Assembly Code Coverage

In addition to industry standard C/C++ code coverage metrics, C++test can collect code coverage at the assembly level (also called object code coverage), which provides visibility into which sections of assembly code were covered during test execution, as well as details on branching points execution. Assembly code coverage is supported for the following compilers:

- Green Hills Software Compiler for PPC v. 3.5.x
- Green Hills Software Compiler for PPC v. 4.0.x
- Green Hills Software Compiler for PPC v. 4.2.x
- Green Hills Software Compiler for PPC v. 2017.1.x

C++test is preconfigured to collect assembly level coverage data from Green Hills software PPC simulator. If you wish to configure collecting object coverage data from on-target tests execution please contact Parasoft Support for more information.

Unlike C/C++ code coverage, assembly coverage results are not integrated with C++test views. Reports are available as external html, xml or cvs files. See Generating the Assembly Coverage Report for more details.

Configuring Project Settings

Assuming that your project is already prepared for unit tests or application memory monitoring, there are no additional configuration activities required to prepare it for assembly coverage monitoring. Note, however, that the project must be setup for a supported compiler.

You can exclude test-harness initializers (added by C++test) from the assembly coverage report. Go to **Project Properties> Parasoft> C++test> Build Settings> Compiler Options** and add the following option:

```
-DCPPTEST_USE_GLOBAL_OBJECTS_TO_INIT_RUNTIME=0
```

It will disable adding test-harness initializers to compilation units under test. Instead,C++test will generate a separate "test runner" for performing test-harness initialization.

Collecting Assembly Code

C++test ships with a built-in test configuration for executing unit test cases with assembly coverage monitoring. To view the test configuration settings:

1. Choose **Parasoft> Test Configurations**
2. Choose **Builtin> Embedded Systems> Green Hills Software> Run GHS Tests with Assembly Coverage Monitoring**

Coverage is stored in tested program memory buffers, and when the scheduled tests are executed and test executable exits, collected coverage information is saved via the defined communication channel (typically, directly to the file). Buffers with coverage information may become corrupted or may not be saved if, for example, the test executable crashes or the memory becomes corrupted. We recommend reviewing all unit test cases and exclude the ones that may cause application crash from the testing session.

The following Test Execution Flow properties are available in the test configuration for assembly coverage reports configuration (also see Generating the Assembly Coverage Report):

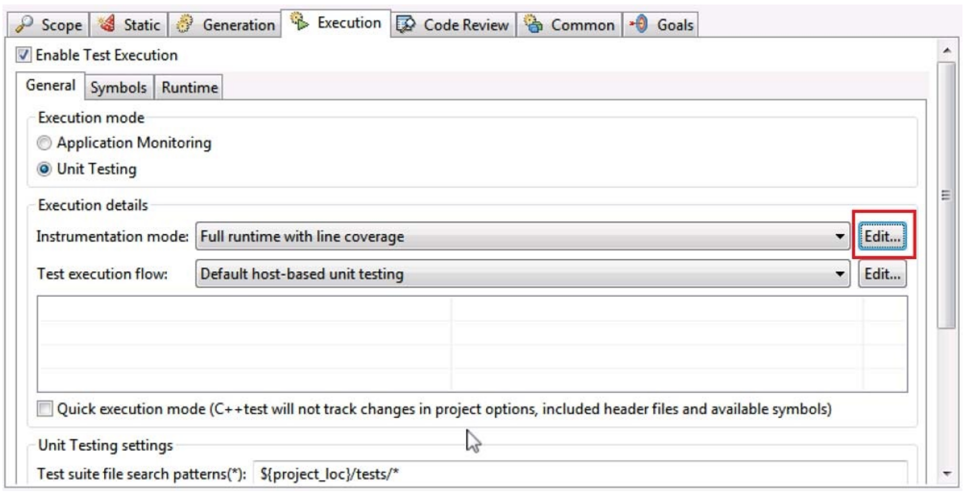
Property Name	**Values**
Assembly coverage report format	Available formats: html, xml, txt, flattxt, csv html is a default value
Assembly coverage report encoding	Encoding used for html reports. default is UTF-8

Contact Parasoft Support for more information if you want to collect object coverage data from on-target tests execution.

Enabling Assembly Code Coverage for User-defined Test Configuration

You can duplicate the built-in Run GHS Tests with Assembly Coverage Monitoring test configuration (right-click the on the test configuration and choose **Du plicate**) as a base for project specific customizations.

1. Choose **Parasoft> Test Configurations** and open the test configuration settings
2. Choose **Execution> General** tab and click **Edit** to open the Instrumentation mode options T



☑ Enable Test Execution

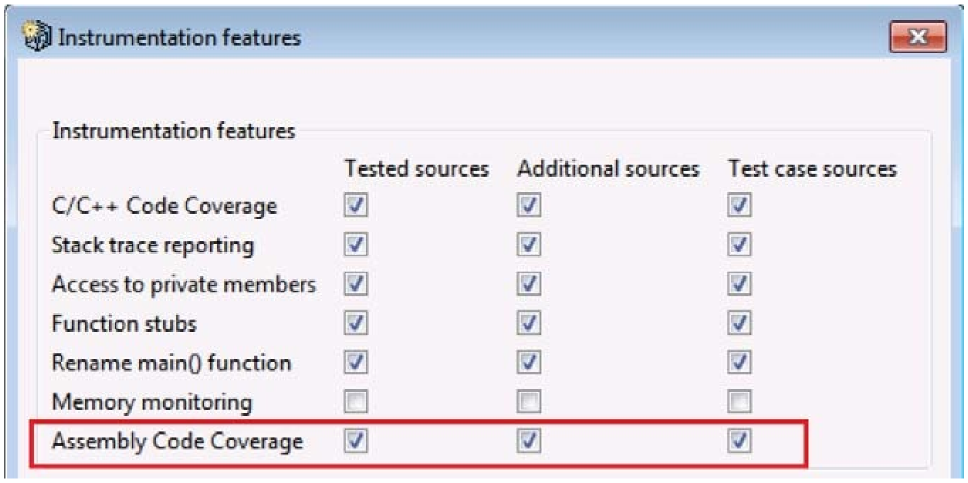
Execution mode
○ Application Monitoring
● Unit Testing

Execution details
Instrumentation mode: Full runtime with line coverage [Edit...]
Test execution flow: Default host-based unit testing [Edit...]

☐ Quick execution mode (C++test will not track changes in project options, included header files and available symbols)

Unit Testing settings
Test suite file search patterns(*): ${project_loc}/tests/*

3. Enable the Assembly Code Coverage sources in the Instrumentation features section and save your changes to the configuration

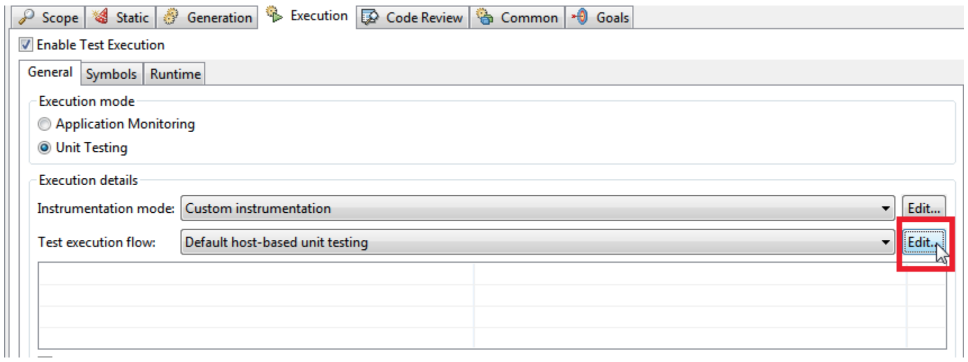

Instrumentation features

Instrumentation features	Tested sources	Additional sources	Test case sources
C/C++ Code Coverage	☑	☑	☑
Stack trace reporting	☑	☑	☑
Access to private members	☑	☑	☑
Function stubs	☑	☑	☑
Rename main() function	☑	☑	☑
Memory monitoring	☐	☐	☐
Assembly Code Coverage	☑	☑	☑

4. Run the test configuration and assembly code coverage will be collected during test execution

In addition to enabling assembly level code instrumentation you also need to modify the test execution flow to include a dedicated step for assembly coverage report generation. This step is already configured in the built-in Run GHS Tests with Assembly Coverage Monitoring test configuration.

1. Choose **Parasoft> Test Configurations** and open the test configuration settings
2. Choose **Execution> General tab** and click **Edit** to open the Test execution flow editor



☑ Enable Test Execution

Execution mode
○ Application Monitoring
● Unit Testing

Execution details
Instrumentation mode: Custom instrumentation [Edit...]
Test execution flow: Default host-based unit testing [Edit...]

3. Locate the following test execution flow steps in the editor and click after the closing bracket to create an insertion point:

```
<ReadTestLogStep
             testLogFile="${cpptest:testware_loc}/
cpptest_results.tlog"
             timeoutInfoProperty="test_exec_timeouted"
        />
        <ReadDynamicCoverageStep
                covLogFile="${cpptest:testware_loc}/
cpptest_results.clog"
        />
```

4. Insert the following flow step:

```
<GenASMToolReportFlowStep
        asmToolCmdPattern="$(asmReportTool) --generate-report=on --workspace=$(asmWorkspace)
--psrc=$(additionalOptions) --report-format=$(asmReportFor-mat) --results-directory=$(asmReportdir)
--project-name=$(asmProjectName)  --input-file=$(asmInputFile) --report-encoding=$(asmReportEncoding) --
psrc=$(advancedOp-
tionsFile)"
        asmReportFormat="${cpptestproperty:repor_format}"
        asmReportEncoding="${cpptestprop-erty:repor_encoding}"
        asmReportdir="${cpptestproperty:repor_loc}"
        asmInputFile="${cpptest:testware_loc}/${cpptestprop-erty:oc_result_file_name}"
/>
```

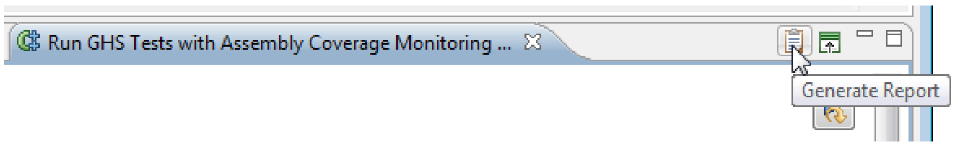
The `asmToolCmdPattern` attribute defines the command line for too that generates the report. This attribute uses the variables to facilitate report generation configuration. To make the variables editable via the test execution flow properties you can add the following at the begging of your test execution flow:

```
<SetProperty key="repor_format" value="html||xml||flat-
txt||txt||csv" uiEditable="true"
displayName="Assembly coverage report format" />
        <SetProperty key="repor_encoding" value="UTF-8" uiEdit-
able="true" displayName="Assembly
coverage report encoding" />
        <SetProperty key="report_loc"
value="${workspace_loc}/.cpptest/${project_name}/cpptestasm/
report-data" uiEditable="true"
displayName="Assembly coverage report location" />
```

Generating the Assembly Coverage Report

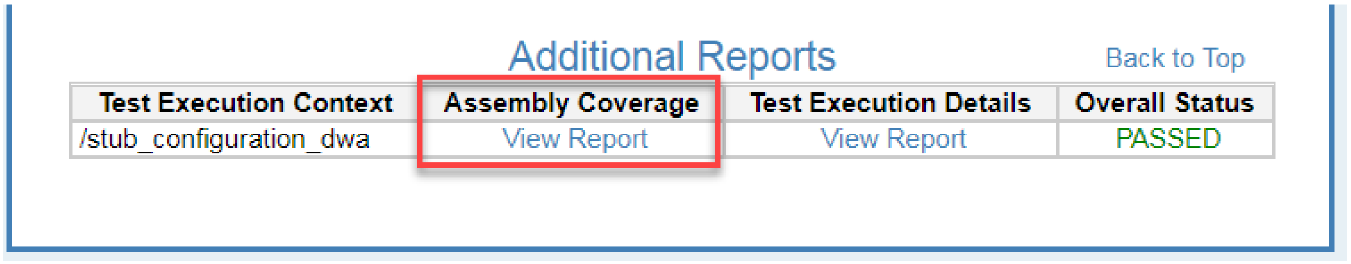
Assembly language coverage reports are generated after tests execution and are available for review in html, text or csv format. There is no assembly coverage data visible in C++test's coverage view, source code editor, or any other component of the C++test Eclipse or Visual Studio plug-in. If you are generating assembly coverage from a user-defined test configuration, make sure to configure the test execution flow described above.

1. After executing the test configuration, click **Generate Report** button.



2. Click **Preferences** and choose **HTML (C++test's Unit Testing details)** from the **Report Format** drop-down menu.
3. Apply your settings and click **OK** to exit the Preferences dialog
4. Click **OK** on the **Report & Publish** dialog.

The link to the assembly coverage report will be available in the **Assembly Coverage** column included in the **Additional Reports** section at the bottom of main report:



Additional Reports

Back to Top

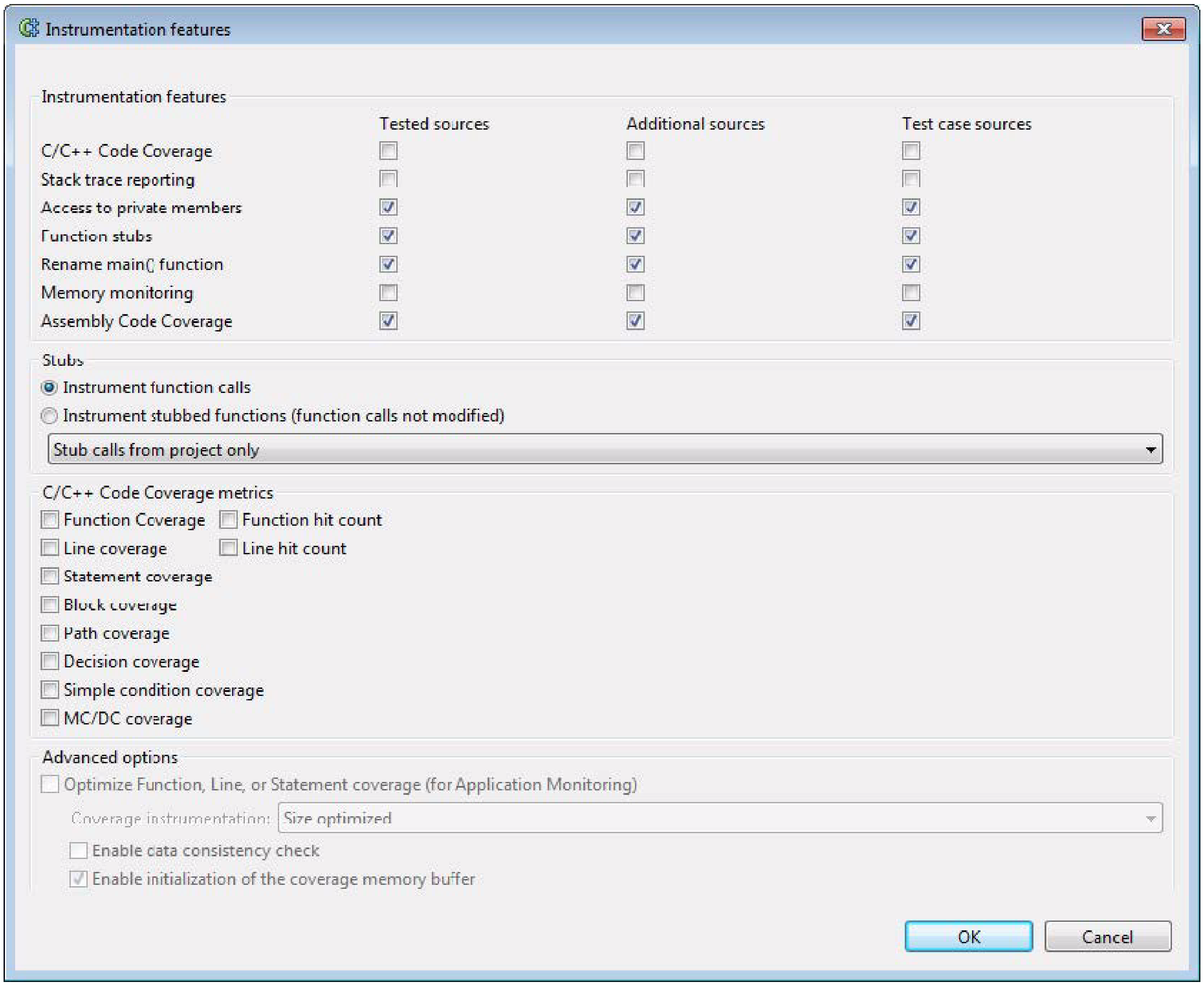
Test Execution Context	Assembly Coverage	Test Execution Details	Overall Status
/stub_configuration_dwa	View Report	View Report	PASSED

The HTML report links include relative paths that will not be broken when the reports are moved to another location.

See the table under Collecting Assembly Code Coverage, for Test Execution Flow properties that are available for assembly coverage reports configuration

Additional Notes About Assembly Coverage

- Assembly coverage metrics are only supported for select GHS PPC compilers (see Assembly Code Coverage). The C++test console will print a message to the console if you run assembly coverage monitoring with a project configured for an unsupported compiler.
- Assembly coverage is collected through assembler source code instrumentation generated from C/C++ instrumented code. You should not therefore enable other instrumentation features that will make C/C++ instrumented code significantly different from original code. The recommended configuration of instrumentation features is shown on the screen shot below.



Instrumentation features

Instrumentation features

	Tested sources	Additional sources	Test case sources
C/C++ Code Coverage	☐	☐	☐
Stack trace reporting	☐	☐	☐
Access to private members	✓	✓	✓
Function stubs	✓	✓	✓
Rename main() function	✓	✓	✓
Memory monitoring	☐	☐	☐
Assembly Code Coverage	✓	✓	✓

Stubs

[● Instrument function calls]

○ Instrument stubbed functions (function calls not modified)

Stub calls from project only

C/C++ Code Coverage metrics

☐ Function Coverage ☐ Function hit count
☐ Line coverage ☐ Line hit count
☐ Statement coverage
☐ Block coverage
☐ Path coverage
☐ Decision coverage
☐ Simple condition coverage
☐ MC/DC coverage

Advanced options

☐ Optimize Function, Line, or Statement coverage (for Application Monitoring)

Coverage instrumentation: Size optimized

☐ Enable data consistency check

✓ Enable initialization of the coverage memory buffer