_ROOT	C:\tools\fitnessse\cpp\Platforms\VisualCpp
CPPFIT_ROOT	C:\tools\fitnessse\cpp\Fit\include
CPPTEST_LIB	C:\tools\fitnessse\cpp\lib
CPPUNIT_ROOT	C:\tools\fitnessse\cpp\CppUnitLite\include

To use CppUnitLite include “TestHarness.h” from \$(CPPUNIT\_ROOT) directory and link to CppUnitLite.lib and VisualCpp.lib in the \$(CPPTEST\_LIB) directory. A conventional usage is to have a test file for each class. See the HomeGuard example.

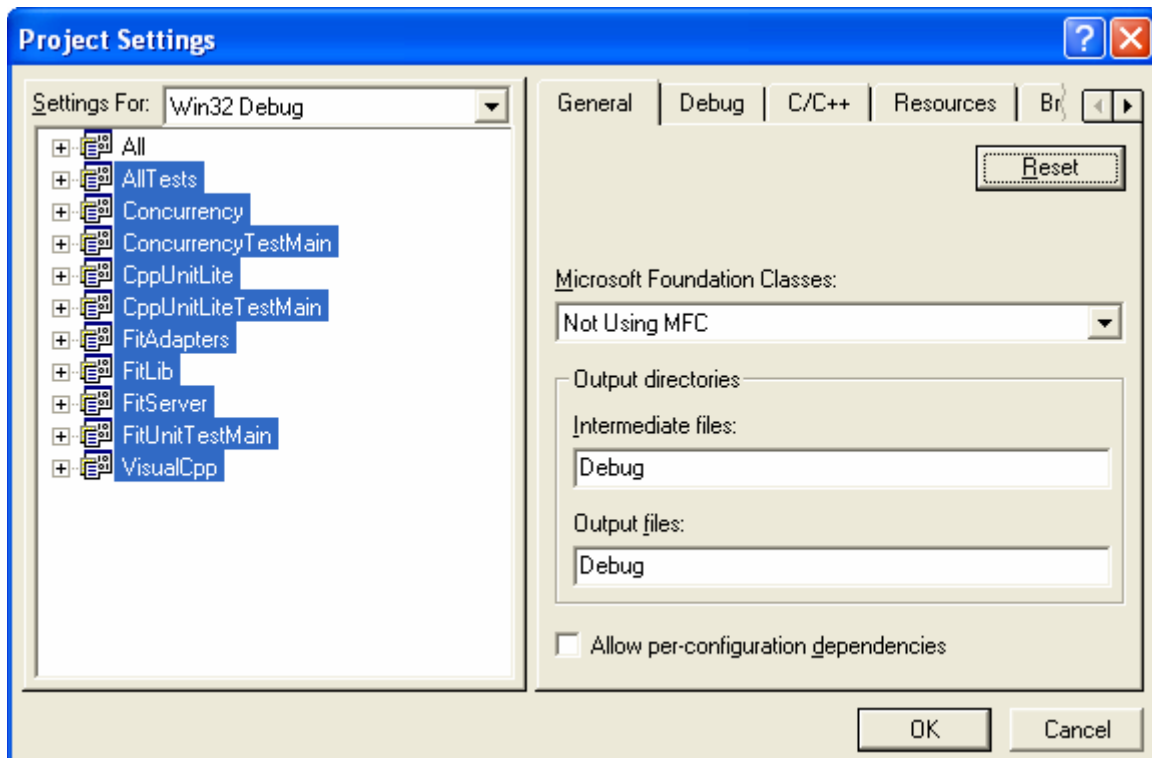
To use CppFit include “Fit.h” from the \$(CPPFIT\_ROOT) directory and link to FitLib.lib and VisualCpp.lib in the \$(CPPTEST\_LIB) directory.

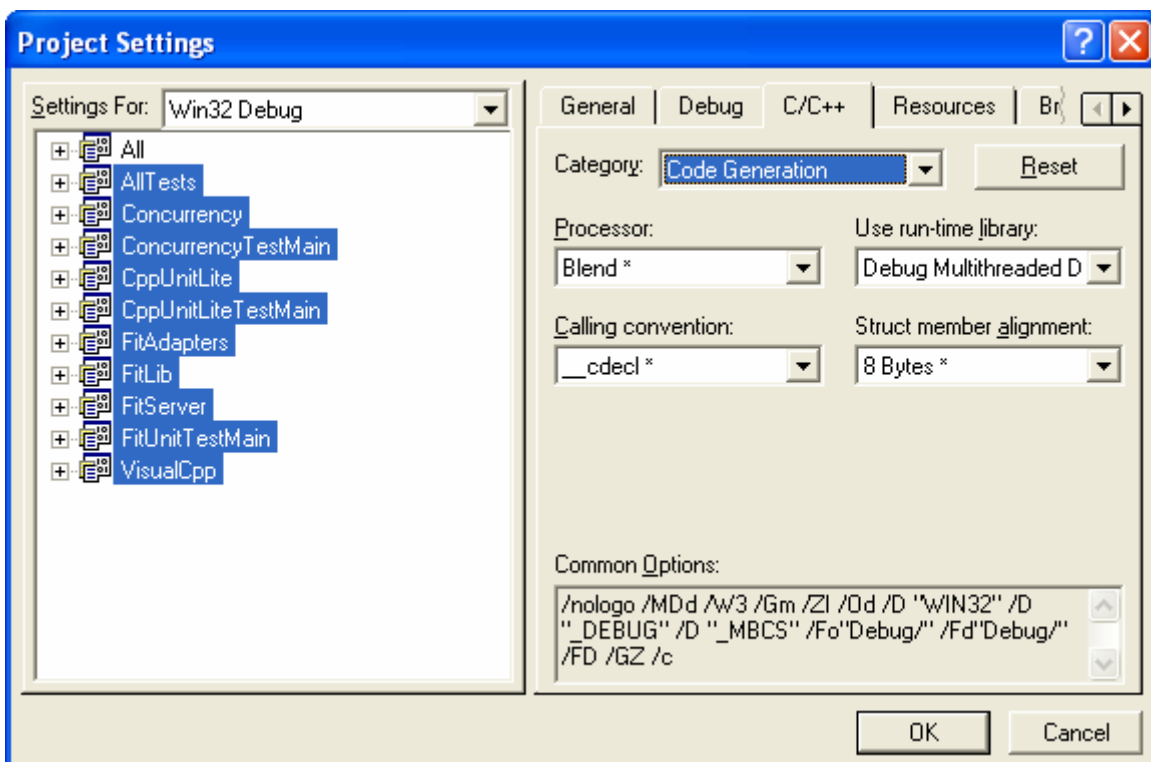
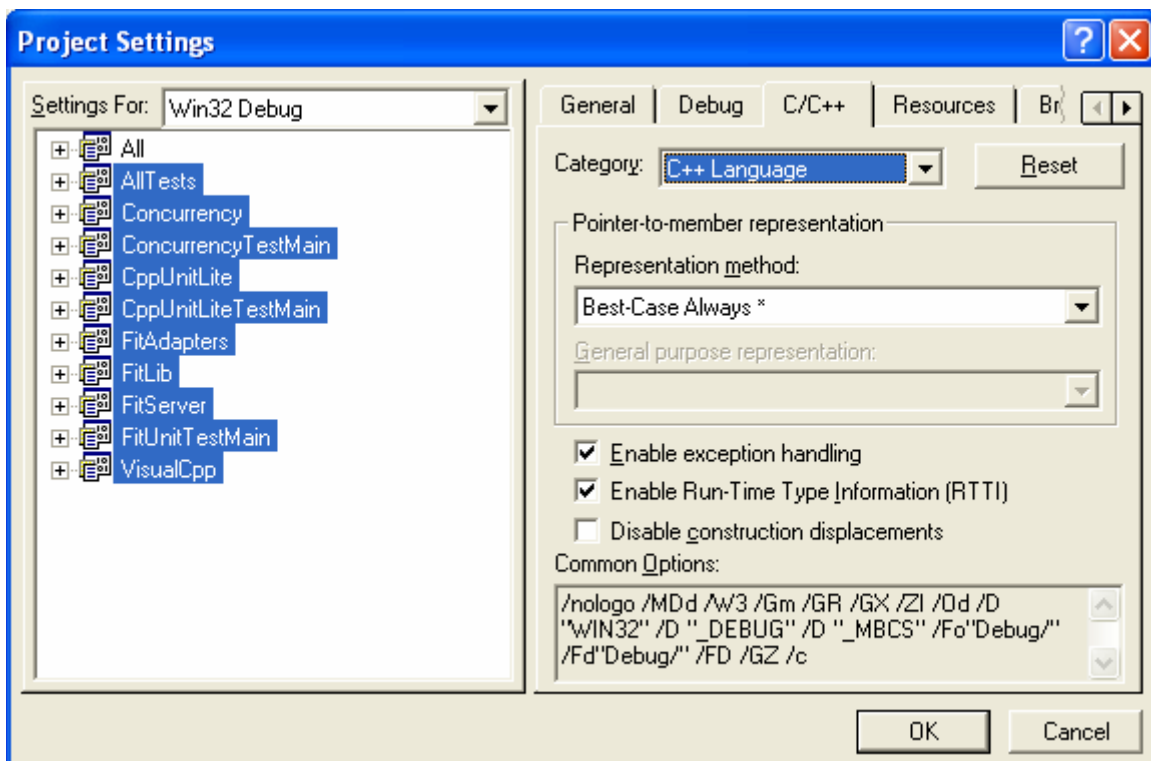
## TestingTools Solution/Workspace

Assuming the directories above (C:\tools\fitness\cpp), there is a VS.NET solution and a VC++ V6 workspace named TestTools.sln and TestTools.dsw. You can use CppFit and CppUnitLite without doing anything with this workspace. You can directly use the header files and binary files provided. The workspace contains these projects:

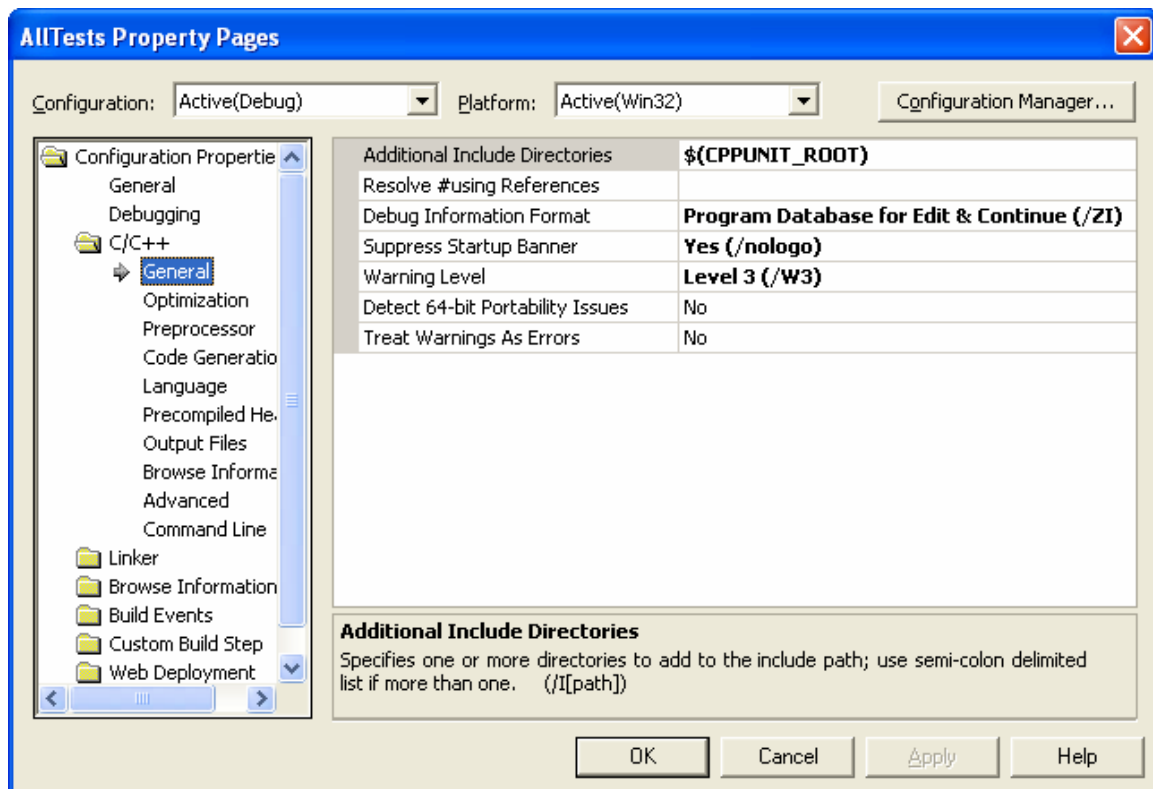
Project	Notes
All	Dummy product used to build and clean all projects
AllTests	All the unit tests
Concurrency	Concurrency tools for windows
CppUnitLite	Unit test harness used for CppFit development
FitServer	Creates FitServer.exe, that FitNesse talks with to run FitNesse pages
FitAdapters	Static library that implements the socket adapter
FitLib	Static library that holds the C++ port of the FIT classes. Your test will link to the
FitUnitTestMain	Unit tests for FIT
VisualCpp	Visual C++ specific code

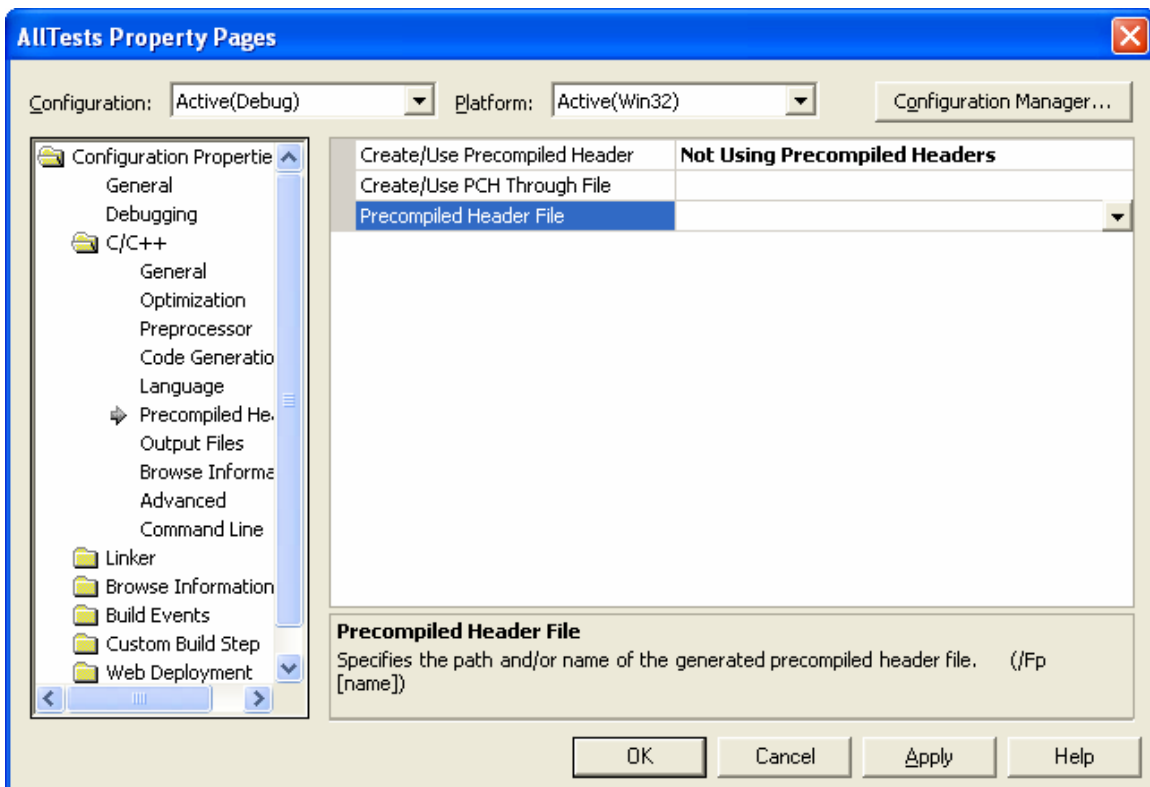
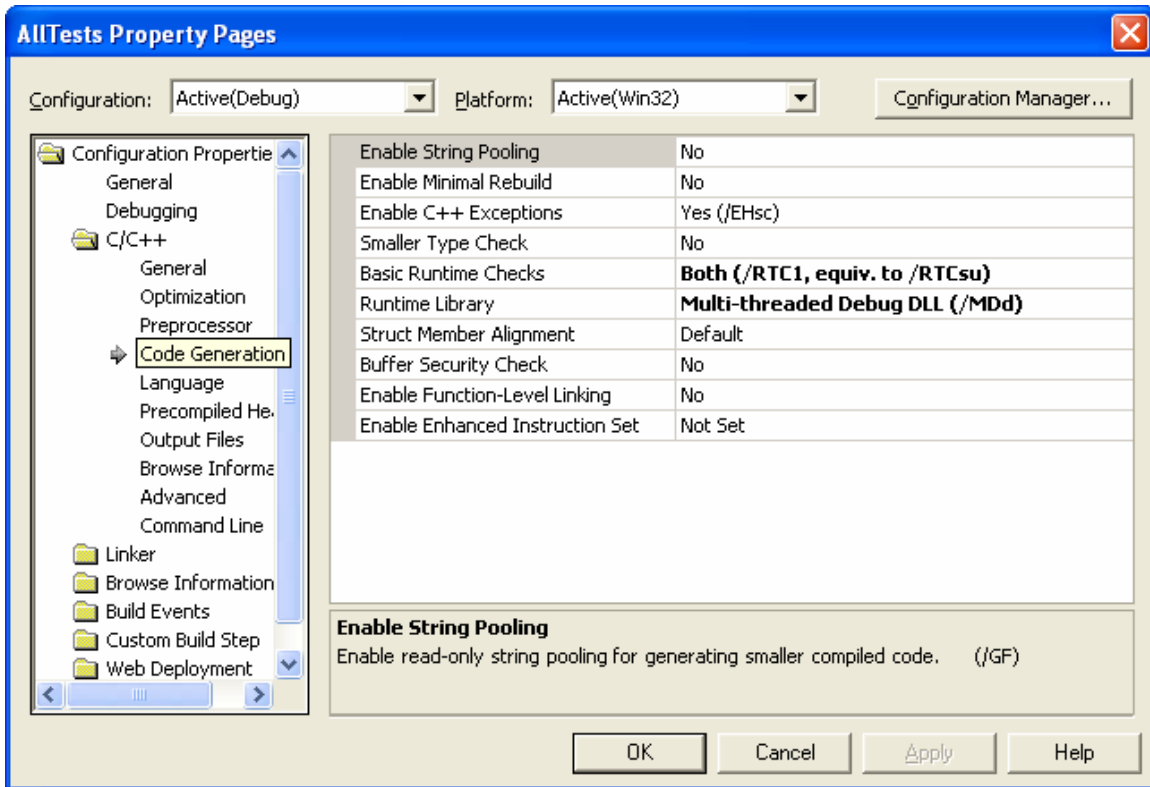
These are the significant project settings for CppTestTools in VC++6:





These are the significant project settings for CppTestTools in VS.NET:





## **CppUnitLite Enhancements**

- **Memory leak detection**  
CppUnitLite declares a class called MemoryLeakWarning, with methods that have platform specific implementations. There is an implementation for it the VisualCpp project that enables the Microsoft memory leak warning debug heap, and performs checks around each unit test. If you leave static variables behind referring to allocated objects that are not released until program termination you will see memory leaks reported on a per test basis. This feature requires a debug build.
- **SetUp() and TearDown() support**  
SetUp is called before, and TearDown is called after, the body of each non-ignored and non-filtered out test body. If you are using a previous version of CppUnitLite you will get compile errors unless you have a SetUp and TearDown function defined in each file. If you already have your own SetUp and TearDown functions that you call manually, they will probably get called twice. Note that SetUp and TearDown are placed in the unnamed namespace to avoid global namespace conflicts. Declaring them static also works, but is a deprecated construct.
- **IGNORE\_TEST macro**  
Ignores the test. The test still must compile, but it is not run.
- **Verbose mode (-v command line argument)**  
prints the test group and test name of tests that are run
- **Test filtering**  
The final command line argument specifies a string that is in the name of the tests or test groups that should be run. The other tests are skipped.
- **Test Repeat (-r [#])**  
Repeat execution of all tests # times. Default is 2. This is helpful to reveal if there are initialization problems.
- **EXPORT\_TEST\_GROUP and IMPORT\_TEST\_GROUP macros**  
Support for libraries that hold tests
- **More summary information**

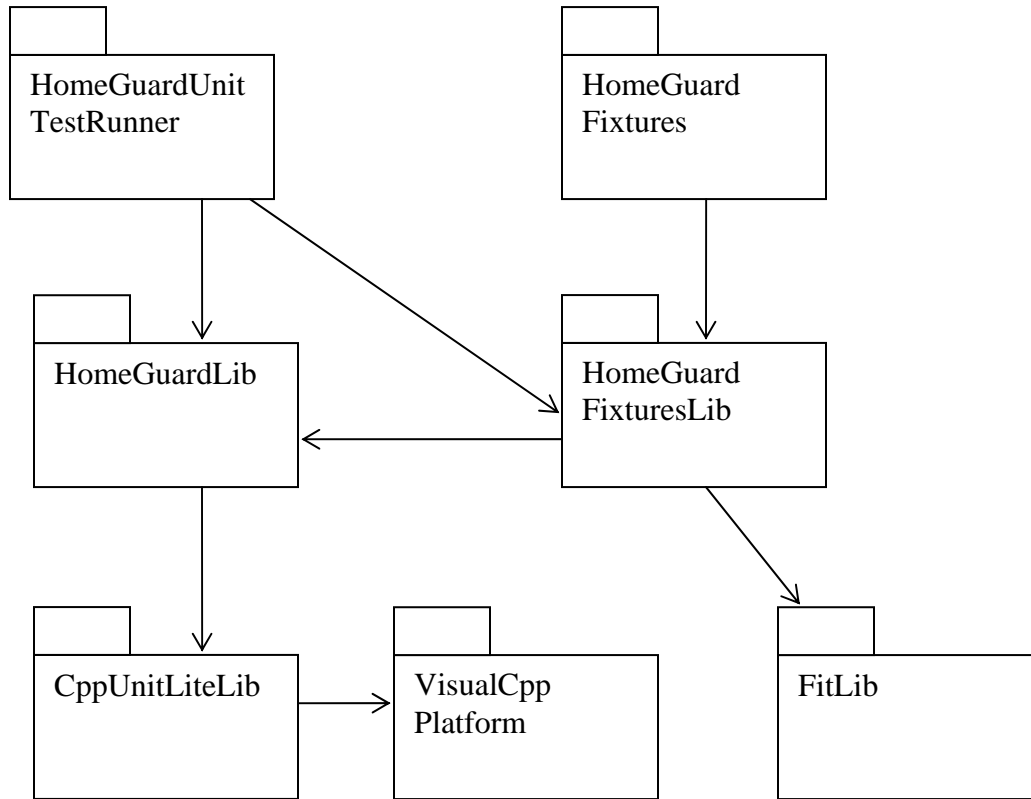
## **CppFit**

The Java version of FIT relies on reflection to access the members of the Fixture classes. Without reflection, CppFit uses macros and hand crafted code to make up for the deficiency. Fixtures can be rather tedious to setup. Fixtures for an application are linked into a DLL. The FitServer dynamically loads the DLLs when it is looking for fixtures to create. See the HomeGuard example.

FitNesse needs to see a COMMAND\_PATTERN to know how to find the C++ FitServer and DLLs. Most likely you will need a BAT file that the COMMAND\_PATTERN will reference. The BAT file will set up the PATH environment variable so that the DLL(s) containing the tests can be found and it will start the FitServer.exe. See the HomeGuard example. The HomeGuardTests fitness page holds the COMMAND\_PATTERN.

# HomeGuard C++ Acceptance Tests using FitNesse

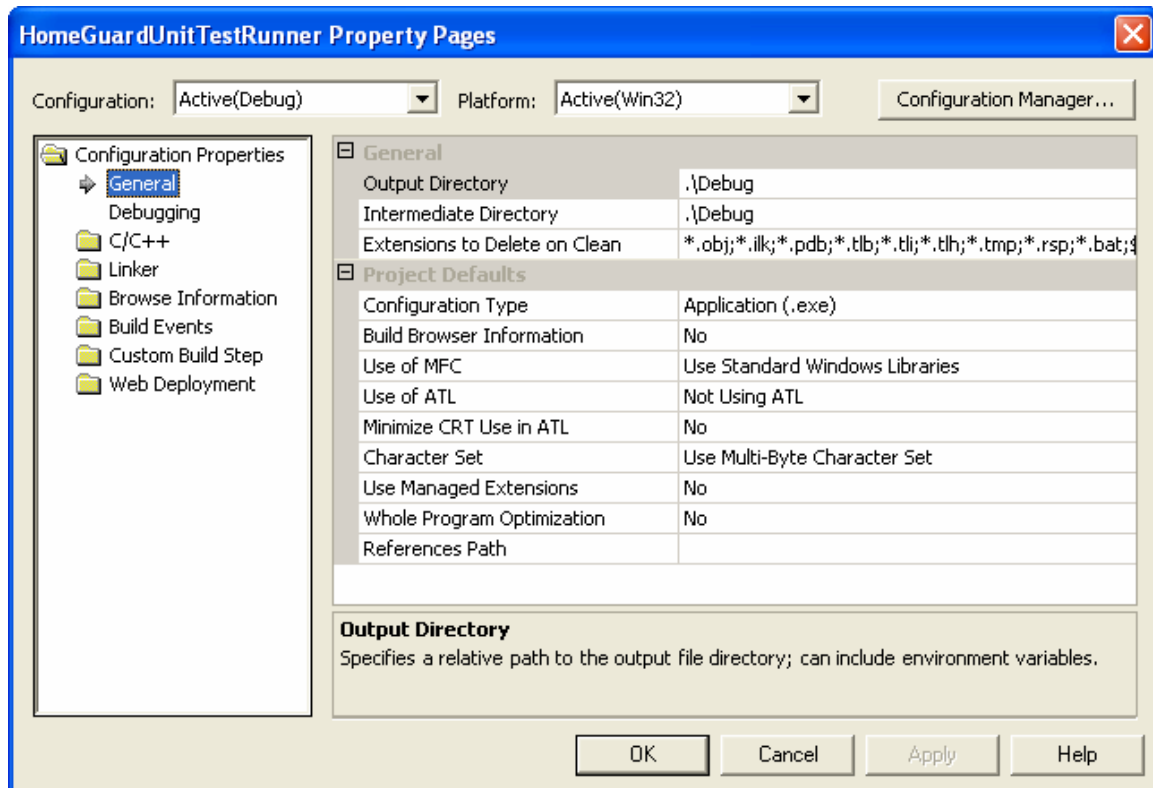
## Package Structure



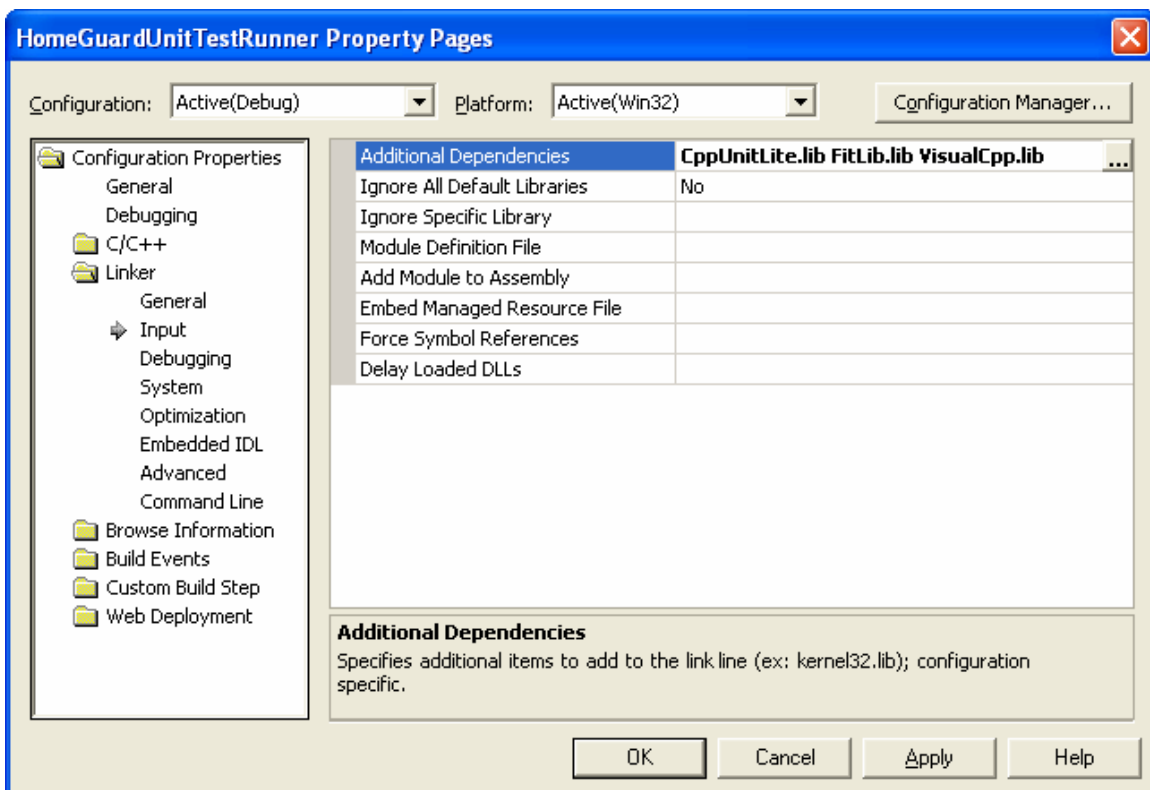
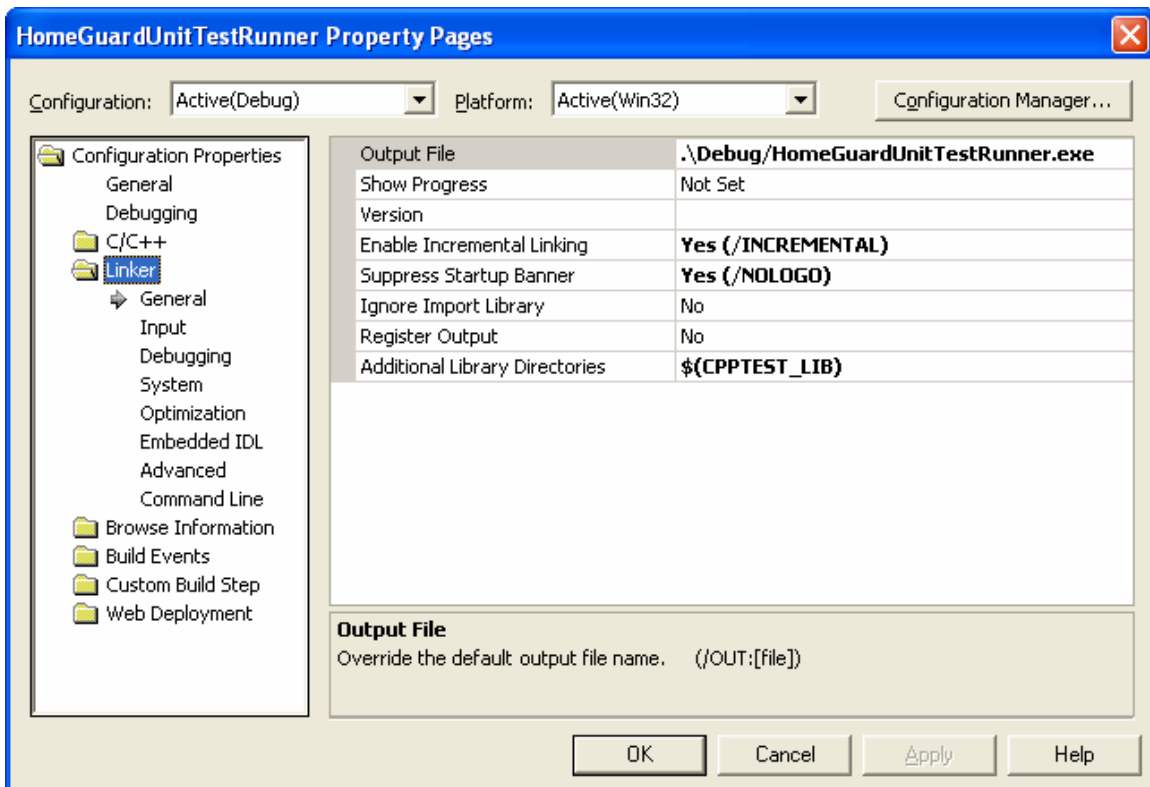
Package	Description
HomeGuardUnitTestRunner	The product of this package is the main executable for running unit tests
HomeGuardLib	The product of this package is the library that contains the HomeGuard application and the unit tests for the application and fixtures
HomeGuardFixtures	The product of this package is the DLL that FitServer.exe will use to create the HomeGuard specific FIT test fixtures.
HomeGuardFixturesLib	The product of this package is the library that contains the HomeGuard specific FIT test fixtures
CppUnitLiteLib	This package represents the header files and library for the CppUnitTestLite framework
FitLib	This package contains the header files and library for the C++ version of FIT
VisualCppPlatform	This library contains the memory leak detection enabling implementation for Microsoft Visual C++.

These are the significant project settings for CppTestTools in VC++6:

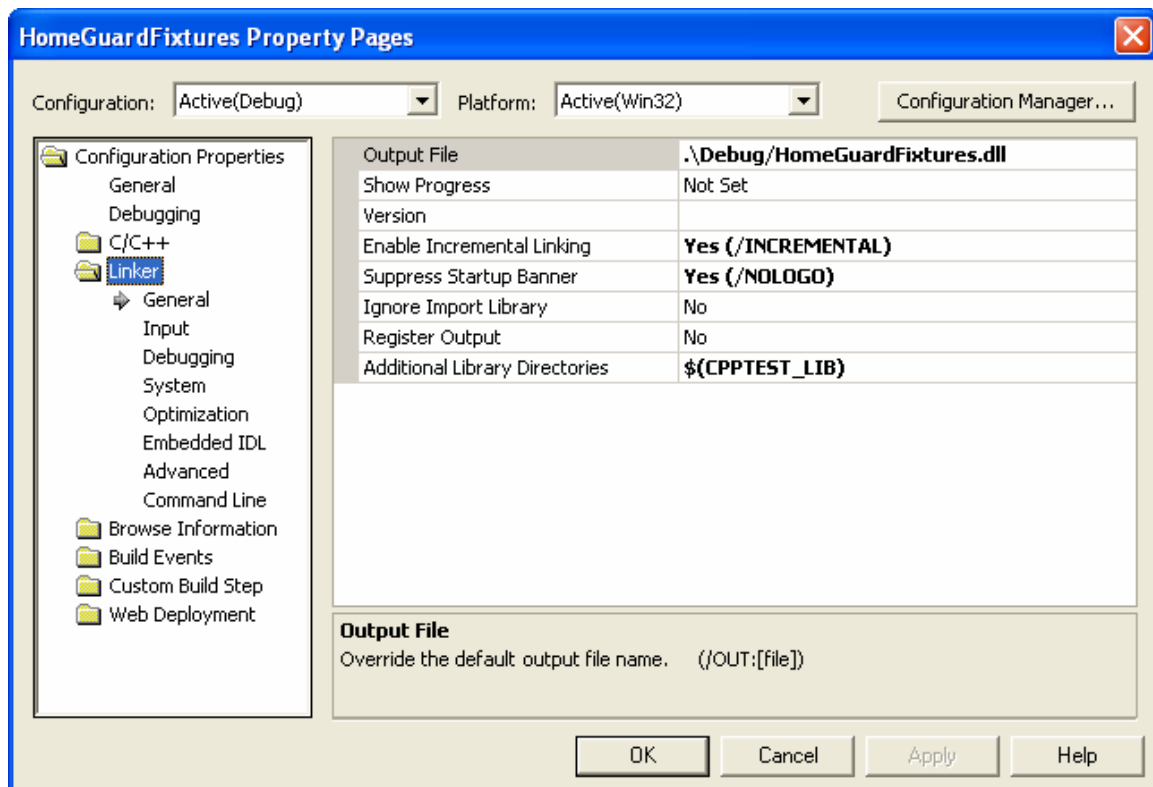
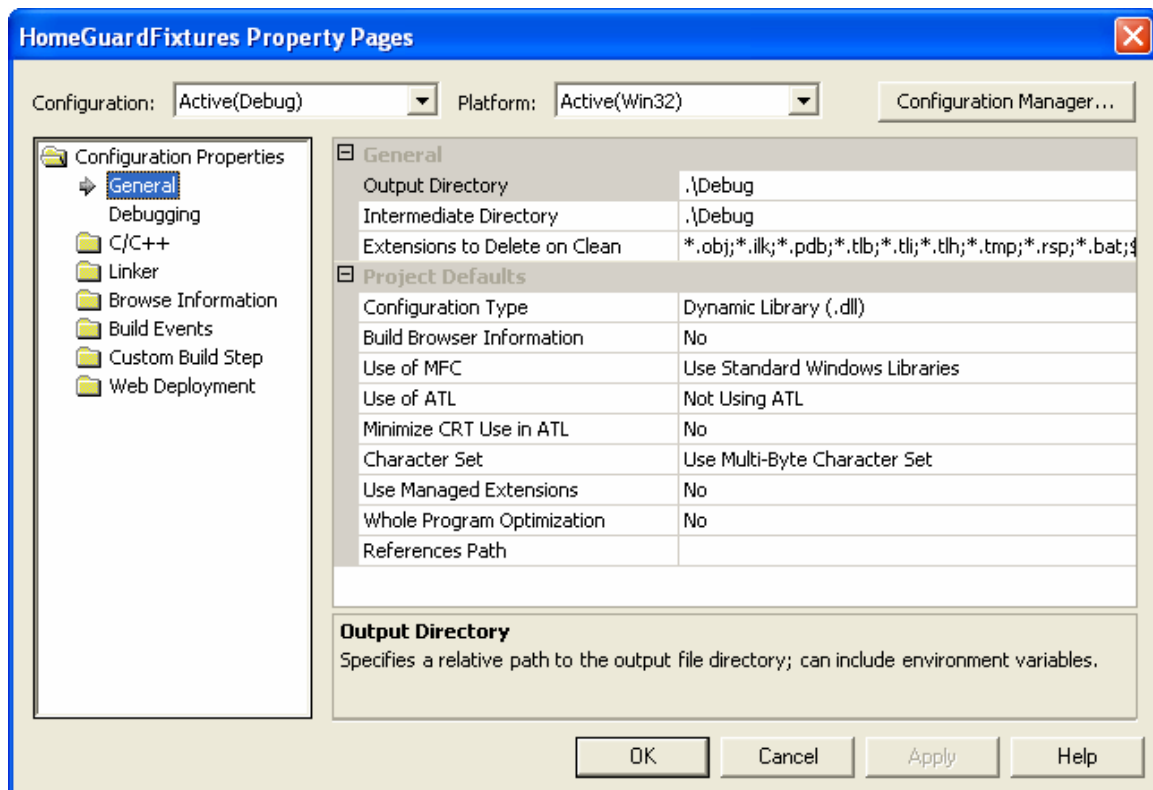
These are the significant project settings for HomeGuardUnitTestRunner win32 console application in VS.NET:

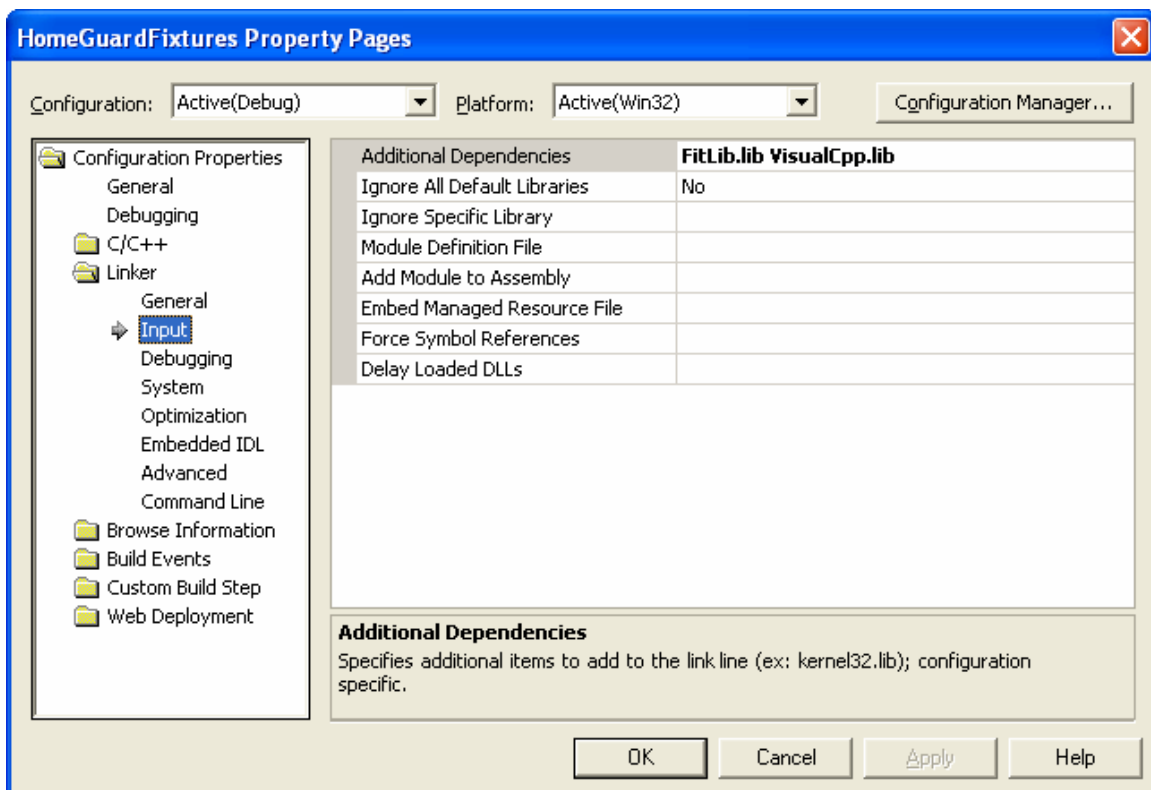






**These are the significant project settings for HomeGuardFixtures.dll in VS.NET:**





## Run unit tests and add a new Fitness test to HomeGuard

Following this procedure demonstrates writing and running a fixture, as well as the likely errors you may run into.

1. Build and run HomeGuardUnitTestRunner. All test pass.
2. Start FitNesse
3. Run the FitNesse test suite on HomeGuardTests. If you get an ErrorLog containing "java.io.IOException: CreateProcess:" from Fitnessse then you have to point the COMMAND\_PATTERN on the HomeGuardTests page to the HomeGuardAtRunnerBat.bat in your installation.
4. Run the suite again. You should not get an ErrorLog but should get a bunch of Fit exceptions.
5. Build HomeGuardFixtures and run the HomeGuardTest suite and get these results
 

<a href="#">PowerOn</a>	5 right, 0 wrong, 0 ignored, 0 exceptions
<a href="#">ArmSystem</a>	12 right, 0 wrong, 0 ignored, 0 exceptions

Add the DefineSensors test page under HomeGuardTest using this text:

!3 Define some sensors, make sure duplicate sensor IDs are rejected

```
! | HomeGuardFixtures.DefineSensors |  
| id | name           | submit() |  
| 1  | Paul's room      | true     |  
| 2  | Porch            | true     |  
| 3  | Front Door       | true     |  
| 3  | Back Door        | false    |
```

6. Run this page and get an exception because the DefineSensors fixture cannot be found.

## Create a new FIT fixture

7. Make the Fixture library the active project (e.g. HomeGuardFixturesLib)
8. Create a new fixture files (DefineSensors.h, DefineSensors.cpp, DefineSensorsTest.cpp).
9. Add the IMPORT\_TEST\_GROUP macro to your unit test runner main file (e.g. add IMPORT\_TEST\_GROUP(DefineSensors) to HomeGuardUnitTestRunner). This makes sure the any unit tests you write in the test file are linked into your test runner.
10. Make the unit test runner project the active project. Build it and run it.
11. Decide what kind of fixture you need
  - a. Fixture – useful for setup/tear down
  - b. ActionFixture – simulate events
  - c. ColumnFixture – submit table data or inspect function call return values
  - d. RowFixture – inspect tabular result data
12. Have your new fixture class inherit publicly from the needed fixture. In this case make ColumnFixture
13. When you get a million compile warnings, include Platform.h as your first include file. Get a clean compile.
14. Publish your new fixture into the fixtures DLL main function. (e.g. in the HomeGuardFixtures project add PUBLISH\_FIXTURE(DefineSensors). Don't forget to include DefineSensors.h)
15. Make the fixtures DLL project the active project and build the DLL
16. Run the fitness page and see that there is no exception on the top row of the table, but there are exceptions for each column.
17. Now define member variables or functions for each column of the appropriate type. The names must match what is in the FIT table.

```
int id;  
string name;  
  
string submit(){return "false"};
```

18. Get this to compile. If you run into trouble, do these one at a time.
19. Publish each member in the constructor of the fixture, like this:

```
DefineSensors::DefineSensors()  
{  
    PUBLISH(DefineSensors,int,id);  
    PUBLISH(DefineSensors,string,name);  
    PUBLISH(DefineSensors,string,submit);  
}
```

20. Get this to compile
21. Run the fitness page and you should have no exceptions. You will have an error for each row but the last because submit()'s return result is wrong.
22. Congratulations, your fixture is ready. Now implement the system behavior to allow the test to pass

## TDD Visual Studio Macro

The overhead to get a class under unit test in C++ is relatively large. In the CppSourceTemplates directory in the CppTestTools distribution is a VC++6 macro and three prototype source files. The macro can be installed into visual studio. You run it when you need to a new class. Three files will be installed into the current project: the .h, .cpp and a test file. You WILL have to adjust the macro to make sure it knows where the source prototype files are (see private function TemplatesDirectory). Be warned, that visual studio will crash if it does not find the files. If anyone knows how to fix this macro so that it won't crash let me know at [grenning@objectmentor.com](mailto:grenning@objectmentor.com).

## Credits

CppFit is based on a port of Ward Cunningham's Fit framework written in Java by Micahel Feathers.

CppUnitLite was originally written by Michael Feathers and extended by James Grenning

FitAdapter and FitServer were created by James Grenning, Micah Martin and Robert Martin

The HomeGuard example, this document and the project organization was developed by James Grenning with the support of the others mentioned hear.