Windows Mobile Support

In this section:

Introduction

The following topics explain how to use C++test to perform static analysis and unit testing on code that is designed to be compiled/built using Windows Mobile.

The C++test Visual Studio plugin directly supports Mobile SDKs for Window Mobile 5 and 6. No additional installation is required.

For testing Microsoft eMbedded Visual C++ 4.0 projects, please use the C++test standalone (Eclipse based) distribution.

Project Creation and Configuration

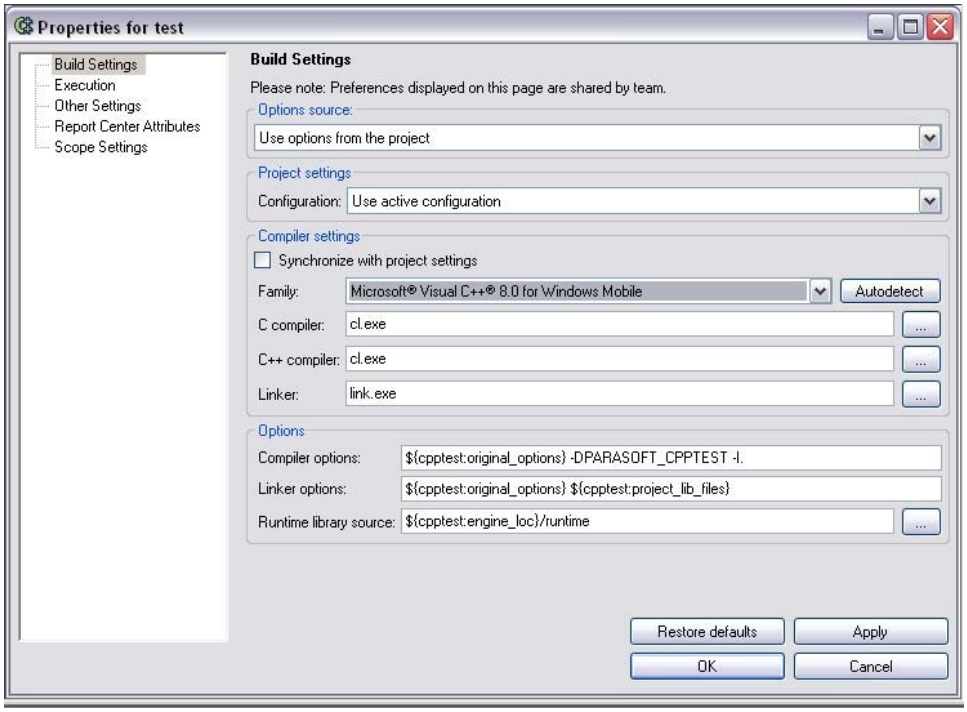
This topic explains how to configure Visual Studio projects for Windows Mobile to be tested with C++test.

A Microsoft Visual Studio project that is correctly configured for Windows Mobile/Windows CE does not require any additional setup for C++test static analysis.

For unit testing, the Runtime Library is required. It can be built automatically by C++test (the default) or manually from Visual Studio IDE or from a Makefile. To learn about the runtime library, see Working with the C++test Runtime Library.

To verify that the project setup is correct:

1. Right-click the Solution Explorer tree (a.k.a. "the project tree") node for the project whose settings you want to review and modify, then choose **C++test Properties** from the shortcut menu. The Properties dialog will open.
2. Verify that the proper compiler family is specified (in this case, Microsoft Visual C++ x.x for Windows Mobile).
    - In most cases, you can use the **Autodetect** button to automatically select the appropriate compiler configuration. If you need to use another compiler's binaries, clear the **Synchronize with project settings** check box, and then manually specify the appropriate location (s).



3. If you want to use an alternative project configuration (e.g., Debug, Release, etc.), choose it in the **Configurations** box.

For more information about C++test project properties, see Reviewing and Modifying Settings.

Static Analysis

This topic explains how to configure and run static analysis on code that is designed to be compiled/built using Windows Mobile.

Since static analysis runs on pure code only, information and settings related to linking and running the test object are not relevant to this kind of testing. However you must have all compiler settings set properly.

To perform static analysis:

1. Select the Solution Explorer node that represents the resource(s) you want to test (a single file, a selection of multiple files, or the entire project).
2. Start the analysis in one of the following ways:
    - From the **Parasoft** menu, choose **Test Using> [Preferred Test Configuration]**.
    - Open the pull-down menu for the **Test Using** toolbar button (this is a blue triangle), then choose your preferred static analysis test configuration.

After the analysis begins, C++test will collect the analysis options, calculate the scope of the analysis, and then start the static analysis. After the analysis is completed, the summary dialog and the static analysis results will display in the C++test output panel. You can now review and respond to results, as well as generate reports, as described in the following sections:

- Viewing Results
- Reviewing Static Code Analysis Results
- Reviewing BugDetective Static Analysis Results
- Understanding Reports

Learning More

For general information on performing static analysis with C++test, see Static Code Analysis and Flow Analysis.

Runtime Testing

This section explains how to configure and run runtime testing. It covers:

- Environment-Specific Configuration
- Unit Testing
- Application Monitoring

Environment-Specific Configuration

Automating Unit Testing with Microsoft ActiveSync

To facilitate using Microsoft ActiveSync as a communication channel during unit testing, an ASconnector command line tool is provided. ASconnector can automate the following tasks:

- Copying the test executable to the target or emulator.
- Starting the test executable remotely.
- Detecting when the unit test are finished.
- Copying log files from the target or emulator back to host.
- Launching the given emulator.

For details on ASconnector, see Unit Testing with ASconnector.

The "Build and run test executable for Windows Mobile/CE using ActiveSync" test flow was specifically designed to facilitate setting up unit tests with ActiveSync as a communication channel. ASconnector is invoked in the CustomStep in this flow. The CustomStep can be further customized to better suit your testing environment.

This flow supports unit testing with the actual target device, as well as with the emulator.

In this case, we are assuming that the target device (or emulator) is properly connected with the host, and the ActiveSync is up before running unit testing. This connection will be used by ASconnector.

This is an example of a CustomStep, which can be used in this scenario:

```
<CustomStep
        id="run_test"
        label="Running test executable..."
        commandLine="&amp;quot;ASconnector.exe&amp;quot;
                &amp;quot;--
        te=${cpptest:testware_loc}\${project_name}Test.exe&amp;quot;
                &amp;quot;--ta=arm&amp;quot;
                --testExecutableCmd
                --start-after=${cpptestproperty:test_case_start_number}"
                workingDir="${cpptest:testware_loc}"
/>
```

Unit Testing with ASconnector

ASconnector is a tool that allows you to automate unit testing on a target device or emulator using Microsoft ActiveSync as a communication channel.

ASconnector can be customized by the following options:

`ASconnector  [options]... [--ExecutableCmd ...]`

Option /Format	Short Equivalent	Description
--testExecutable=<path>	--te	Specifies the path to the target executable on the host machine. Mandatory option.
--logFile=<target_path>	--lf	Specifies a path to the log file on the target device. The log file is downloaded to the host after the test executable execution finishes. The default host location for downloaded log files is the working directory. The host location can be changed with the --logDir option. A comma-separated list of files can be issued. If this option is not present, standard log files are processed (cpptest_results.tlog and cpptest_results.clog).
--logDir=<path>	--ld	Specifies how long (in milliseconds) ASconnector will wait for the ActiveSync connection. If set to 0, there will be no timeout. Default value: 8000 milliseconds
--targetArch=<arch>	--ta	Target platform architecture. May be one of the following: arm, mips, sh, x86 or emu for emulator.
--printHelp	--help	Displays options summary.
--testExecutableCmd <...>	N/A	After this option, the rest of the command line is directly passed to the target executable.

The program flow consists of the following steps:

1. The target executable is copied from the host to the target executable.
2. If the program detects that the same version of the target executable is already copied (in the previous run of the test flow, for example) the executable is not copied. Timestamps are checked in order to detect this.
3. The target executable is started and testing begins.
4. When the testing finishes, the log files are copied back from the target system to the host.
5. Temporary log files are deleted.

Unit Testing

The following Test Configurations were designed specifically for use with Windows Mobile/Windows CE applications:

- **Build and Run Test Executable for Windows Mobile or Windows CE Using ActiveSync:** For building and running a test executable for Windows Mobile/Windows CE. ActiveSync is used as a communication channel. To use this flow, both host and target machines must support ActiveSync. The target can be an actual device connected in ways supported by ActiveSync, or it can be an emulator. For details, see Application Monitoring.
- **Build Test Executable for Windows Mobile:** For building a test executable for Windows Mobile/Windows CE. File communication is used. By default `"./Storage Card/cpptest_results.tlog"` and `"./Storage Card/cpptest_results.clog"` locations are used for storing test and coverage results.

- **Build and Run Test Executable for Pocket PC:** For building and running a test executable for Windows Mobile Pocket PC. File communication is used, through sharing the host directory as a Storage Card in the emulator. By default `"./Storage Card/cpptest_results.tlog"` and `"./Storage Card/cpptest_results.clog"` locations are used for storing test and coverage results. The test executable is generated to a subdirectory named "2577" to facilitate autostart after system boot. PPC_USA.BIN image is used from Windows Mobile SDK. When execution completes, close the emulator. C++test will then read and display test results.
- **Build and Run Test Executable for Smartphone:** For building and running a test executable for Windows Mobile Smartphone. File communication is used, through sharing the host directory as a Storage Card in emulator. By default `"./Storage Card/cpptest_results.tlog"` and `"./Storage Card/cpptest_results.clog"` locations are used for storing test and coverage results, respectively. The test executable is generated to a subdirectory named "2577" to facilitate autostart after system boot. SP_USA_GSM_QVGA_VR.BIN image is used from Windows Mobile SDK. When execution completes, close the emulator. C++test will then read and display test results.

The last three flows are intended for use with Microsoft Device Emulator. The following properties used in the Windows Mobile/Windows CE test flow definitions can be customized to adjust your flow definition to a particular environment:

- **emulator_path:** The path to the emulator executable file. By default, set to "C:\Program Files\Microsoft Device Emulator\1.0\DeviceEmulator.exe".
- **platform_dir:** The name of the subdirectory in which the executable will be generated. By default, set to "2577".
- **Image_path:** The path to the emulator Windows CE image file. By default (for Pocket PC), set to "C:\Program Files\Windows Mobile 5.0 SDK R2\PocketPC\DeviceEmulation\0409\PPC_USA.BIN".
- **Additional_parameters:** Additional command-line parameters for the emulator. By default (for Pocket PC), set to "/VMID {d1a7b239-28d5-4485-9b8c-5764e3dc79b6} /memsize 128".
- **skin:** The path to the skin xml file. By default, (for Pocket PC) set to "C:\Program Files\Windows Mobile 5.0 SDK R2\PocketPC\DeviceEmulation\Pocket_PC\Pocket_PC.xml".

Learning More

For general information on generating and executing unit test cases with C++test, Test Creation and Execution.

Application Monitoring

The following Test Configurations were designed specifically for use with Windows Mobile/Windows CE in Application Monitoring Mode:

- **Build and Run Application with Memory Monitoring for Windows Mobile or Windows CE Using ActiveSync:** For building and running an application for Windows Mobile/Windows CE. ActiveSync is used as a communication channel. To use this flow, both host and target machines must support ActiveSync. The target can be an actual device connected in ways supported by ActiveSync, or it can be an emulator. For details, see Application Monitoring.
- **Build and Run Application with Memory Monitoring for Pocket PC:** For building and running an application for Windows Mobile Pocket PC. File communication is used, through sharing the host directory as a Storage Card in the emulator. By default, `./Storage Card/cpptest_results.tlog` and `./Storage Card/cpptest_results.clog` locations are used for storing test and coverage results. The test executable is generated to a subdirectory named 2577 to facilitate autostart after system boot. The PPC_USA.BIN image is used from Windows Mobile SDK. When you finish testing your application, close the emulator. C++test will then read and display test results.
- **Build and Run Application with Memory Monitoring for Smartphone:** For building and running an application for Windows Mobile Smartphone. File communication is used, through sharing host directory as a Storage Card in the emulator. By default `./Storage Card/cpptest_results.tlog` and `./Storage Card/cpptest_results.clog` locations are used for storing test and coverage results, respectively. The test executable is generated to a subdirectory named 2577 to facilitate autostart after system boot. The SP_USA_GSM_QVGA_VR.BIN image is used from Windows Mobile SDK. When you finish testing your application, close the emulator. C++test will then read and display test results.

The last two flows are intended for use with Microsoft Device Emulator. You can customize them like you can customize other unit testing flows.

Learning More

For general information on performing application monitoring and runtime error detection with C++test, see Runtime Error Detection.