Coverage measurement and

TEST REPORTS

Objectives

At the end of this section, you will be able to

- Expand tests to include code coverage
- Enable code coverage for microcontrollers without a hardware program trace interface
- Create a test report for the executed tests

testIDEA

Contents

TEST REPORTS

testIDEA

1 SOURCE COVERAGE TYPES

Typically, standards demand a more thorough testing approach as the risk to life the application poses increases. Code coverage, as will be covered in this unit, is one of the evidence metrics that is often required.

Code coverage is essentially recording the order of instructions that a microcontroller executed. The results display which sections of code were executed and which were not.

At the simplest level, it is enough to prove that all statements in the code were executed; at the highest, it must be proven that the logic of decision-making instructions were executed for all input combination that result in a change of the outcome.

- Statement coverage
- Function coverage
- Call coverage
- Decision coverage
- Branch coverage
- MC/DC – Modified Condition/Decision Coverage

Least Risk to Life

Greatest Risk to Life

2 CONFIGURING THE ANALYZER

Code coverage can be created for individual tests. However, it is typically more useful to enable it for a collection of unit tests and combine the results into a code coverage report for the suite of tests executed.

Code coverage is generated using the *Analyzer* feature of the iSYSTEM tools as we cover in this unit.

The process involves:

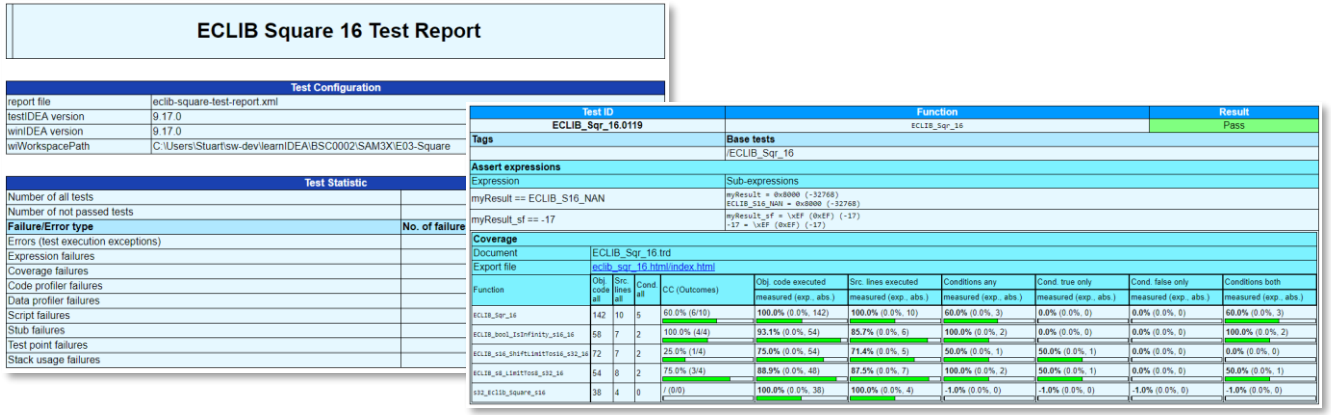
- Creating a base test with the default code coverage settings
- Modifying the first derived to ensure the program trace file starts empty
- Modifying the last derived test to close the file and generate the test report.

Configuration process for code coverage during Unit Test

1. Configure *Analyser* settings in base test
   - Define settings that are common to all tests
   - Configure modification of program trace file (**.TRD*) to *Append*

2. Modify first derived test to create clean program trace file

3. Modify final derived test to generate coverage report



ECLIB Square 16 Test Report

Test Configuration	
report file	eclib-square-test-report.xml
testIDEA version	9.17.0
winIDEA version	9.17.0
wiWorkspacePath	C:\Users\Stuart\sw-dev\learnIDEA\BSC0002\SAM3X\E03-Square

Test Statistic	
Number of all tests	
Number of not passed tests	
Failure/Error type	**No. of failure**
Errors (test execution exceptions)	
Expression failures	
Coverage failures	
Code profiler failures	
Data profiler failures	
Script failures	
Stub failures	
Test point failures	
Stack usage failures	

Test ID	Function	Result
ECLIB_Sqr_16.0119	ECLIB_Sqr_16	Pass

Tags	Base tests
	/ECLIB_Sqr_16

Assert expressions	
Expression	Sub-expressions
myResult == ECLIB_S16_NAN	myResult = 0x8000 (-32768) ECLIB_S16_NAN = 0x8000 (-32768)
myResult_sf == -17	myResult_sf = \xEF (0xEF) (-17) -17 = \xEF (0xEF) (-17)

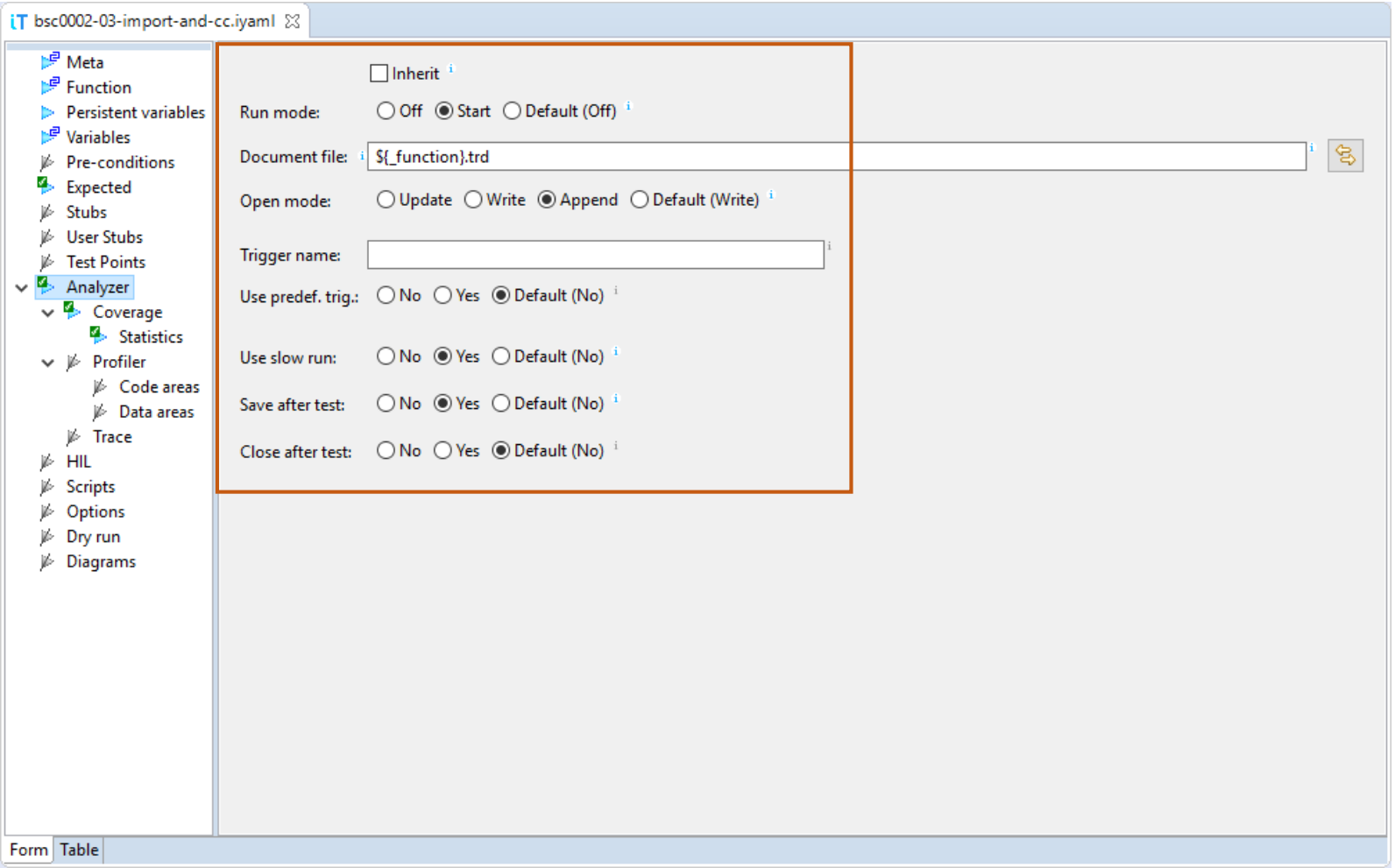
Coverage	
Document	ECLIB_Sqr_16.trd
Export file	eclib_sqr_16.html/index.html

Function	Obj. code all	Src. lines all	Cond. all	CC (Outcomes)	Obj. code executed measured (exp., abs.)	Src. lines executed measured (exp., abs.)	Conditions any measured (exp., abs.)	Cond. true only measured (exp., abs.)	Cond. false only measured (exp., abs.)	Conditions both measured (exp., abs.)
ECLIB_Sqr_16	142	10	5	60.0% (6/10)	**100.0%** (0.0%, 142)	**100.0%** (0.0%, 10)	**60.0%** (0.0%, 3)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**60.0%** (0.0%, 3)
ECLIB_bool_IsInFinity_s16_16	58	7	2	100.0% (4/4)	**93.1%** (0.0%, 54)	**85.7%** (0.0%, 6)	**100.0%** (0.0%, 2)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**100.0%** (0.0%, 2)
ECLIB_s16_ShiftLimitToS16_s32_16	72	7	2	25.0% (1/4)	**75.0%** (0.0%, 54)	**71.4%** (0.0%, 5)	**50.0%** (0.0%, 1)	**50.0%** (0.0%, 1)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)
ECLIB_s8_LimitToS8_s32_16	54	8	2	75.0% (3/4)	**88.9%** (0.0%, 48)	**87.5%** (0.0%, 7)	**100.0%** (0.0%, 2)	**50.0%** (0.0%, 1)	**0.0%** (0.0%, 0)	**50.0%** (0.0%, 1)
s32_Eclib_Square_s16	38	4	0	/ (0/0)	**100.0%** (0.0%, 38)	**100.0%** (0.0%, 4)	**-1.0%** (0.0%, 0)	**-1.0%** (0.0%, 0)	**-1.0%** (0.0%, 0)	**-1.0%** (0.0%, 0)

2 CONFIGURING THE ANALYZER

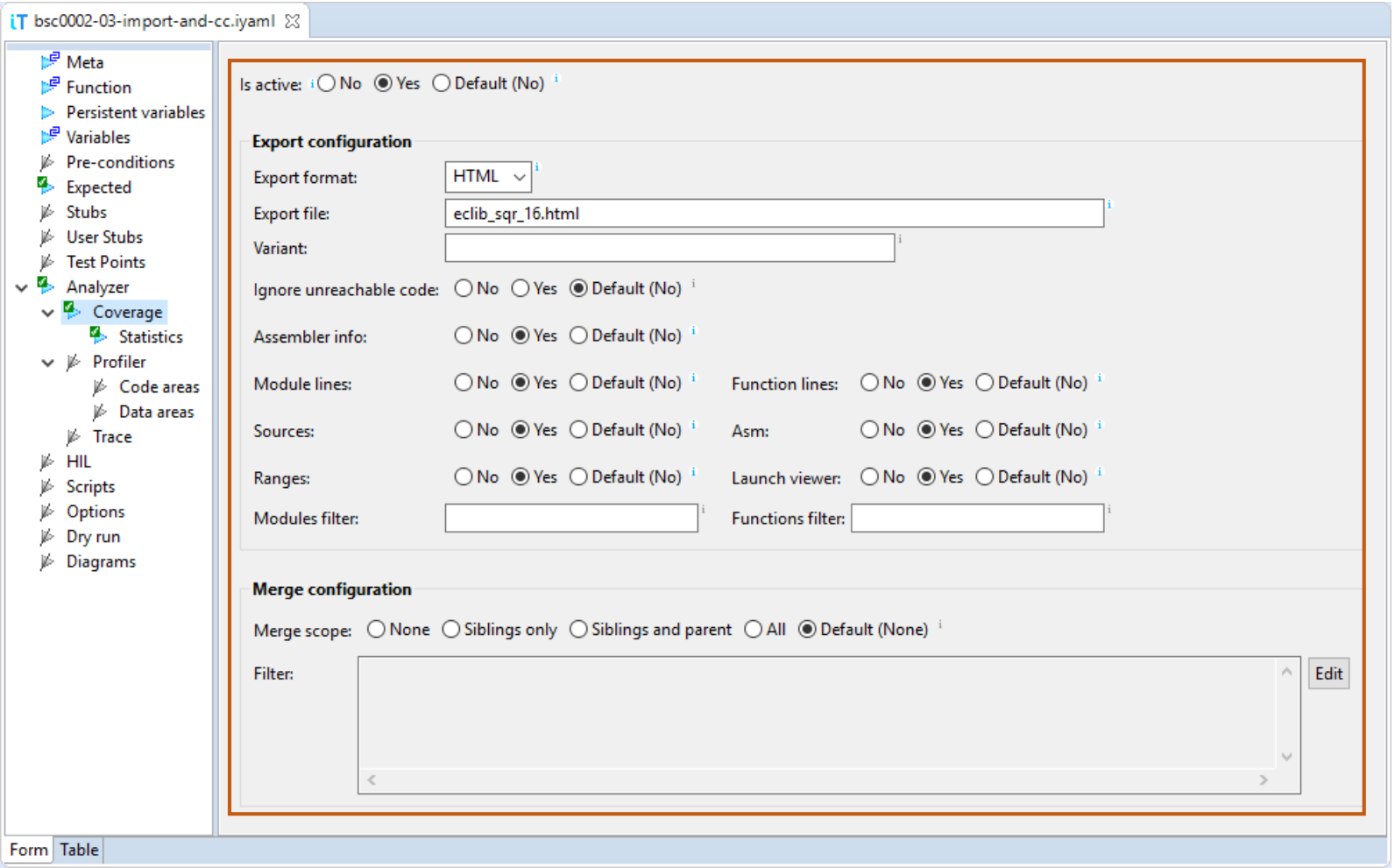
The *Analyzer* is responsible for configuring and enabling the collection of program trace data from the target (shown opposite).

The sub-options further determine how to configure the *Coverage* measurement and the resulting *Statistics* as will be covered on the following slides.



2 CONFIGURING THE ANALYZER

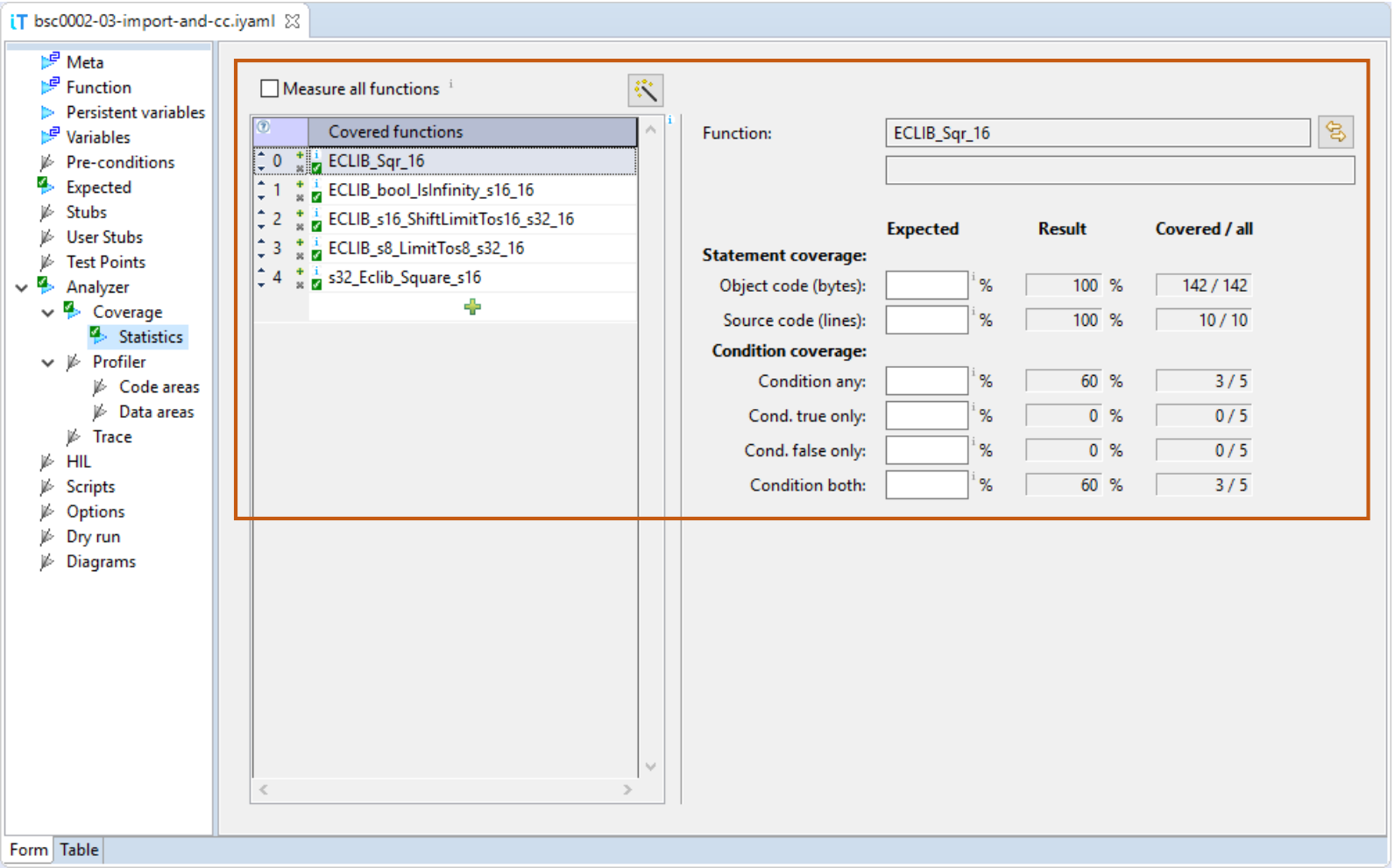
The *Coverage* is responsible for configuring and enabling coverage analysis calculations used to create the test report's coverage results (shown opposite).



2 CONFIGURING THE ANALYZER

The *Statistics* is responsible for configuring which functions should be included in the code coverage statistics output (shown opposite).

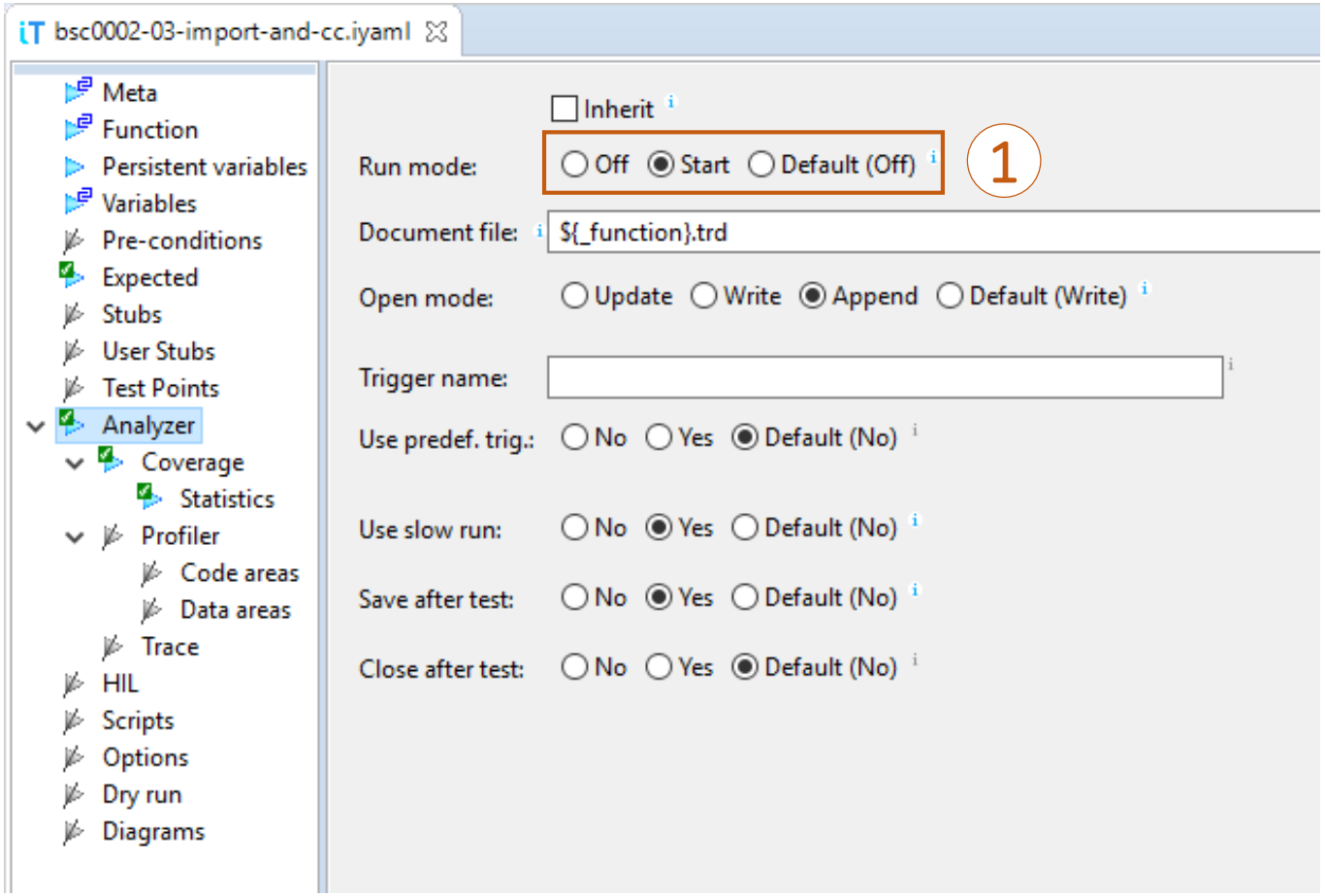
Additionally, further pass/fail criteria can be set in the form of minimum coverage values that must be achieved for the associated tests to be considered a ‘pass’. Thus, although the actual test vectors may pass, the tests will be considered to have failed if the desired minimum coverage has not been achieved.



2 CONFIGURING THE ANALYZER – BASE TEST

1 In the base test, set Analyzer Run Mode to Start

This simply turns on the collection of program trace for the target microcontroller. The next steps configure where the data should be stored and how to analyze it.



2 CONFIGURING THE ANALYZER – BASE TEST

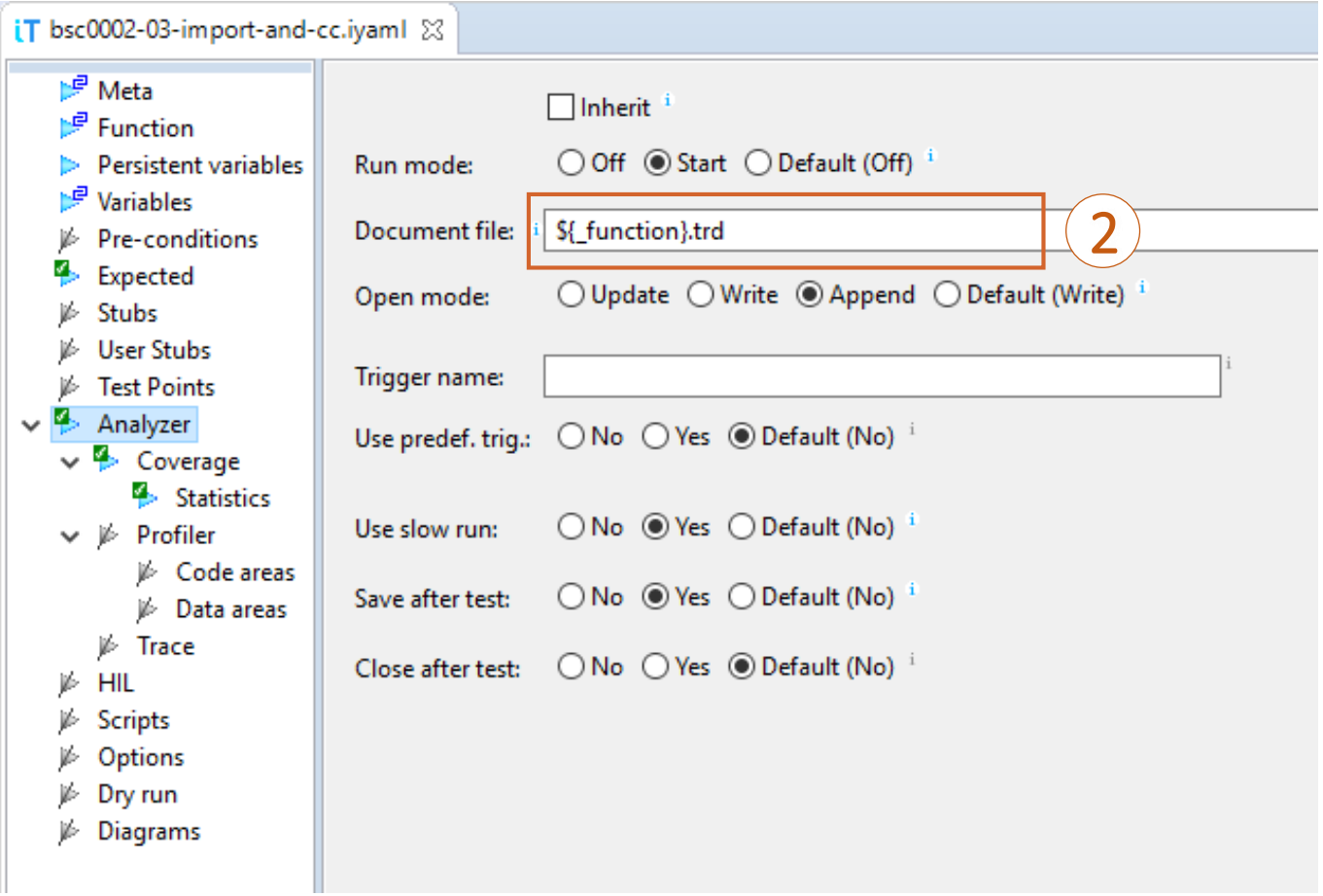
2 Provide Document file name where trace data is to be saved:

The *Analyzer* data is saved using iSYSTEM's *.TRD format files.

To maintain an overview of the data collected, we can define a distinct analyzer file name for the *.TRD files. These are entered into the "Document file" field.

By typing '$' into this field a list of possible tokens are offered. These tokens are replaced with the labels when the *.TRD file is generated.

For example, if the file name is defined as ${_testId}-${_function}.trd, the resulting file name will be created from the current test's ID and the name of function under test.

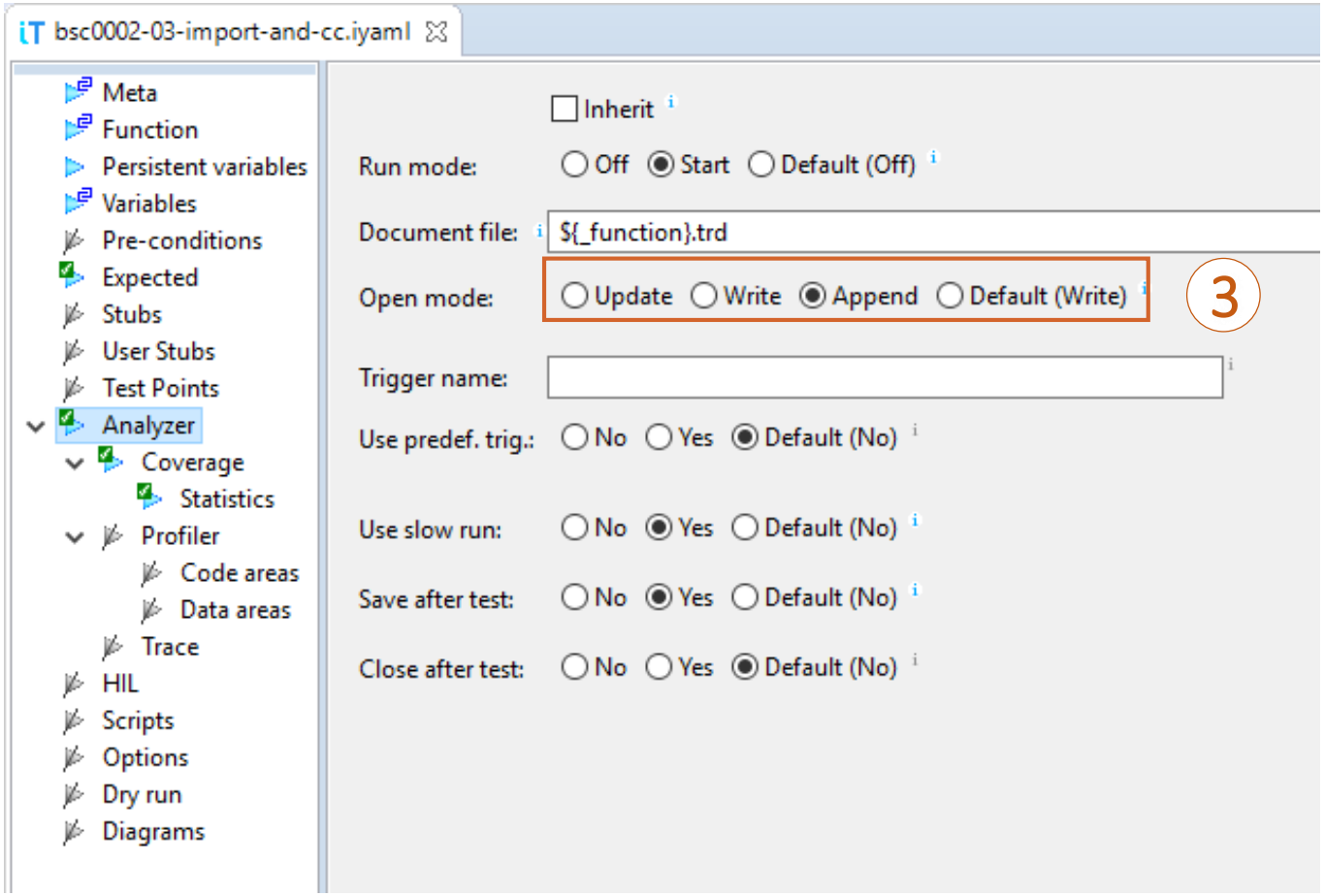


2 CONFIGURING THE ANALYZER – BASE TEST

3 Open mode - Append

To collect accumulated program trace data from several tests, the *Open mode* for the *.TRD can be configured to accumulate the results from several tests.

By selecting the *Open mode* as *Append* as shown opposite, all coverage results will be stored to a single file.



2 CONFIGURING THE ANALYZER – BASE TEST

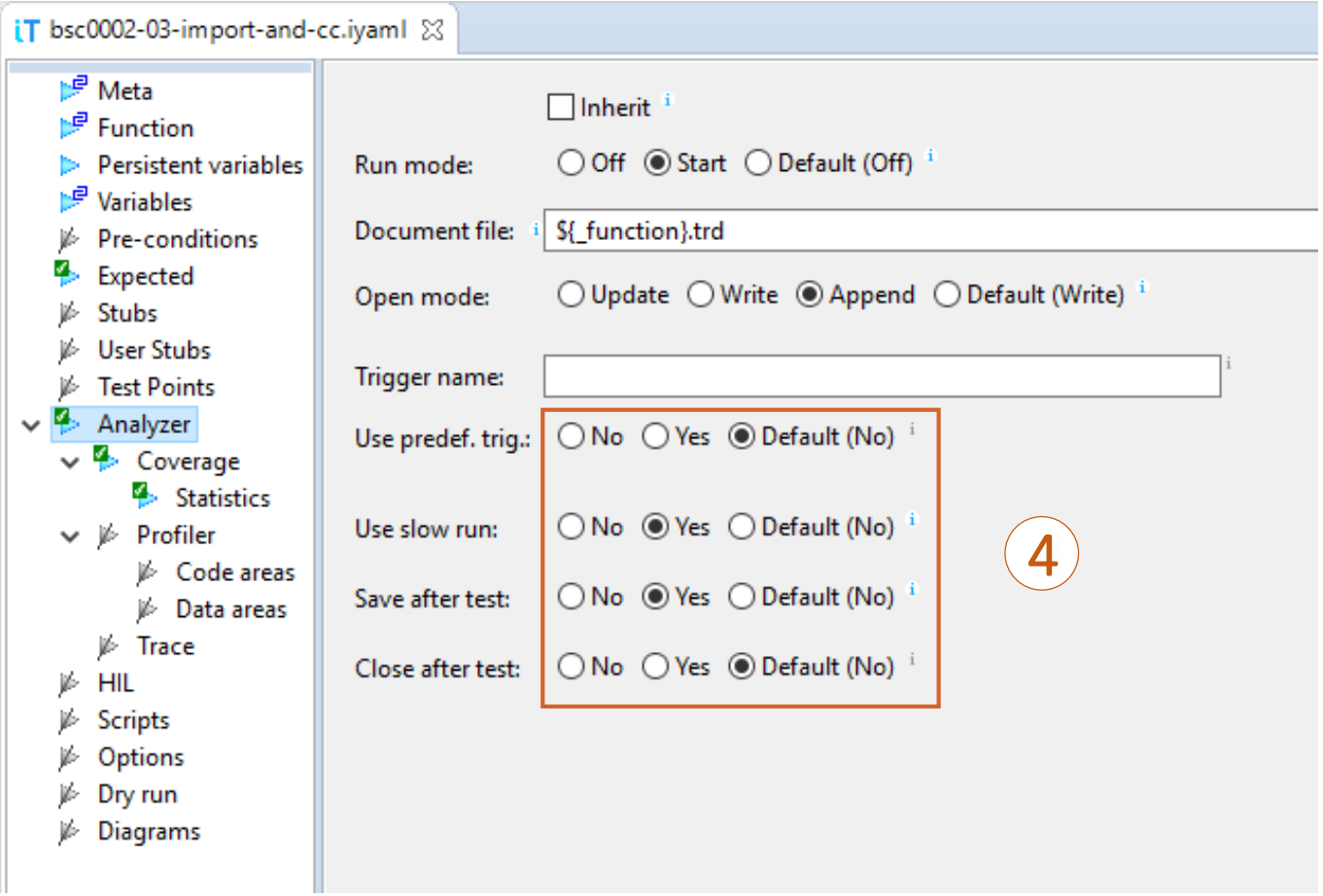
4 Further Settings of Analyzer

Slow run: Yes
Save after test: Yes
Close after test: No

If the target microcontroller does not have a hardware trace implementation, the *Slow Run* feature can be used to collect trace information.

The **.TRD* file is also saved and left open after each test as the common inherited settings for derived tests. The "left open" option simply means the **.TRD* file is left open in the *Analyzer* window within winIDEA.

These files can consume a large amount of RAM, so it is recommended to keep the number of files opened simultaneously to a minimum.

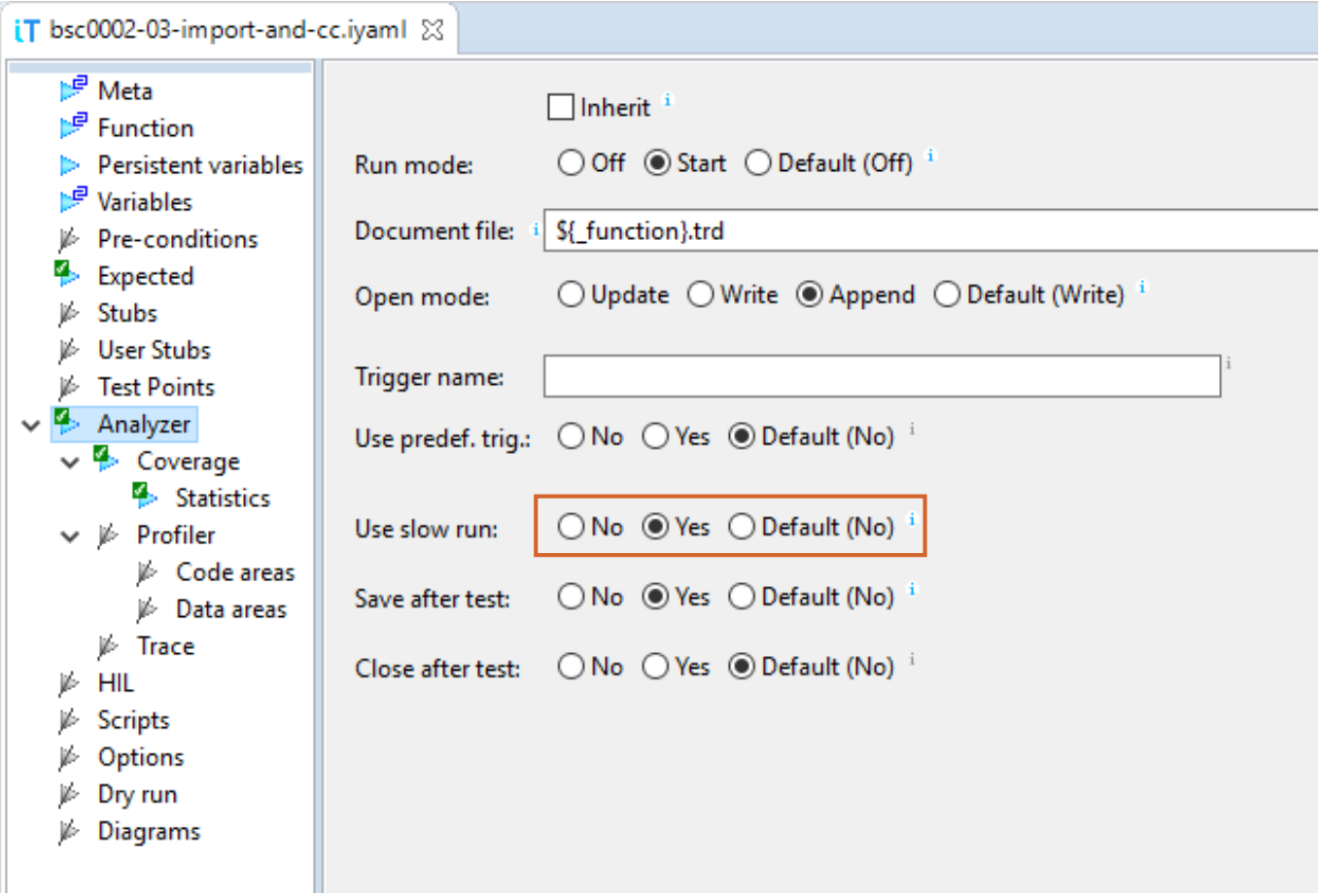


2 CONFIGURING THE ANALYZER – BASE TEST

Slow Run for MCUs without trace port:

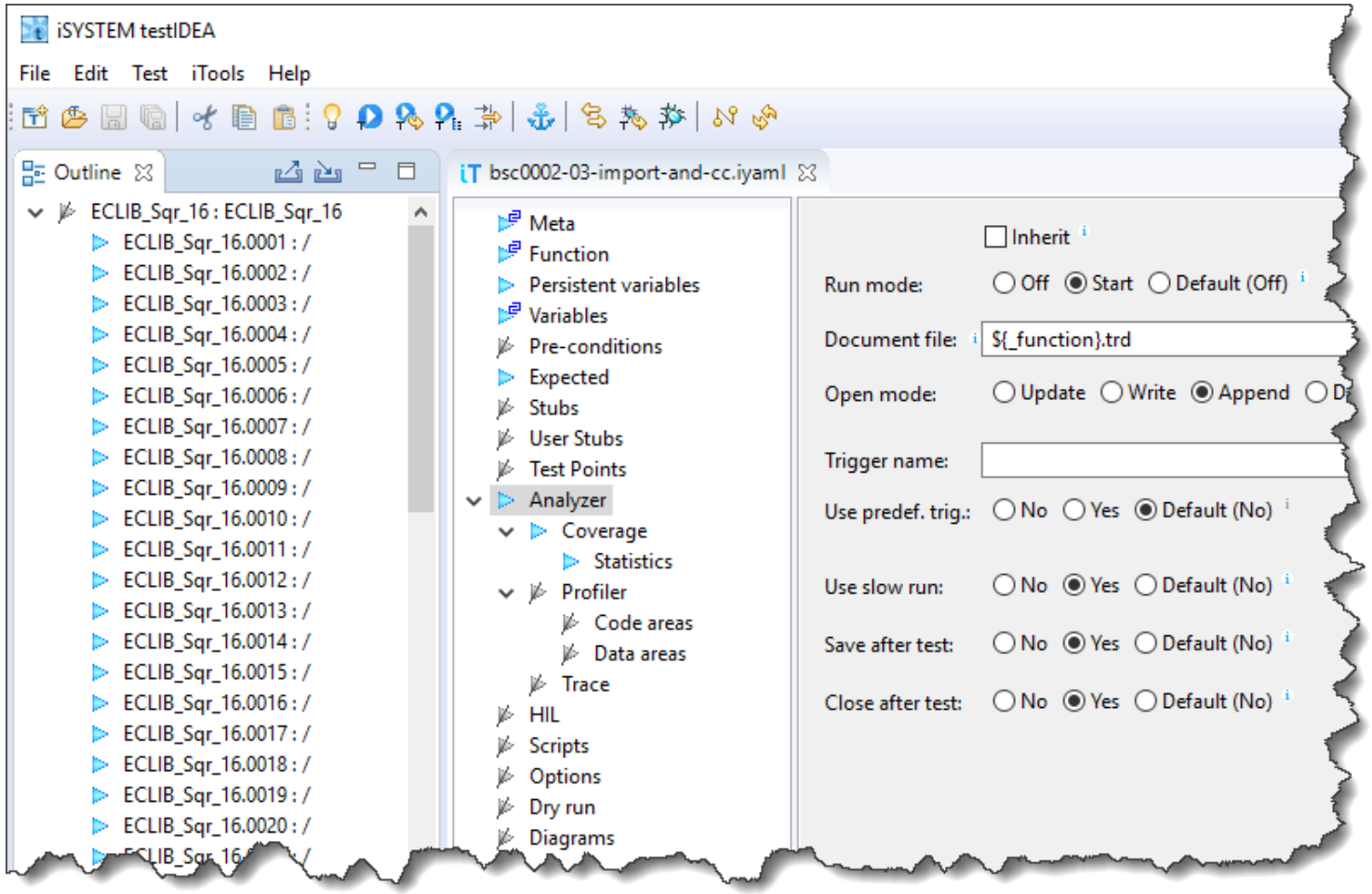
*As the name suggest, **Slow Run** executes the code slowly. Sometimes very slowly.*

*If the microcontroller has no hardware trace with which to collect the program trace data, **Slow Run** executes every assembly instruction individually, stopping after each instruction. In the context of Unit Tests, this should not be an issue as, at this level, we are checking software functionality, not hardware functionality or real-time performance.*



2 DEVELOP UNIT TESTS

Now that the base test is configured with the common *Analyzer* settings, the tests can be created, deriving them from the base test.

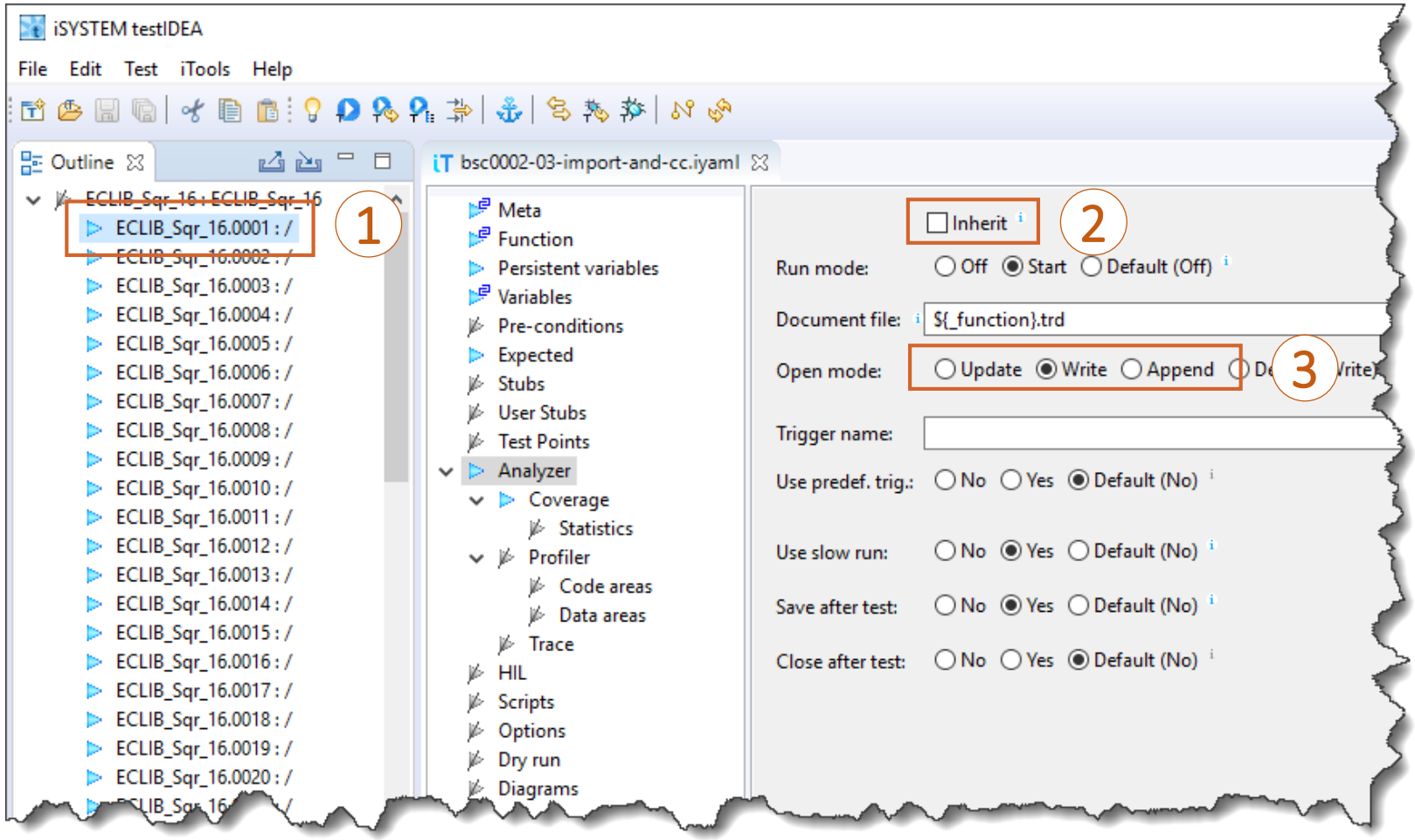


2 CONFIGURING THE ANALYZER – 1st DERIVED TEST

We will now configure the *Analyzer* settings so that the **.TRD* file is created anew prior to collecting trace data. This ensures that the only trace data in the file is from this suite of tests.

1. Select the first derived test.
2. Click the *Inherit* option twice to change to *Checked* and then *Unchecked*.
   This ensures the entered settings are retained.
3. Change the *Open mode* setting from *Append* to *Write*.

The trace data collected will now be appended to the freshly created **.TRD* file.



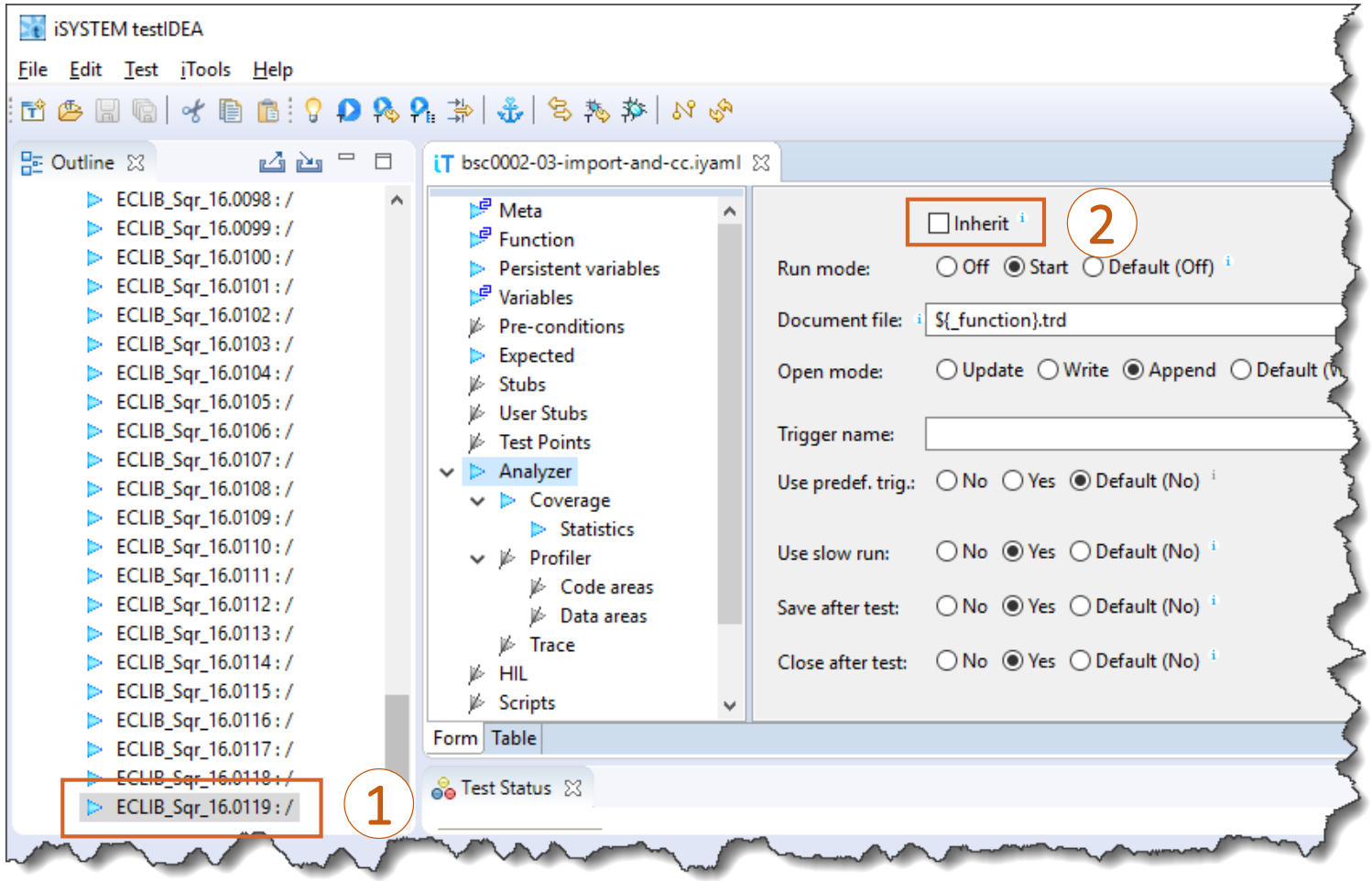
2 CONFIGURING THE ANALYZER – FINAL DERIVED TEST

The *Analyzer*'s *Inherit* option applies to the entire *Analyzer* setting **and** the sub-settings below it. In order to configure the last test to generate a code coverage report, we first need to turn off inheritance.

1. Select the last derived test.

2. Click the *Inherit* option twice to change to *Checked* and then *Unchecked*.
   This ensures the entered settings are retained.

Do not change any other settings here.



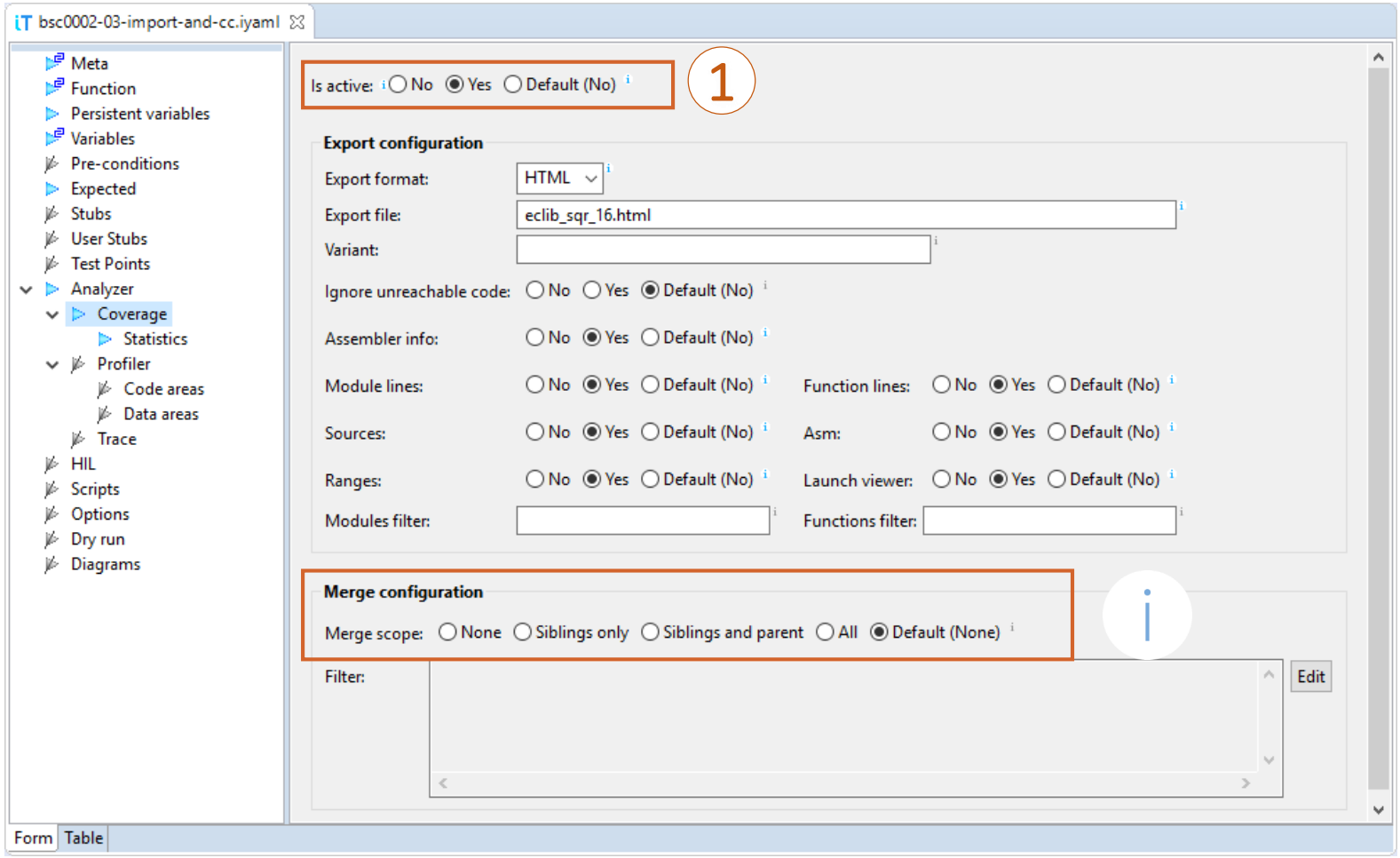
3 COVERAGE SETTINGS – FINAL DERIVED TEST

Now we can configure *Coverage*:

1 Activate the coverage settings via *Is active:* → yes

*Code Coverage can be calculated using two methods. Here we have collected all trace information into a single *.TRD file for analysis in the last test using the **append mode** of the Analyzer.*

*The **Merge configuration** option in Coverage can alternatively be used if each unit test stored its trace data in single *.TRD files. The only difference is between generating one large *.TRD file, or generating many small *.TRD files.*



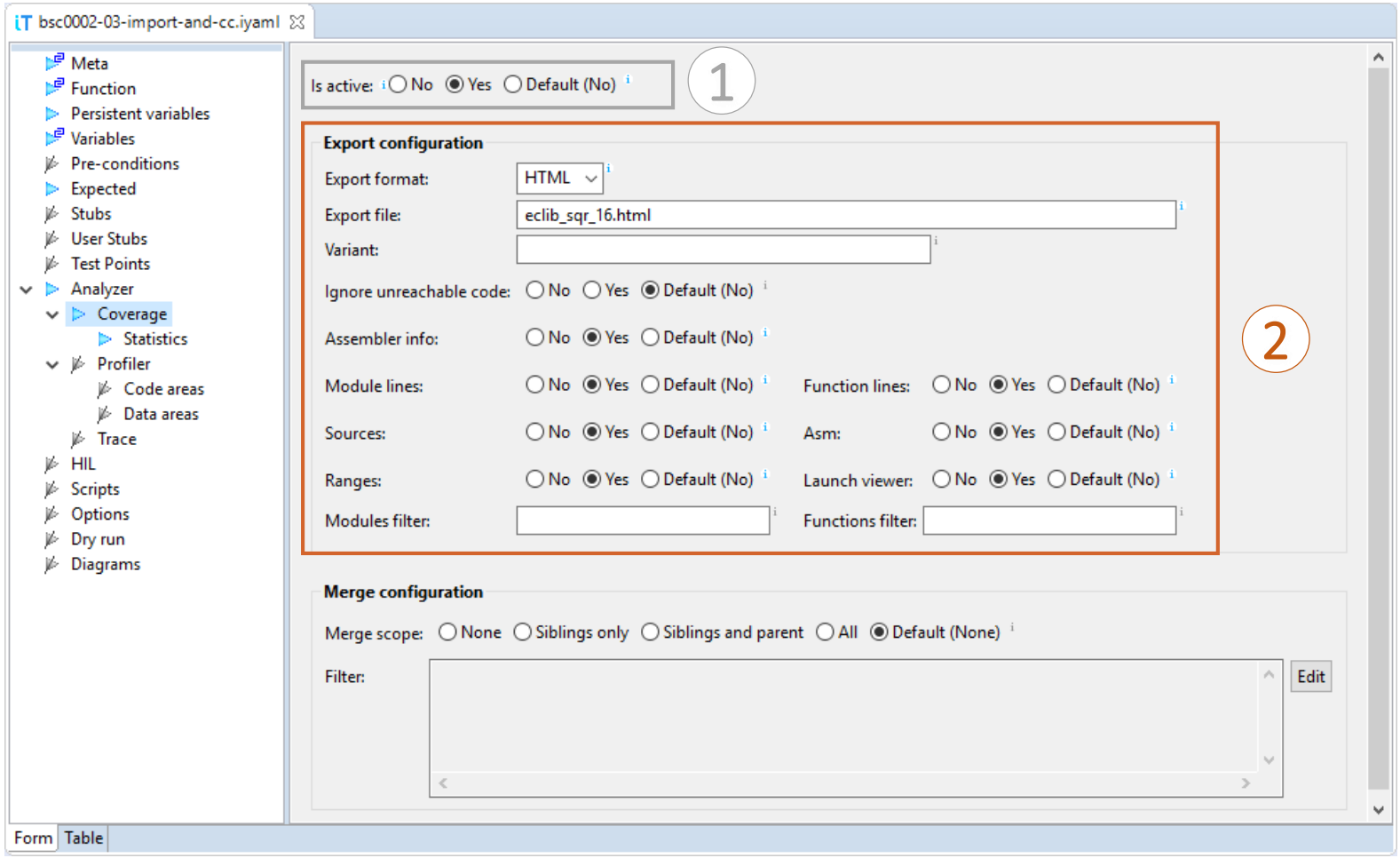
3 COVERAGE SETTINGS – FINAL DERIVED TEST

Now we can configure *Coverage*:

1 Activate the coverage settings via *Is active:* → yes

2 Here we have defined the report output format to be *HTML*, and have provided a file name in the *Export files* field. In addition, we can determine exactly which information is required in the coverage report.

The *Launch Viewer* option is set to *Yes*. This ensures that the results are displayed in the selected HTML format in the default browser upon completion of the tests.



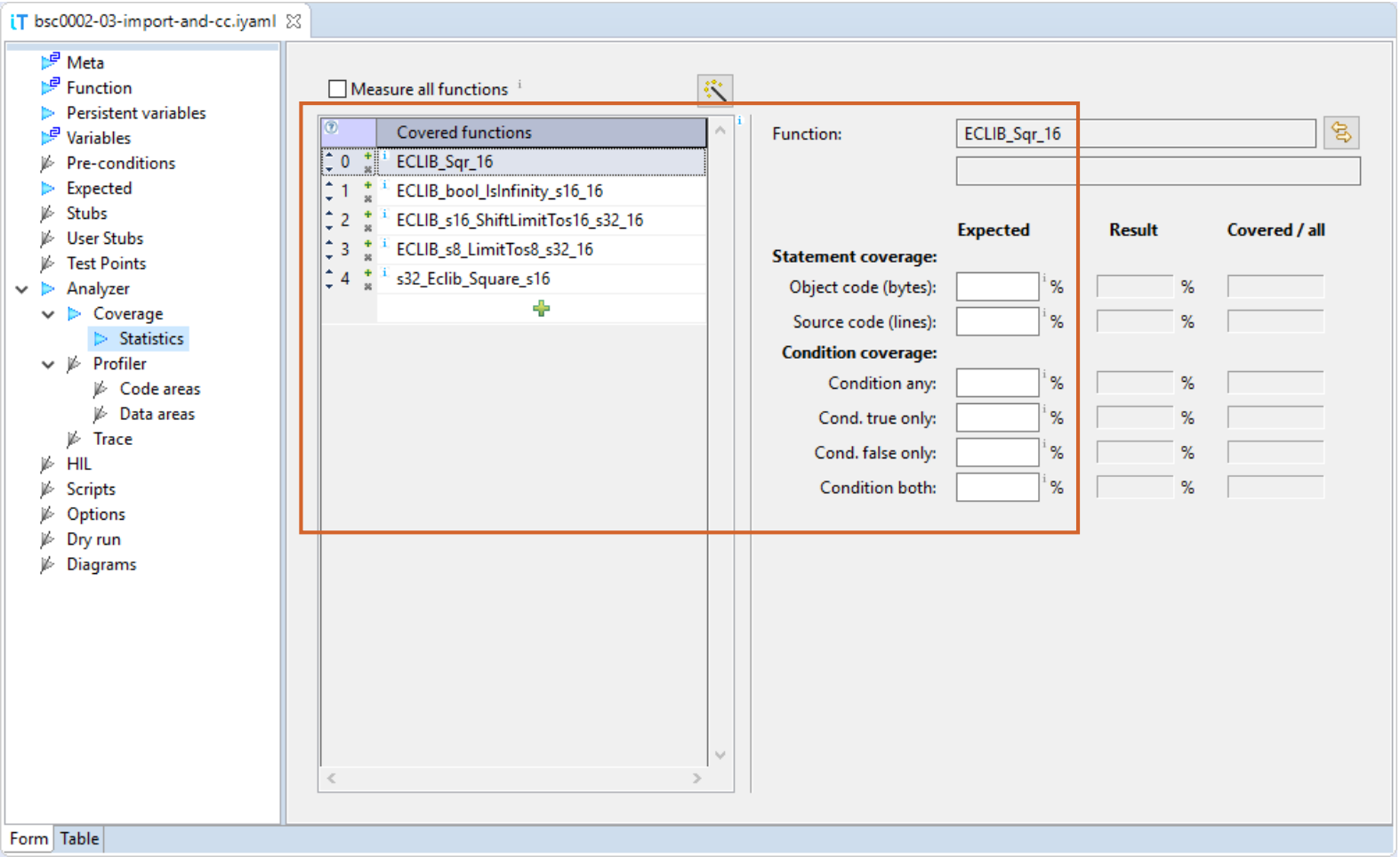
3 STATISTICS SETTINGS – FINAL DERIVED TEST

Finally, we can configure *Statistics* in the final derived test.

We start by listing all the functions we required code coverage results for. To start with, we have added *ECLIB_Sqr_16* since this is the function we are testing. However, this function relies upon many other functions. If so desired, we can add these into our statistics measurement.

Expected values for code coverage can also be entered for each function. If the expected code coverage is not achieved, the test (or suite of tests) are considered to have failed, even if the tests themselves passed.

Here we have left the expected values empty.

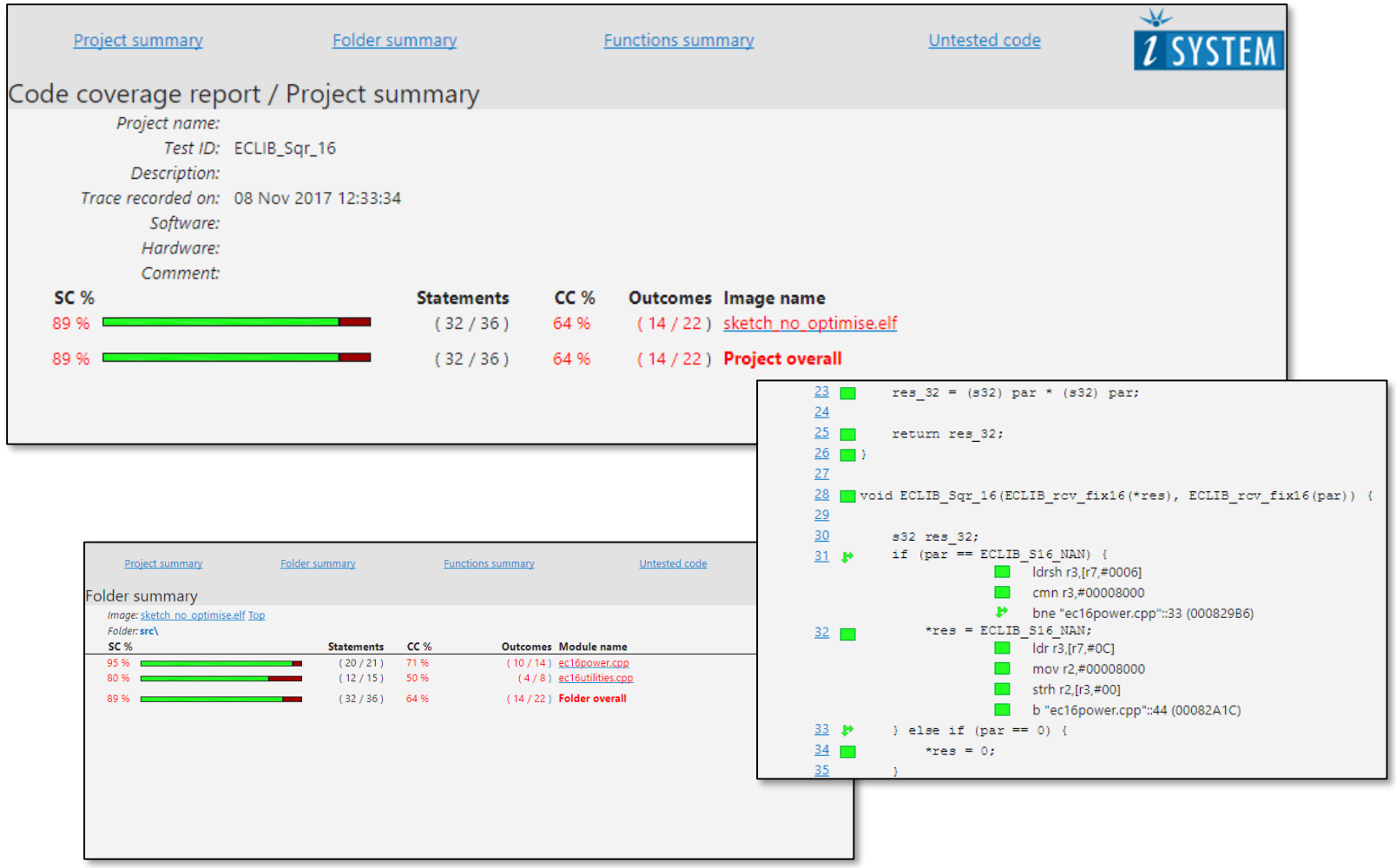


3 HTML REPORT OUTPUT

Once completed, the changes can be saved and the suite of tests executed.

Upon completion of the tests, as configured, the default browser is open with the code coverage results in HTML.

The various links in the file allow the user to drill down as far as the assembler code (if inclusion was selected in the *Statistics* settings).



5 TEST REPORT PROCESS

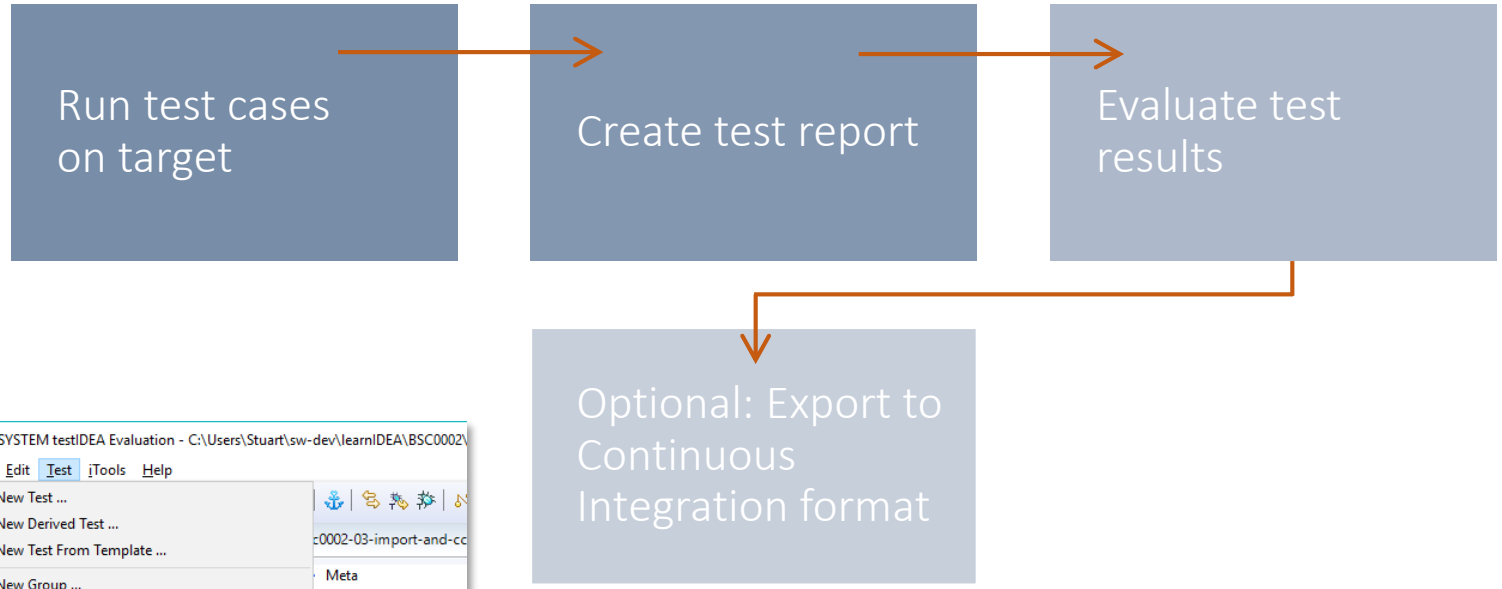
Once the tests have been completed and the code coverage has been generated, it is time to generate the report for the test's outcomes.

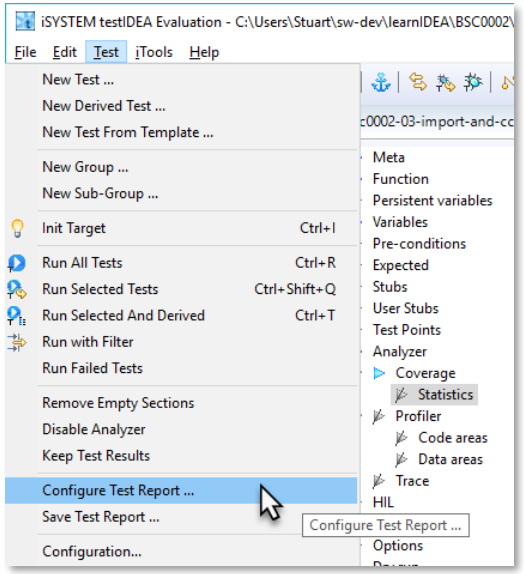
This can be undertaken in two steps:

1. Configuration of the report (*Configure Test Report...*)
2. Export of the report (*Save Test Report...*)

The configuration allows the user to define a template used by all reports, whilst directly exporting a report offers the chance to modify the template settings.

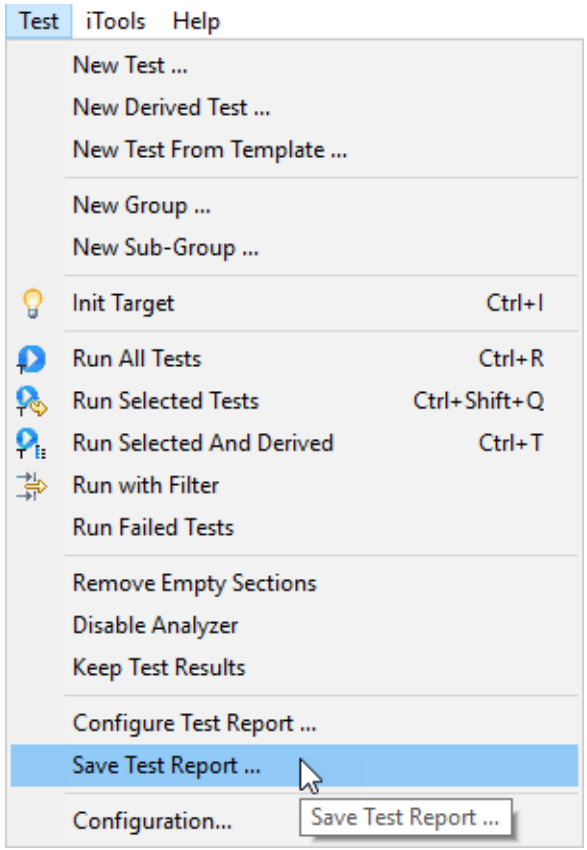
Both options are accessible from the menu under *Test*. In the following slides we will simply create a single report directly.





5 CREATE A TEST REPORT

To create a test report we open the *Test* menu and select the "*Save Test Report*..." option.

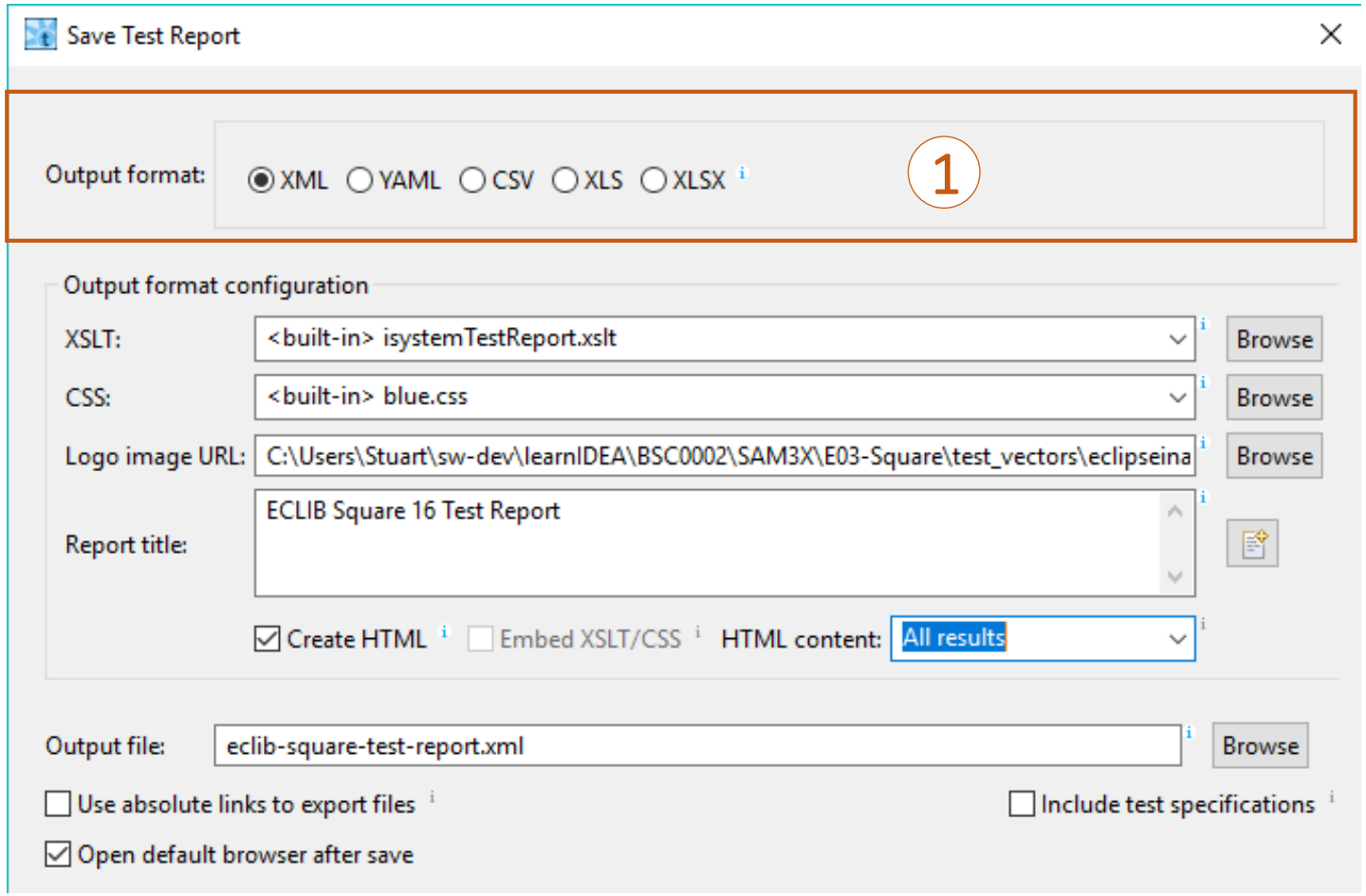


5 TEST REPORT - OUTPUT FORMATS

1 testIDEA can create test reports in several formats, including XML, YAML and Excel's XLSX file formats. Here we will select *XML* and later configure the report to also generate an HTML output.

XML format is convenient for usage in other tools, including viewing in web browsers, because it is widely supported.

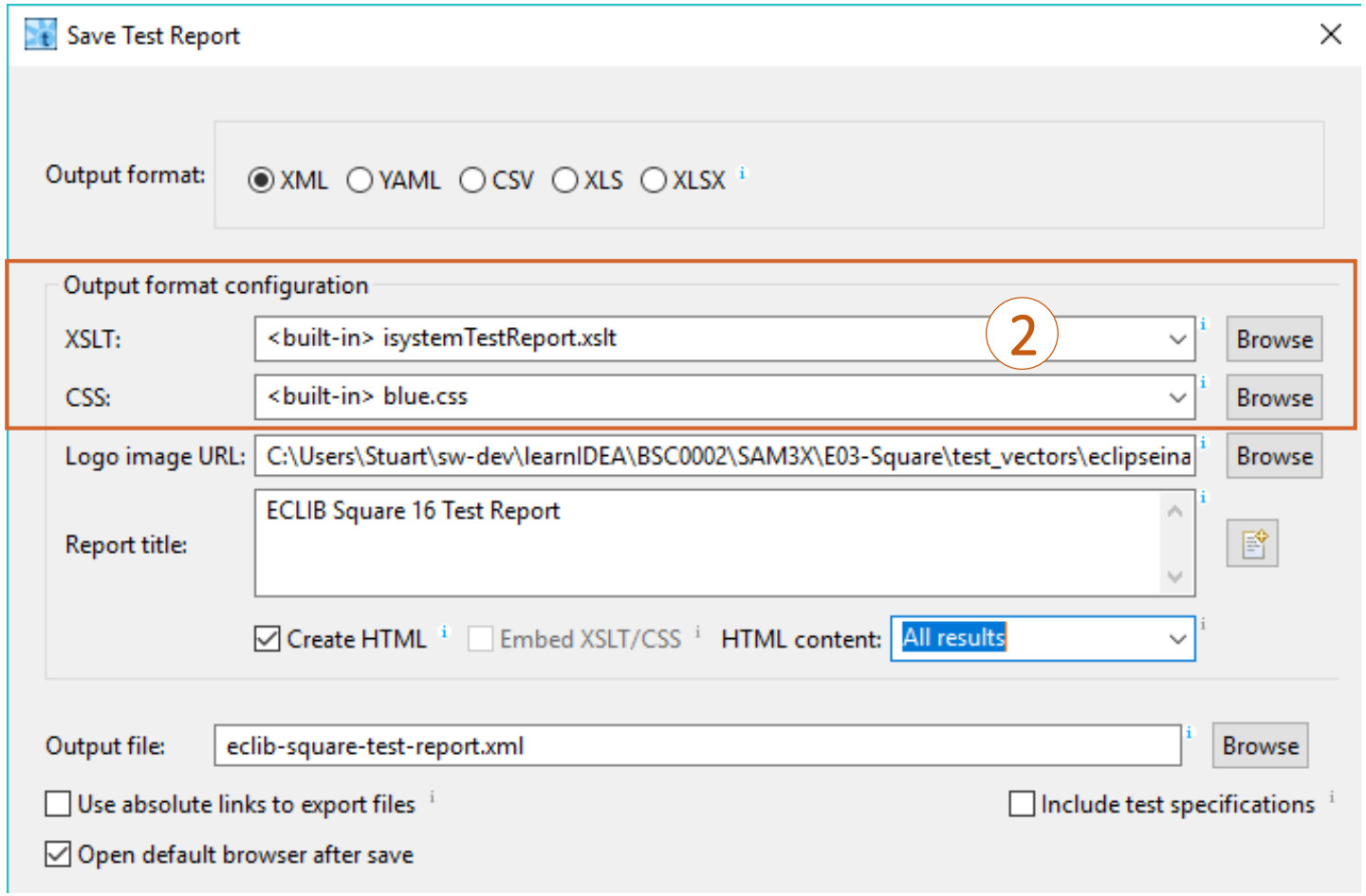
YAML format is common in testing tools. It is a superset of JSON and, due to its human-readable format, convenient for a quick overview in text editor or importing into other tools.



5 TEST REPORT - OUTPUT FORMAT CONFIGURATION

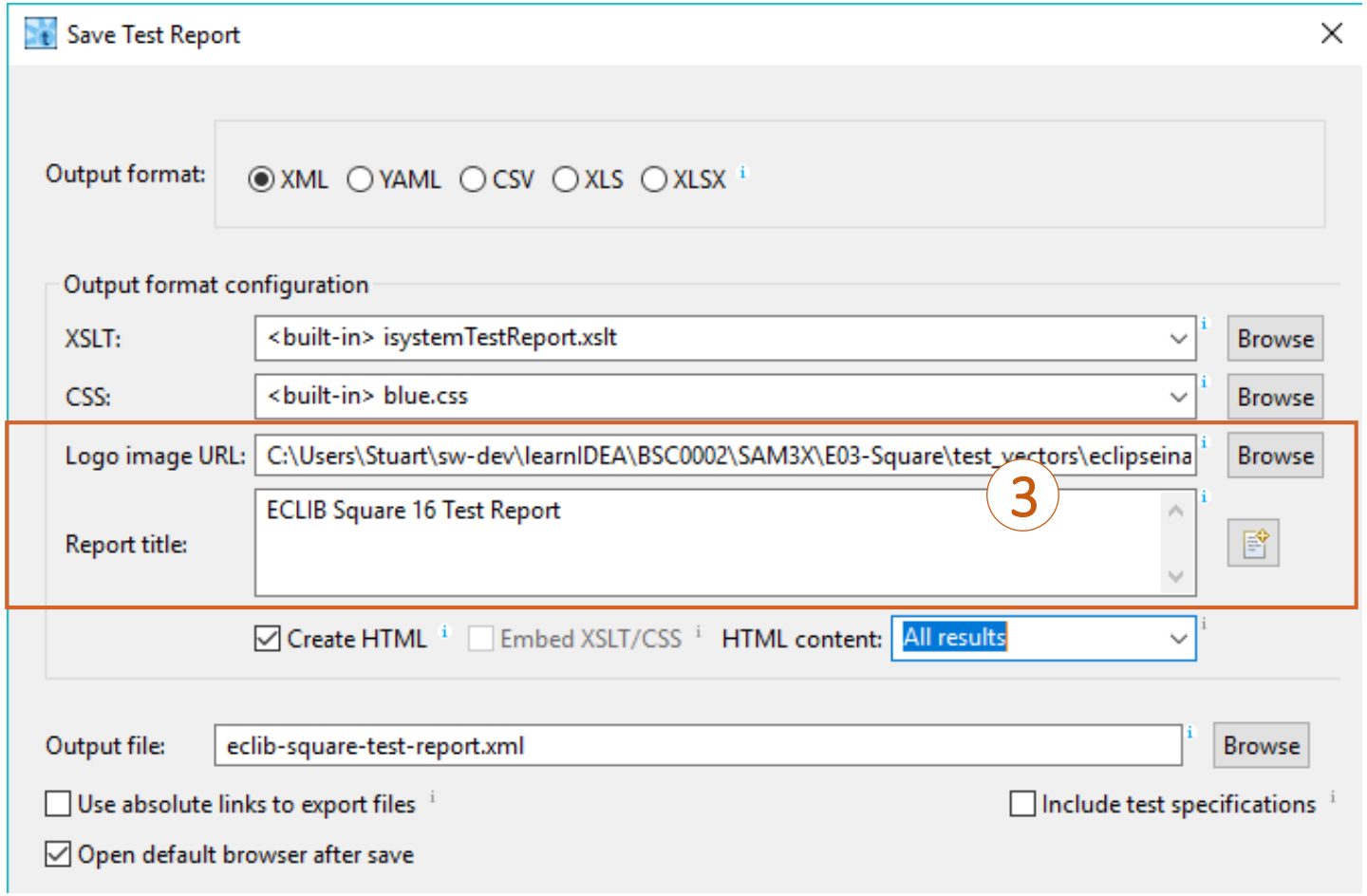
2 XSLTs are Extensible Stylesheet Transformations used to transform XML documents into other formats, such as HTML. CSS adds styling (font style, size, color) to HTML output. You can use the built-in XSLTs, or create your own.

Both the XSLT and CSS files can be modified to match corporate styling and color choices. Both file types come from the world of web design and, if you wish to modify them, simply analyze the existing content and refer to the wealth of online tutorials to learn how to configure them to meet your needs.



5 TEST REPORT - OUTPUT FORMAT CONFIGURATION

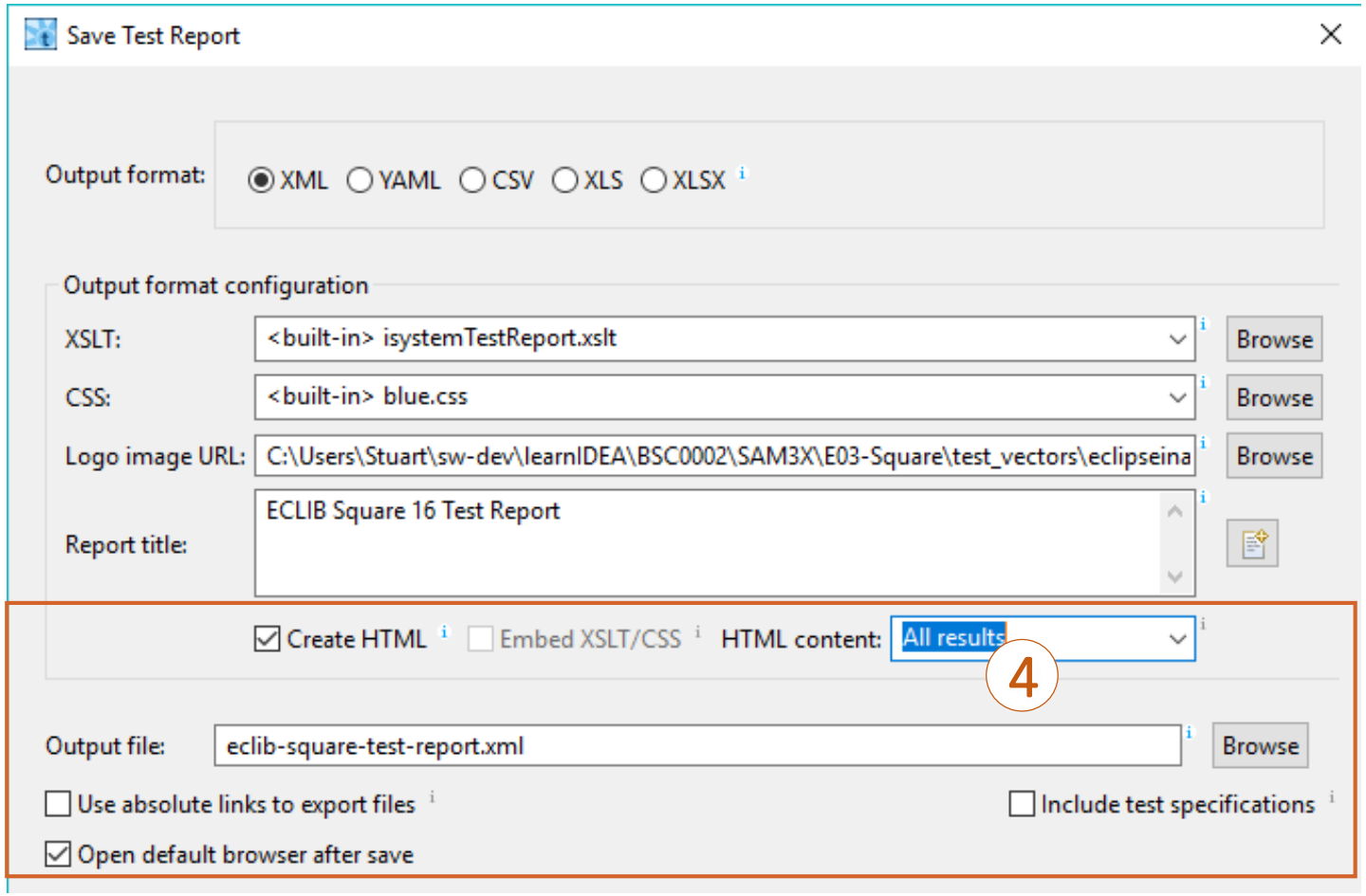
3 Add a link to your corporation's logo if desired. It is also helpful to add a description for the title of the report.



5 TEST REPORT - OUTPUT FORMAT CONFIGURATION

4 Here we will define that an HTML format file is desired for the report, setting the check box *Create HTML*.
The *Output file* name is given for the XML file. The HTML file will be given the same name.
We have also set the check box *Open default browser after save* to ensure the default browser displays the report once it is complete.

The *HTML content* setting can also be configured to show either all results or just the results of failed tests.



5 TEST REPORT - OUTPUT

With the chosen settings the **test report** will be displayed like this.

eclipseina	**ECLIB Square 16 Test Report**

Test Configuration	
report file	eclib-square-test-report.xml
testIDEA version	9.17.0
winIDEA version	9.17.0
wiWorkspacePath	C:\Users\Stuart\sw-dev\learnIDEA\BSC0002\SAM3X\E03-Square

Test Statistic	
Number of all tests	119
Number of not passed tests	0
Failure/Error type	**No. of failures/errors**
Errors (test execution exceptions)	0
Expression failures	0
Coverage failures	0
Code profiler failures	0
Data profiler failures	0
Script failures	0
Stub failures	0
Test point failures	0
Stack usage failures	0

5 TEST REPORT - OUTPUT

The **coverage report** is attached to the final test in the set of derived tests (in this case, test *ECLIB_Sqr_16.0119*).

The details for the code coverage statistics and methods of measurement can be found in the winIDEA help under the topic *Analyzer -> Coverage Concepts*.

Test ID	Function	Result
ECLIB_Sqr_16.0119	ECLIB_Sqr_16	Pass

Tags	Base tests
	/Short_-_ECLIB_Sqr_16

Assert expressions	
Expression	Sub-expressions
myResult == ECLIB_S16_NAN	myResult = 0x8000 (-32768) ECLIB_S16_NAN = 0x8000 (-32768)
myResult_sf == -17	myResult_sf = \xEF (0xEF) (-17) -17 = \xEF (0xEF) (-17)

Coverage	
Document	ECLIB_Sqr_16.trd
Export file	eclib_sqr_16.html\index.html

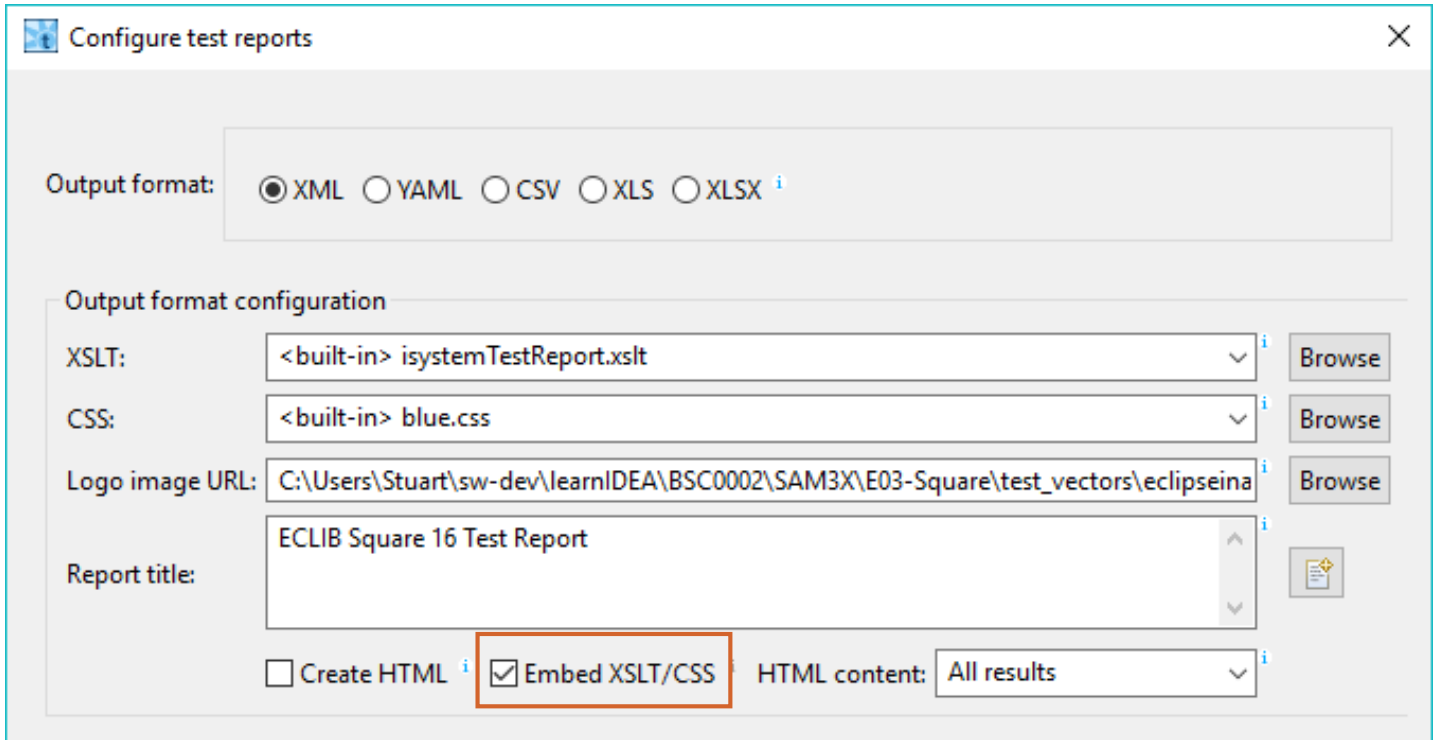
Function	Obj. code all	Src. lines all	Cond. all	CC (Outcomes)	Obj. code executed measured (exp., abs.)	Src. lines executed measured (exp., abs.)	Conditions any measured (exp., abs.)	Cond. true only measured (exp., abs.)	Cond. false only measured (exp., abs.)	Conditions both measured (exp., abs.)
ECLIB_Sqr_16	142	10	5	50.0% (5/10)	**64.8%** (0.0%, 92)	**80.0%** (0.0%, 8)	**60.0%** (0.0%, 3)	**0.0%** (0.0%, 0)	**20.0%** (0.0%, 1)	**40.0%** (0.0%, 2)
ECLIB_bool_IsInfinity_s16_16	58	7	2	50.0% (2/4)	**82.8%** (0.0%, 48)	**57.1%** (0.0%, 4)	**100.0%** (0.0%, 2)	**50.0%** (0.0%, 1)	**50.0%** (0.0%, 1)	**0.0%** (0.0%, 0)
ECLIB_s16_ShiftLimitTos16_s32_16	72	7	2	0.0% (0/4)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)
ECLIB_s8_LimitTos8_s32_16	54	8	2	0.0% (0/4)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)
s32_Eclib_Square_s16	38	4	0	/ (0/0)	**0.0%** (0.0%, 0)	**0.0%** (0.0%, 0)	/ (0.0%, 0)	/ (0.0%, 0)	/ (0.0%, 0)	/ (0.0%, 0)

5 TEST REPORT - XML FORMAT CONFIGURATION

When creating only XML format files with the intention of viewing them in a web browser, be aware that some browsers work differently to others.

If you cannot see the content of the file when it is opened in your default browser, you may wish to try embedding the XLST and CSS into the XML file using the check box option shown.

The most portable solution is the *Create HTML* option although it requires the most disk space for the files generated.



*Not all **browsers** may show standalone and embedded versions of reports correctly. While Internet Explorer has problems with single files (XSLT and CSS embedded), Chrome does not show report contents when XSLT is stored as a separate file. Firefox properly displays report content in both cases.*

SUMMARY

testIDEA

6 SUMMARY

- Code coverage can be easily generated for unit tests within testIDEA
- The *Slow Run* feature enables the generation of program trace data, even on microcontrollers without hardware trace support
- Upon completion of testing, reports can be easily generated for documentation purposes