



Unit test

Practical guide

Date: July 13, 2021

Project Name: Unit test

Document Version: V 1.0

Document Status: Working document

The information contained herein is believed to be accurate as of the date of publication, however, B&R makes no warranty, expressed or implied, with regards to the products or the documentation contained within this document. The software names, hardware names and trademarks used in this document are registered by the respective companies.

I Versions

Version	Date	Comment	Edited by
1.0	4/6/2021	First Edition	Pion Francis

Table 1: Versions

II Table of Contents

1 Intended Use	4
2 Introduction.....	5
3 Unit test in Automation Studio	6
3.1 Download the technology solution	6
3.2 Import the technology solution	7
3.2.1 Technology Solution – Logical View.....	9
3.2.2 Technology Solution – Configuration View.....	10
3.3 Adjusting deployment.....	12
4 First step	14
5 Adding a test.....	18
6 Tips and tricks	25
6.1 Asynchronous function block	25
6.2 Cycle counter	25
6.3 Test location.....	27
6.4 Integration test	27

1 Intended Use

This document is complementary to the Unit test section of Automation Studio's help. It will provide many step by step examples on how to implement develop unit test efficiently.

2 Introduction

Since software is becoming a bigger and bigger part of the overall engineering effort of a project, challenges are arising. Nowadays, in a typical project, there are many developers that are contributing and merging their code together. As part of that process, all functionality must be preserved as new features are added.

The primary reason to use automated test is to minimize resources required for project completion. During a project, bugs and defects that are found later in development cost more resources and are harder to fix. Since unit test focuses on individual system functionality, it can begin early in development and it is not necessary to wait for machine hardware. Automated testing also allows the tests to run faster and consistently. This leads to more thorough testing that can catch bugs quicker.

3 Unit test in Automation Studio

Automation Studio has a technology solution that can be downloaded for free. It's suitable for unit test and depending on your software architecture, integration test as well.

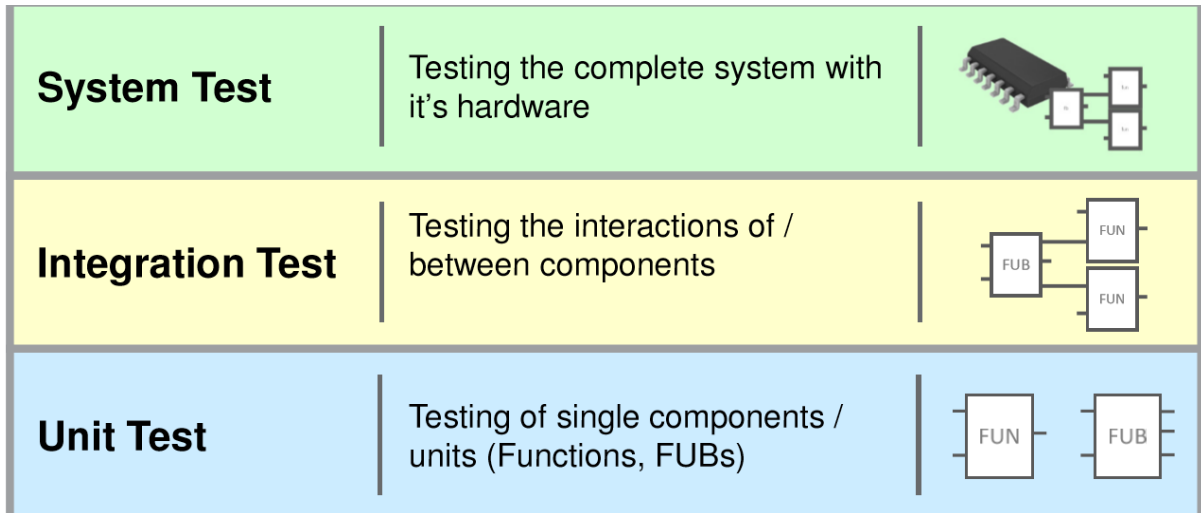


Figure 1 - Different levels of testing

It's easy to get lost in all the small details at first, but ignore them for now and focus solely on the steps described in each section.

3.1 Download the technology solution

From Automation Studio, execute those steps in order.

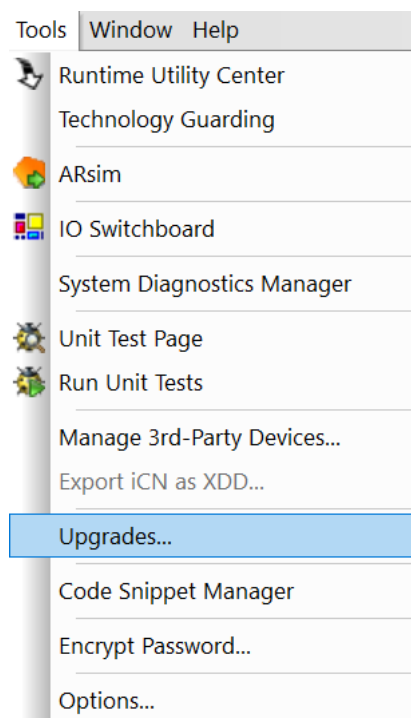


Figure 2 - Tools upgrade

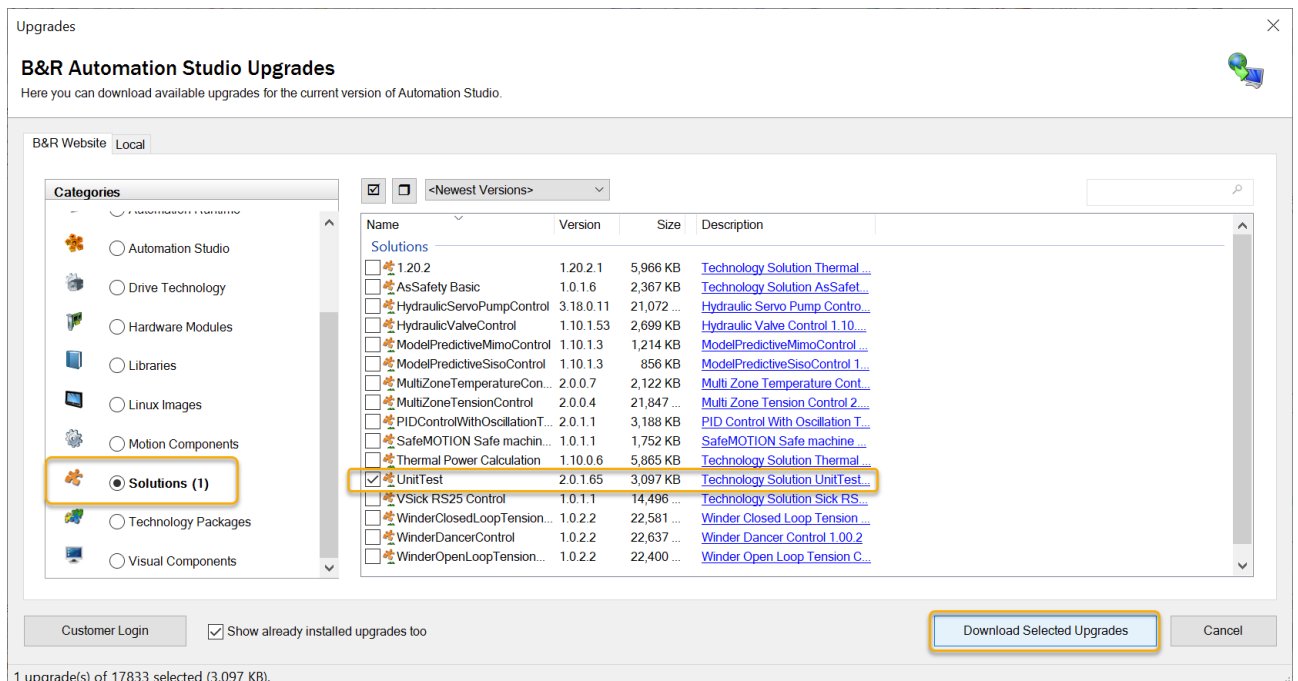


Figure 3 - Select UnitTest

3.2 Import the technology solution

Once the technology solution is installed, it can be imported in a project.

Click on the root of your project.

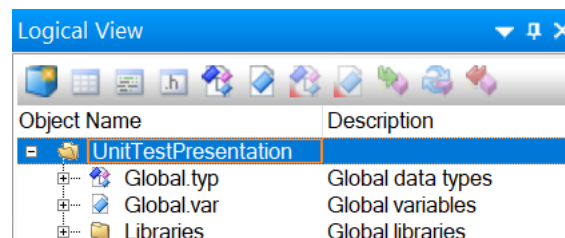


Figure 4 - Select project root

From the toolbox, select Solutions and double-click the Technology Solutions:

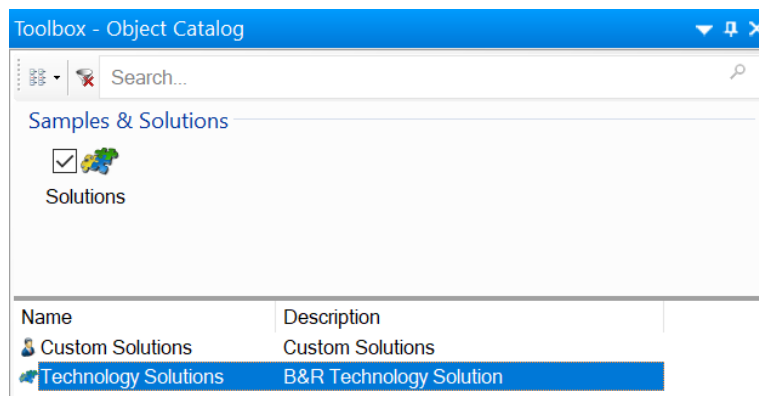


Figure 5 - Technology Solutions toolbox

Select Unit Test technology solution and import it in your project.

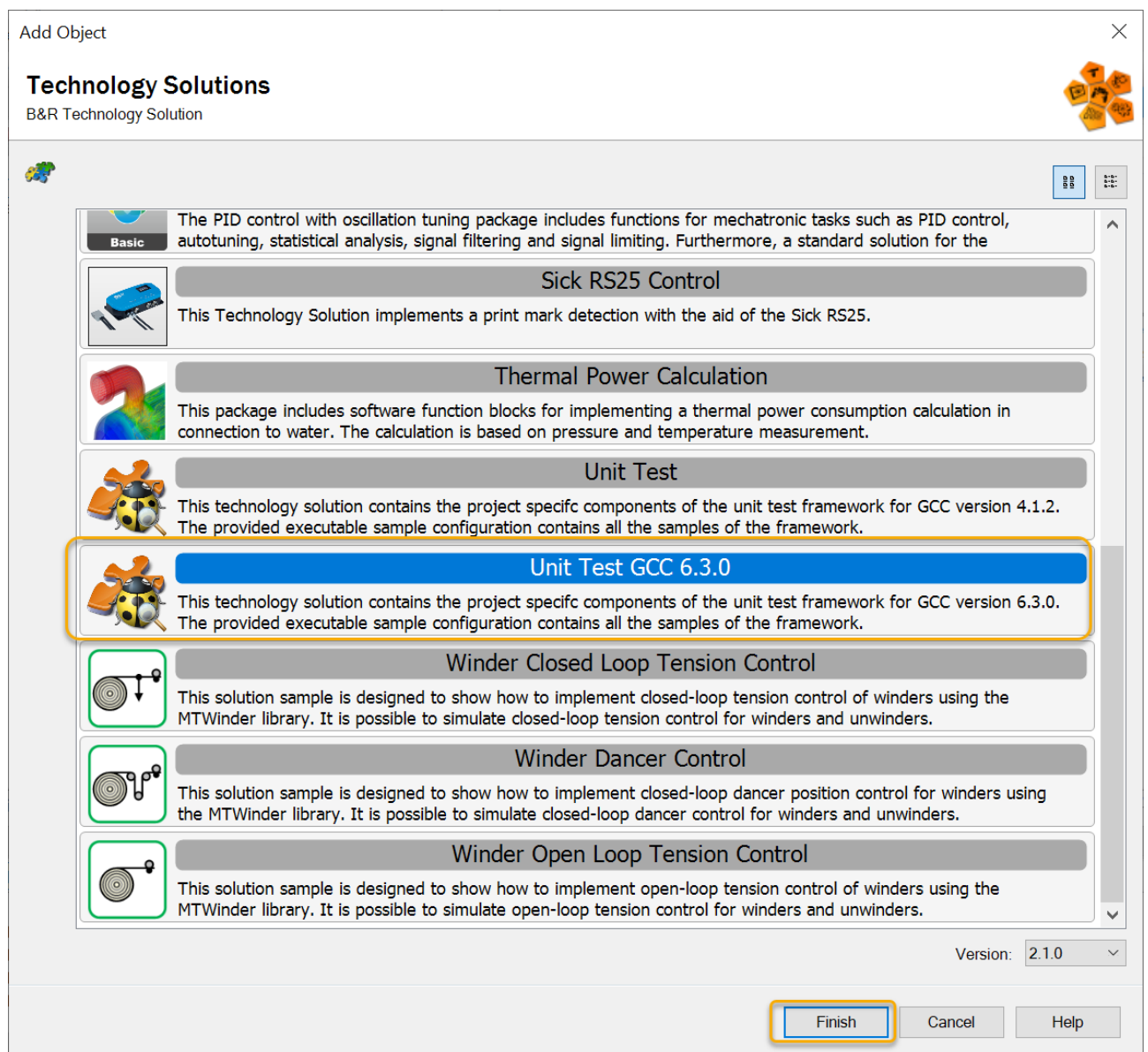


Figure 6 - Technology Solutions window

This will add two new buttons to Automation Studio and a new interface.

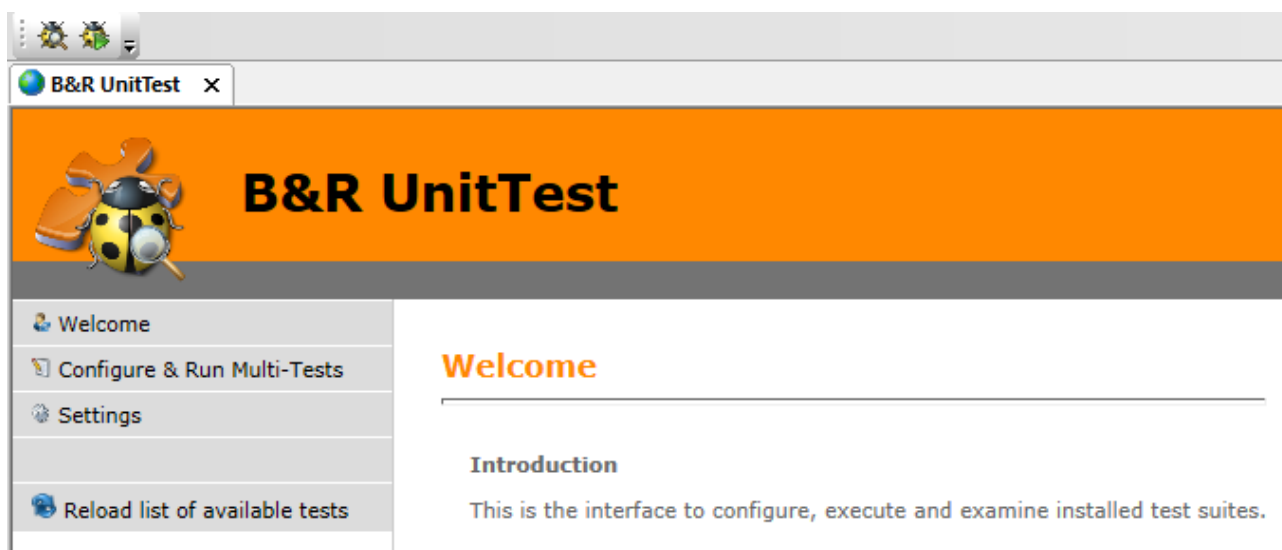


Figure 7 - Unit Test interface

It will also add an entry in the Help.

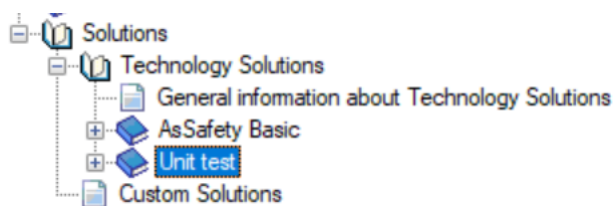


Figure 8 - Help file entry

3.2.1 Technology Solution – Logical View

The import inserted a folder in the logical view.

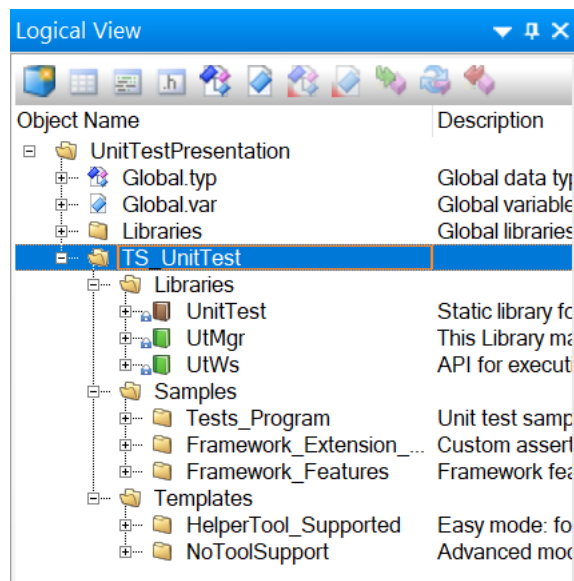


Figure 9 - TS folder

The 3 libraries need to be deployed in the configuration view for the solution to work. Later in this guide we'll explore Samples/Tests_Program and Templates/HelperTool_Supported. Everything else is very niche and will not be covered. Refer to the help documentation for those.

3.2.2 Technology Solution – Configuration View

The import added a new configuration to the project.

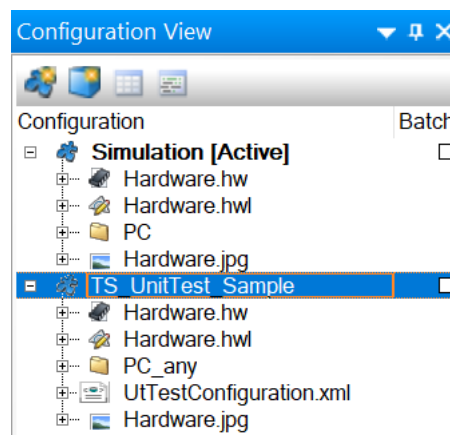


Figure 10 - TS configuration

It's not a requirement, but it is more convenient to keep this configuration than using another one. A dedicated unit test configuration is recommended since the unit tests tasks shouldn't run alongside the actual tasks for the project. If this configuration is not used, then the settings in Figure 11 must be applied to your new configuration in order to prevent undefined reference.

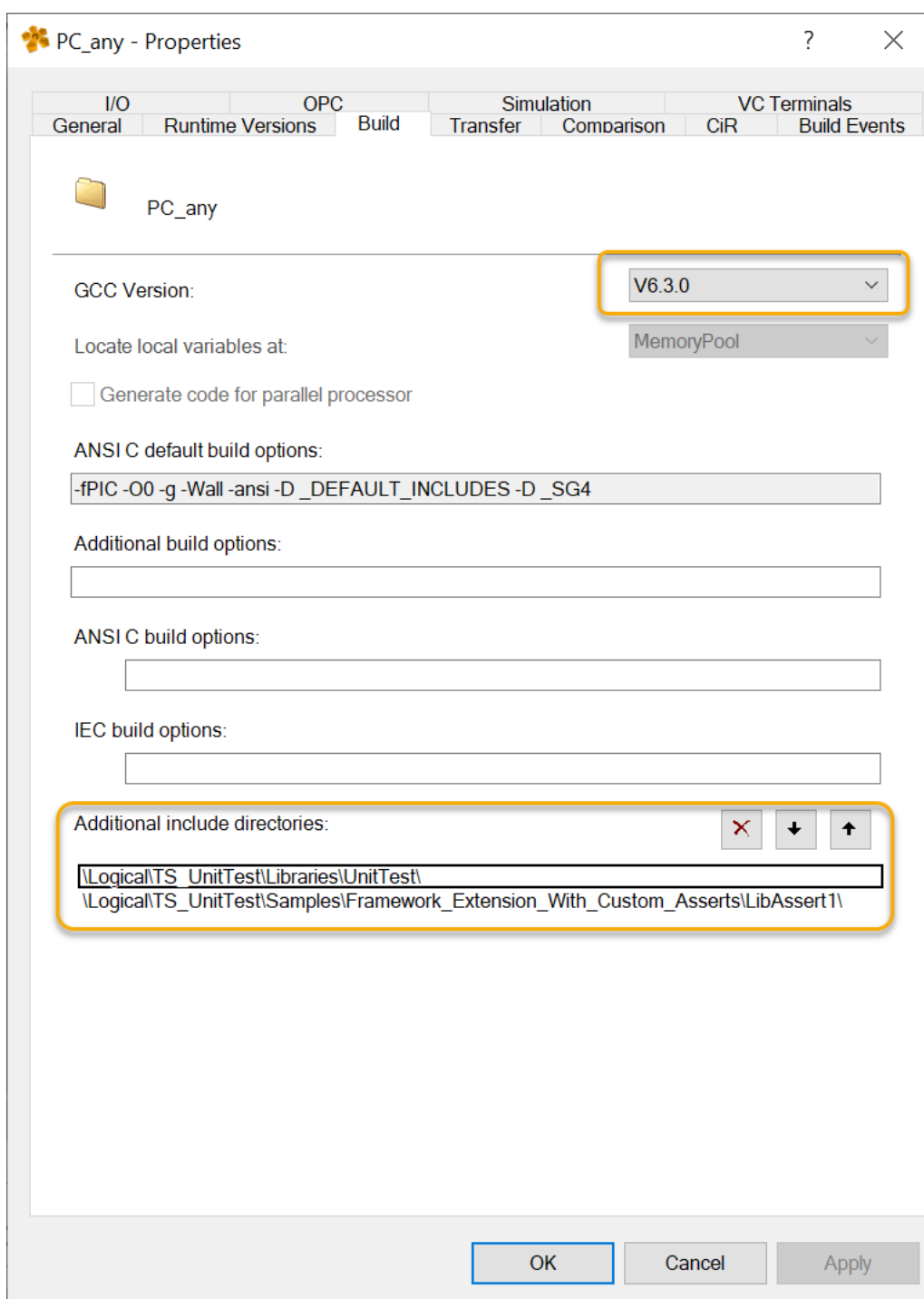
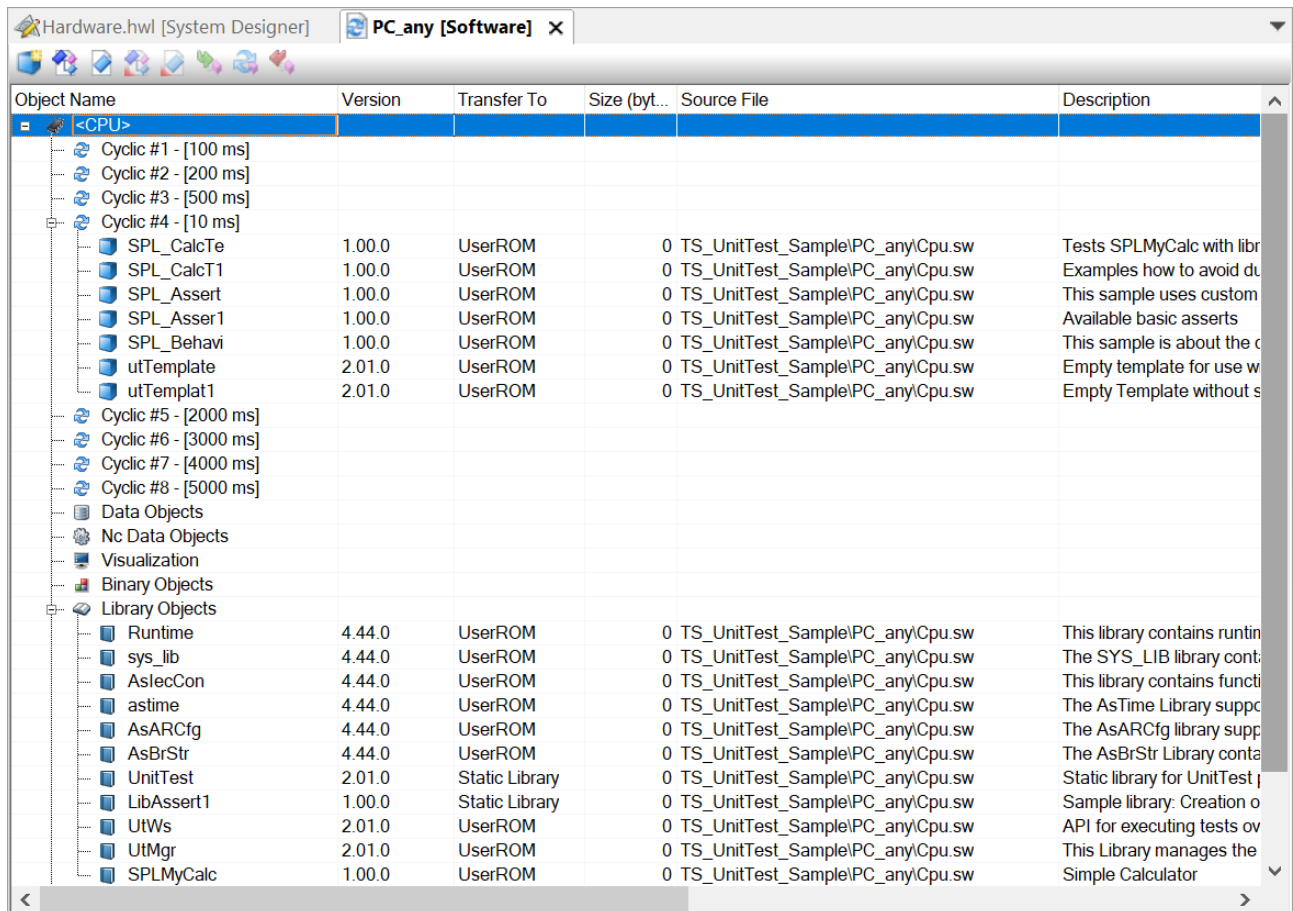


Figure 11 - .h Reference for unit test libraries

The default configuration has a Standard PC configuration as hardware with the task and libraries from the Logical View already deployed.

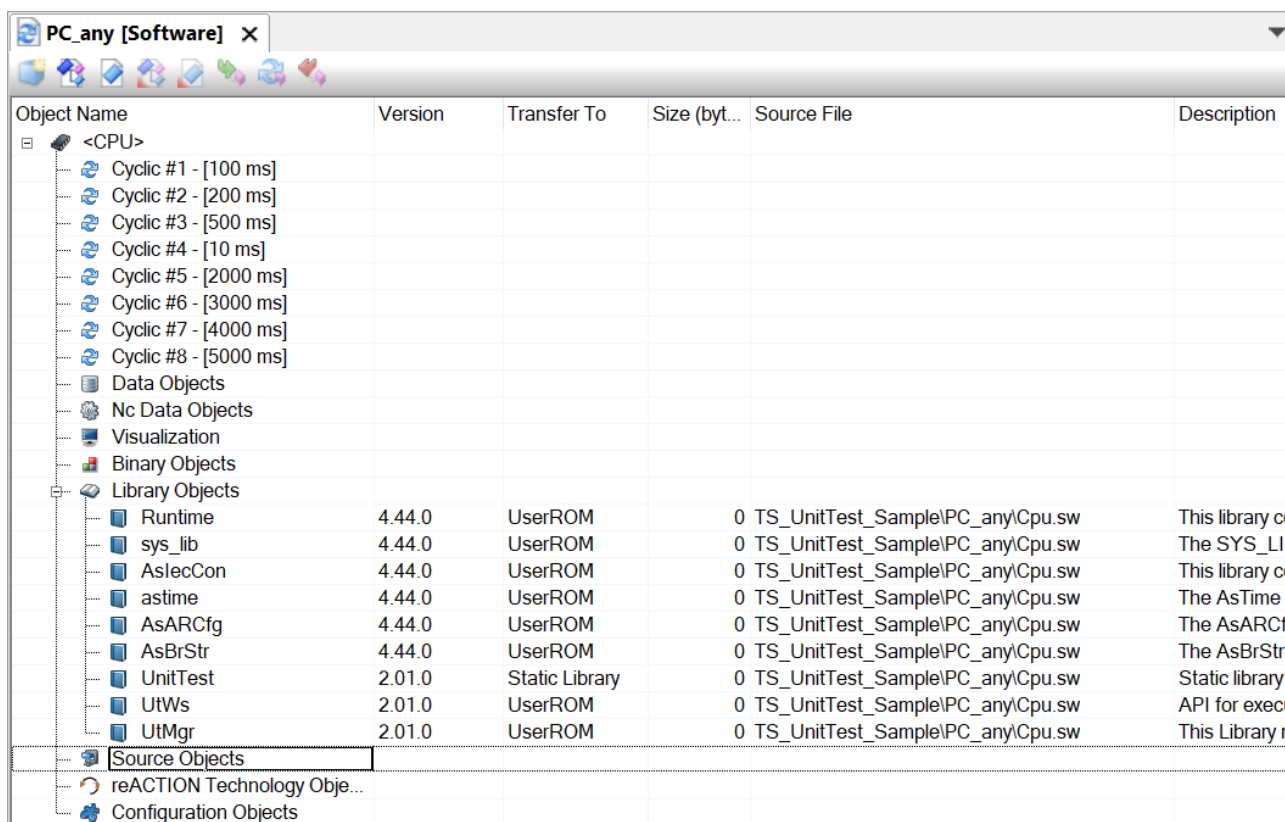


Object Name	Version	Transfer To	Size (byt...	Source File	Description
<CPU>					
Cyclic #1 - [100 ms]					
Cyclic #2 - [200 ms]					
Cyclic #3 - [500 ms]					
Cyclic #4 - [10 ms]					
SPL_CalcTe	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Tests SPLMyCalc with libr
SPL_CalcT1	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Examples how to avoid du
SPL_Assert	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This sample uses custom
SPL_Asser1	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Available basic asserts
SPL_Behavi	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This sample is about the c
utTemplate	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Empty template for use w
utTemplat1	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Empty Template without s
Cyclic #5 - [2000 ms]					
Cyclic #6 - [3000 ms]					
Cyclic #7 - [4000 ms]					
Cyclic #8 - [5000 ms]					
Data Objects					
Nc Data Objects					
Visualization					
Binary Objects					
Library Objects					
Runtime	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This library contains runtin
sys_lib	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The SYS_LIB library cont
AslecCon	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This library contains functi
astime	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The AsTime Library supp
AsARCfg	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The AsARCfg library supp
AsBrStr	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The AsBrStr Library conta
UnitTest	2.01.0	Static Library	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Static library for UnitTest
LibAssert1	1.00.0	Static Library	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Sample library: Creation o
UtWs	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	API for executing tests ov
UtMgr	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This Library manages the
SPLMyCalc	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Simple Calculator

Figure 12 - Original software deployment from Technology Solution

3.3 Adjusting deployment

All the tasks seen in Figure 12 are from the Sample folder. They can all be deleted if you're not playing around with them. The libraries from that folder can also be deleted.



Object Name	Version	Transfer To	Size (byt...	Source File	Description
<CPU>					
Cyclic #1 - [100 ms]					
Cyclic #2 - [200 ms]					
Cyclic #3 - [500 ms]					
Cyclic #4 - [10 ms]					
Cyclic #5 - [2000 ms]					
Cyclic #6 - [3000 ms]					
Cyclic #7 - [4000 ms]					
Cyclic #8 - [5000 ms]					
Data Objects					
Nc Data Objects					
Visualization					
Binary Objects					
Library Objects					
Runtime	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This library co
sys_lib	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The SYS_LIF
AslecCon	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This library co
astime	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The AsTime I
AsARCFg	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The AsARCFg
AsBrStr	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	The AsBrStr
UnitTest	2.01.0	Static Library	0	TS_UnitTest_Sample\PC_any\Cpu.sw	Static library
UtWs	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	API for execu
UtMgr	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw	This Library n
Source Objects					
reACTION Technology Obje...					
Configuration Objects					

Figure 13 - Adjusted software deployment

The preconfigured tests can be deleted as well since they are only used for the Samples.

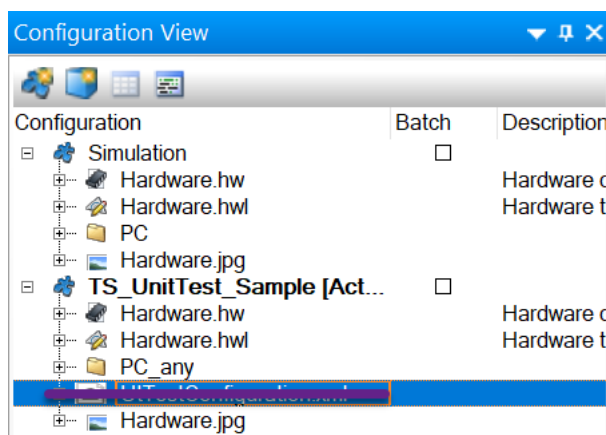
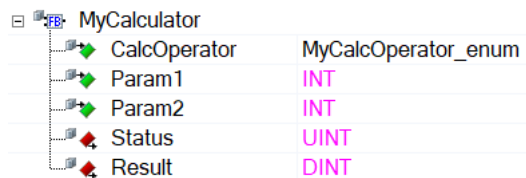


Figure 14 - Remove UtTestConfiguration

4 First step

Use the sample to experiment. It has a custom library partially implemented with some tests already defined.

1. Open the SPLMyCalc.fun file from the Sample folder



Observe the interface of the function block.

2. Go to the implementation.

```
FUNCTION_BLOCK MyCalculator

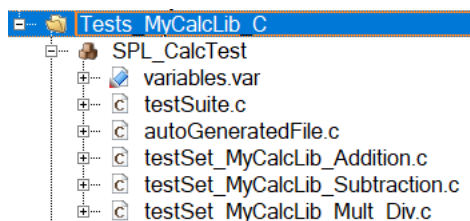
    Status := ERR_NOTIMPLEMENTED ;

END FUNCTION_BLOCK
```

Notice how the function block is almost empty. This is done on purpose. The sample is designed to help you understand the test-driven development methodology.

3. Open the Tests_MyCalcLib_C folder

The tests are part of a C task. This task holds one or multiple test sets which contains one or multiple test cases.



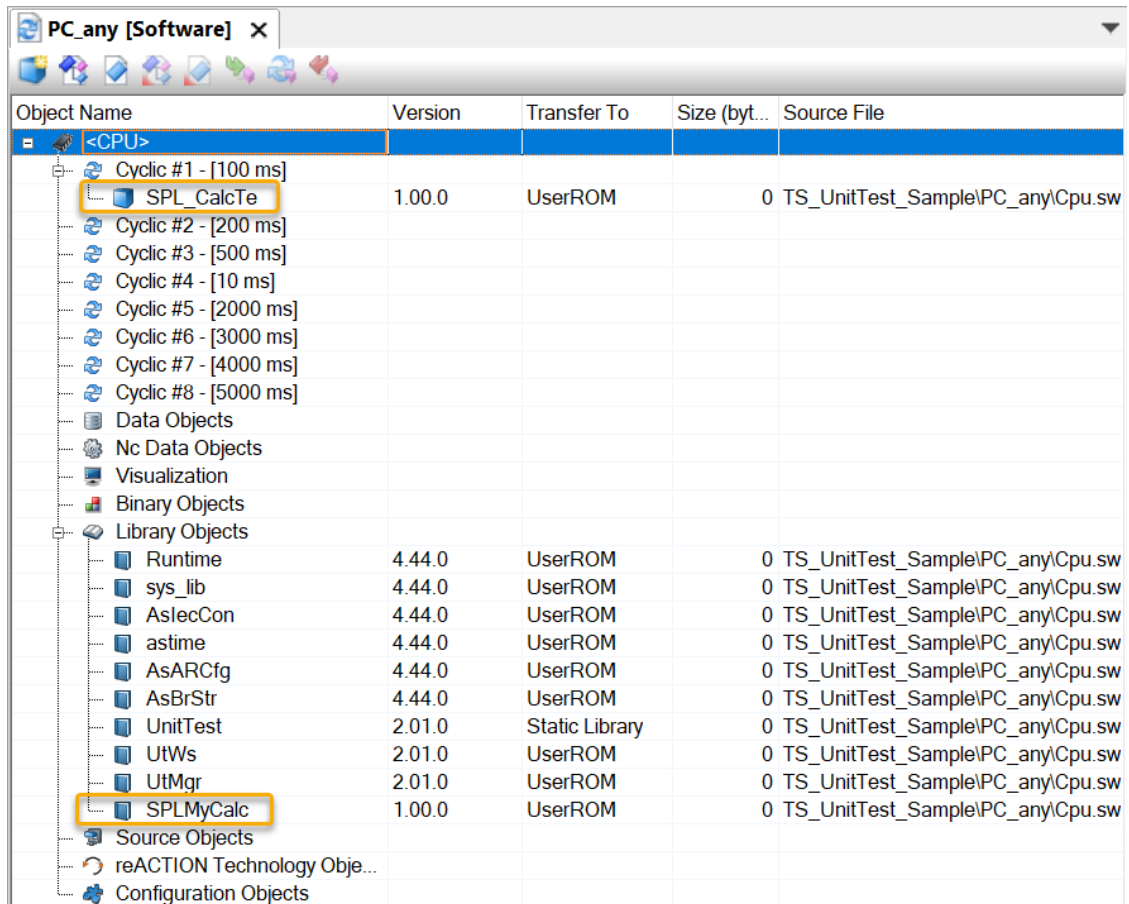
4. Open the testSet_MyCalcLib_Addition.c

```
19 _TEST test_Add(void)
20 {
21     /* initialize functionblock parameters */
22     instMyCalc.CalcOperator = splMyCalcOPERATOR_ADD;
23     instMyCalc.Param1      = 1;
24     instMyCalc.Param2      = 2;
25     /* call synchron functionblock */
26     MyCalculator(&instMyCalc);
27
28     /* check result */
29     TEST_ASSERT_EQUAL_INT(3, instMyCalc.Result);
30     TEST_ASSERT_EQUAL_INT(ERR_OK, instMyCalc.Status);
31
32     /* test is done */
33     TEST_DONE;
34 }
```

This specific test case is configuring the function block from line 22 to 24 then assessing that the result of the addition of 1 and 2 is 3 at line 29. A list of assert can be found in the Help.

5. Transfer the test

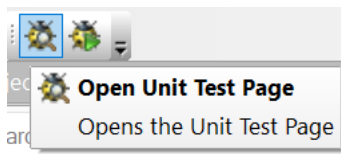
You'll need to have the following library and task deployed.



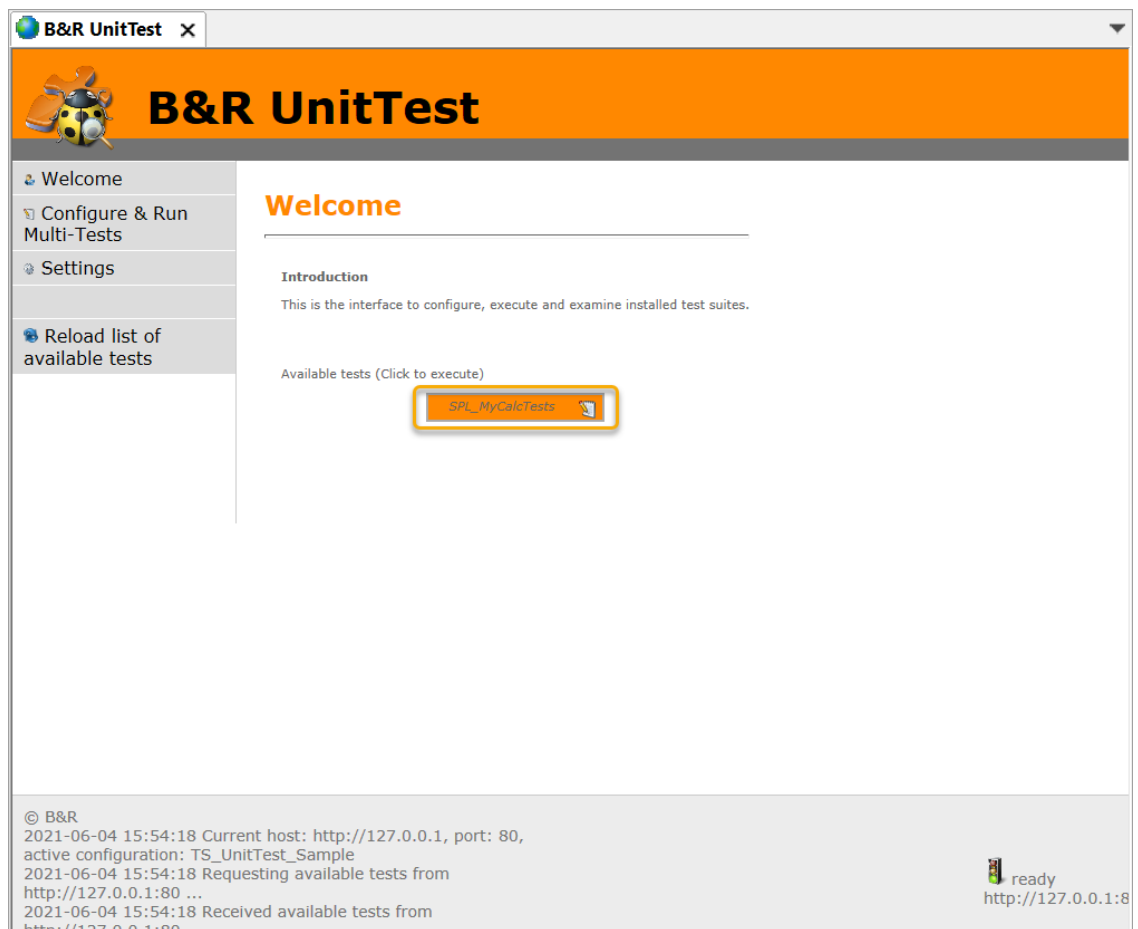
Object Name	Version	Transfer To	Size (byt...	Source File
<CPU>				
Cyclic #1 - [100 ms]				
SPL_CalcTe	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
Cyclic #2 - [200 ms]				
Cyclic #3 - [500 ms]				
Cyclic #4 - [10 ms]				
Cyclic #5 - [2000 ms]				
Cyclic #6 - [3000 ms]				
Cyclic #7 - [4000 ms]				
Cyclic #8 - [5000 ms]				
Data Objects				
Nc Data Objects				
Visualization				
Binary Objects				
Library Objects				
Runtime	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
sys_lib	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
AslecCon	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
astime	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
AsARCFg	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
AsBrStr	4.44.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
UnitTest	2.01.0	Static Library	0	TS_UnitTest_Sample\PC_any\Cpu.sw
UTWs	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
UtMgr	2.01.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
SPLMyCalc	1.00.0	UserROM	0	TS_UnitTest_Sample\PC_any\Cpu.sw
Source Objects				
reACTION Technology Obje...				
Configuration Objects				

6. Open the unit test page

Use the new button that appeared during the import.

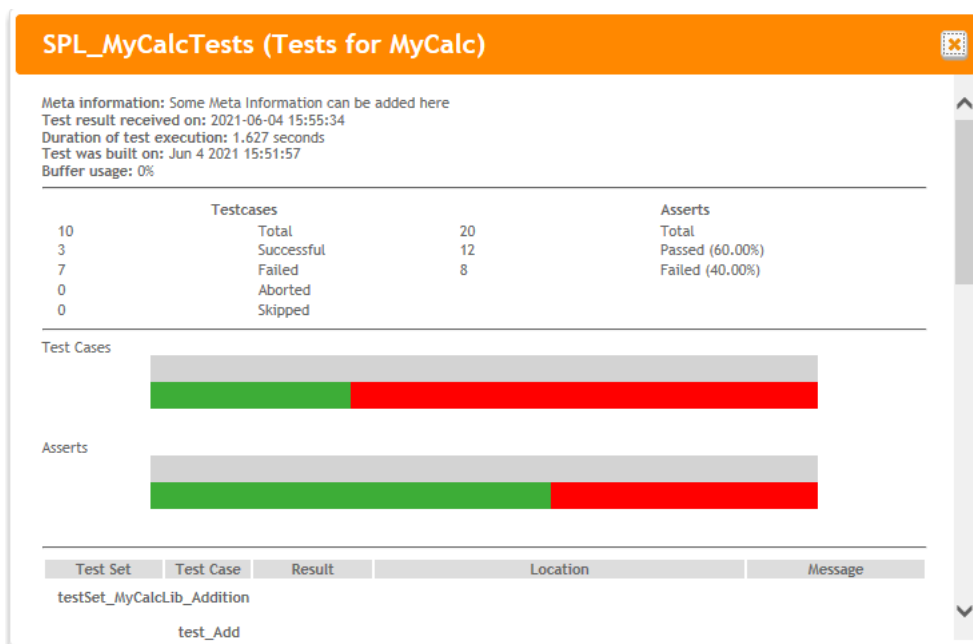


7. Run the test



8. Scroll through the result window

Notice that there are many tests results. That's because the SPL_CalcTest contains 3 tests sets which contains multiple test cases.



9. Implement the bare minimum to fulfil the test_Add test case

```
FUNCTION_BLOCK MyCalculator  
  
    Status := ERR_NOTIMPLEMENTED ;  
  
    CASE CalcOperator OF  
  
        splMyCalcOPERATOR_ADD:  
  
            Result := Param1 + Param2 ;  
  
            Status := ERR_OK ;  
  
        END_CASE;  
  
    END FUNCTION_BLOCK
```

10. Transfer and run the test again

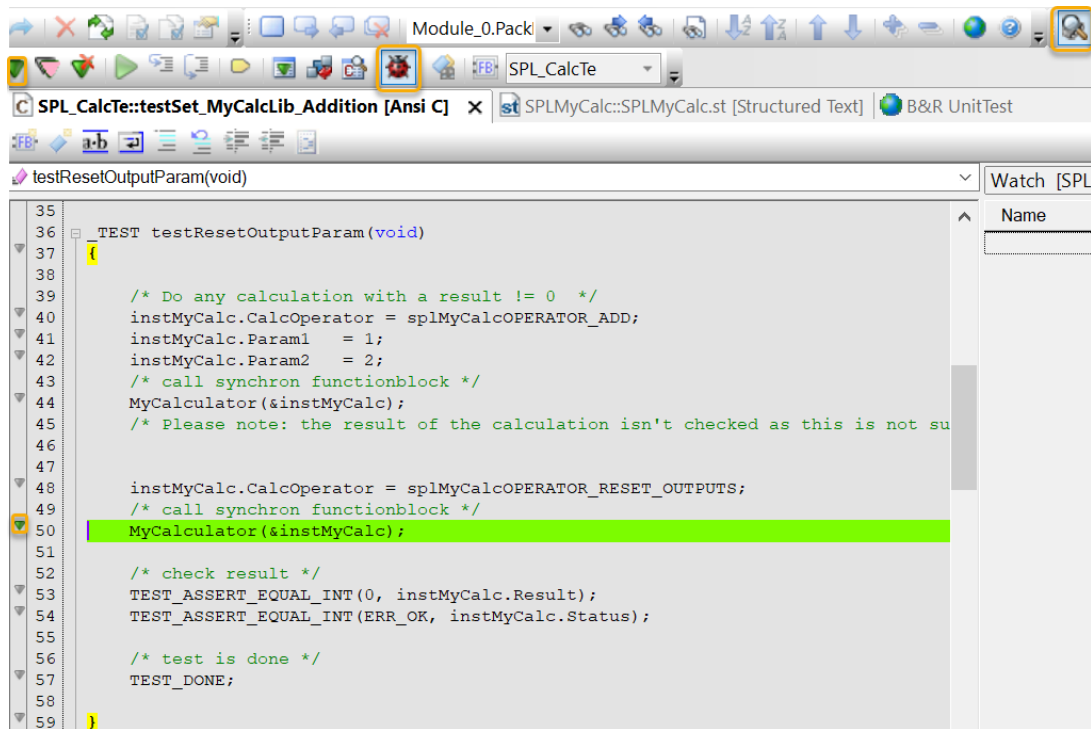
The test_Add test case is now successful. However, the testResetOutputParam is still incomplete. In order to troubleshoot quickly, you can click on a failed test. It will bring you to the assert that failed immediately.

Test Set	Test Case	Result	Location	Message
testSet_MyCalcLib_Addition	test_Add			
		Failed: 0 Passed: 2		
testResetOutputParam				
	failed	line: 53 file: testSet_MyCalcLib_Addition.c	exp 0 was 3	
	failed	line: 54 file: testSet_MyCalcLib_Addition.c	exp 0 was 9999	
		Failed: 2 Passed: 0		

11. Troubleshoot the issue

A fast way to troubleshoot unit test is to use the debugger. Put a break point at the last call before

a failed assert. Run the test again and enter the function block to figure out what's incorrect.

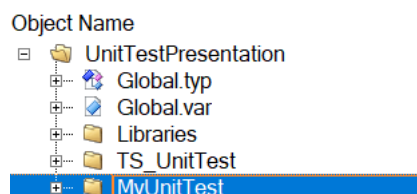


Note that the watch window can also be a useful debugging tool.

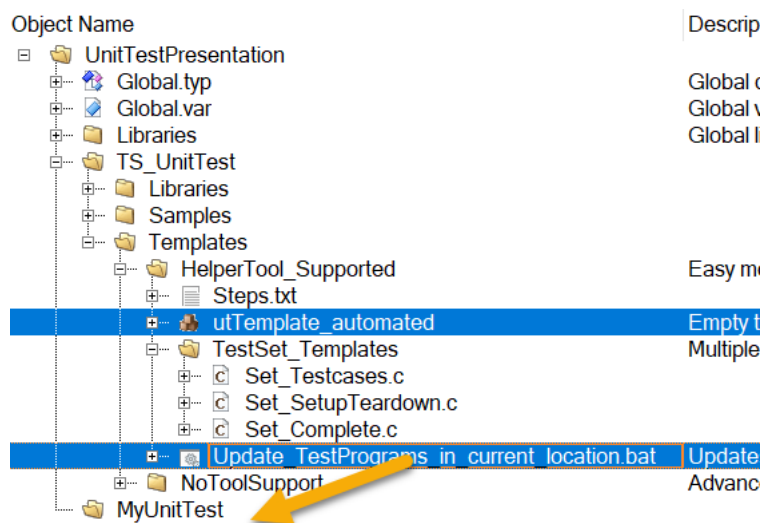
5 Adding a test

In section 4 we saw how we could run and troubleshoot unit test. Let's run through an example on how to create a new one using the Templates folder.

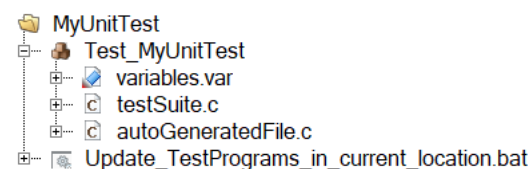
1. Create a unit test folder where you want to put your test



2. Copy utTemplate_automated and Update_TestPrograms_in_current_location.bat from the Help-erTool_Supported folder

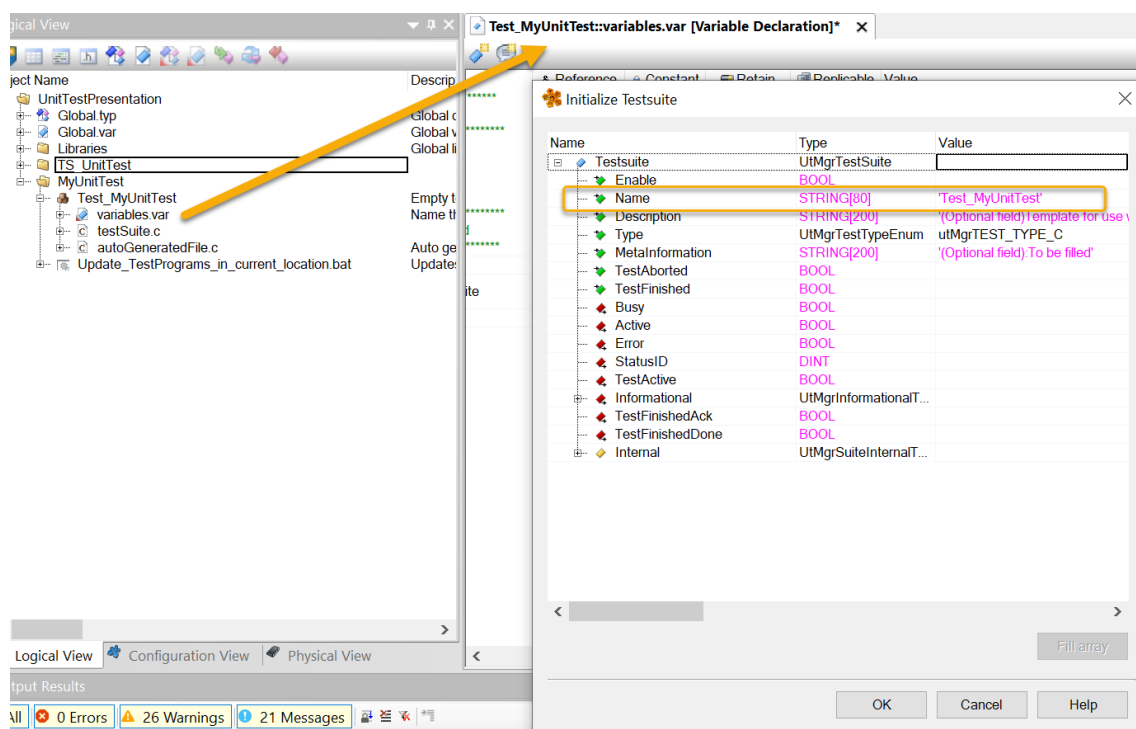


3. Rename your test task



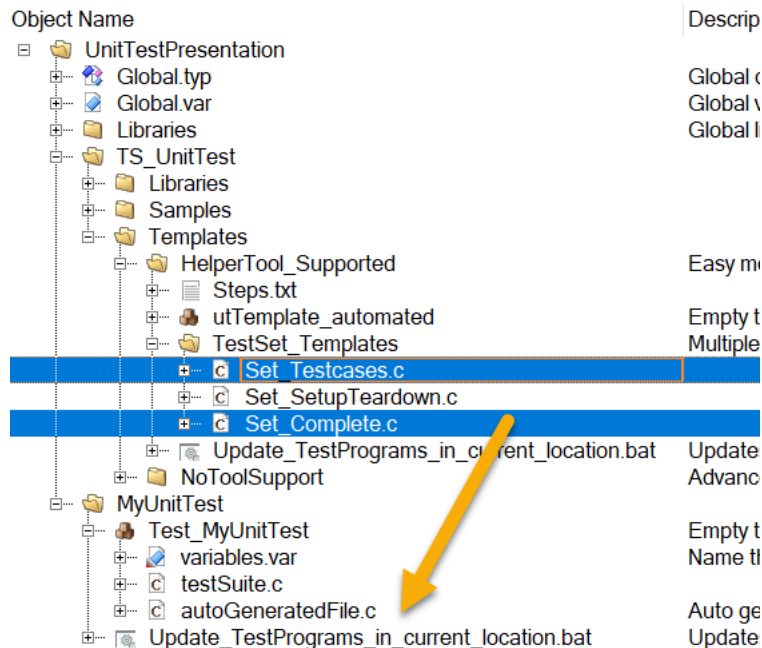
4. Rename the name member of the TestSuite local variable

This is the name that will be found in the Unit Test view.

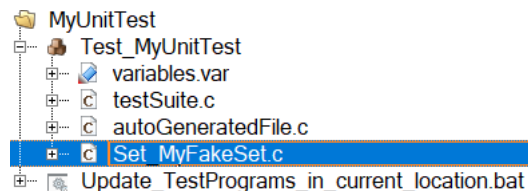


5. Copy the template test set that's suited for your test

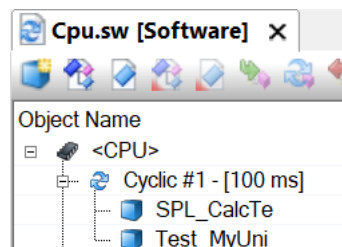
It's usually more convenient to use `Set_Complete` for function blocks and `Set_TestCases` for functions. The `Set_Complete` offers additional functionality such as code that can be executed before and after each test cases and test set. This is quite useful to configure and bring a function block to a known state before testing a given functionality.



6. Rename your set

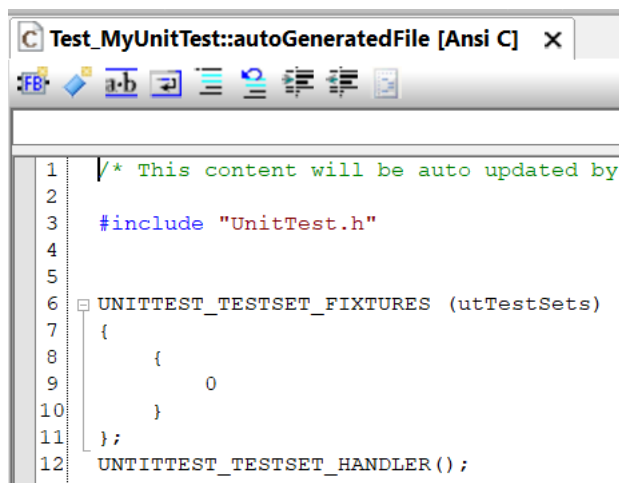


7. Deploy your task



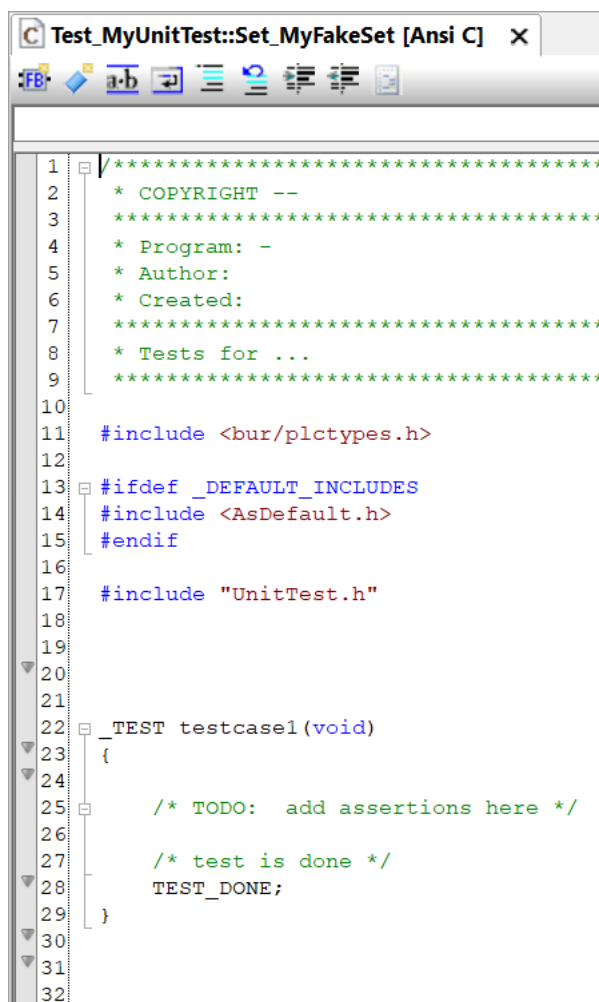
8. Open the autoGeneratedFile.c

Notice that it looks empty.



```
1  /* This content will be auto updated by
2
3  #include "UnitTest.h"
4
5
6  UNITTEST_TESTSET_FIXTURES (utTestSets)
7  {
8      {
9          0
10     }
11 };
12 UNITTEST_TESTSET_HANDLER();
13
```

9. Open Set_MyFakeSet.c



```
1
2  /* *****
3  * COPYRIGHT --
4  * *****
5  * Program: -
6  * Author:
7  * Created:
8  * *****
9  * Tests for ...
10 * *****
11 #include <bur/plctypes.h>
12
13 #ifdef _DEFAULT_INCLUDES
14 #include <AsDefault.h>
15 #endif
16
17 #include "UnitTest.h"
18
19
20
21
22 _TEST testcase1(void)
23 {
24
25     /* TODO: add assertions here */
26
27     /* test is done */
28     TEST_DONE;
29 }
30
31
32
```

Notice that there is no code after line 29.

10. Run the Update_TestPrograms_in_current_location.bat

```
C:\WINDOWS\system32\cmd.exe
COPYRIGHT Bernecker + Rainer
This program is part of the solution "UnitTest".
UnitTest_TestProgram_Helper (Version: 2.0.1.59)

Helper tool to update fixtures of ANSI C/C++ test programs.

*****
Start working on test program Test_MyUnitTest
Working on test set: Set_MyFakeSet
-----
Summary for test program Test_MyUnitTest
Test program contains 1 (0 skipped) test sets in Test_MyUnitTest (fixture list of test sets was updated)
Test set contains 1 (0 skipped) test cases in Set_MyFakeSet (fixture of test set was updated)
Done working on test program Test_MyUnitTest

*****
Exitcode : 0
Press any key to continue . . .
```

This program will create fixtures by adding code in the 2 files that we checked in the prior steps. This will allow the system to know your test exists. It's possible that the batch file can't complete its action. In this case, make sure it is located relatively at the right location as his shown in the Template folder. If it's the case, start building your project and stop it. This will re-assess the file tree and the batch file will now execute properly. You need to run the batch file every time you add a test set or a test case. If you make change within a test case, you don't need to run it. You can just build and transfer.

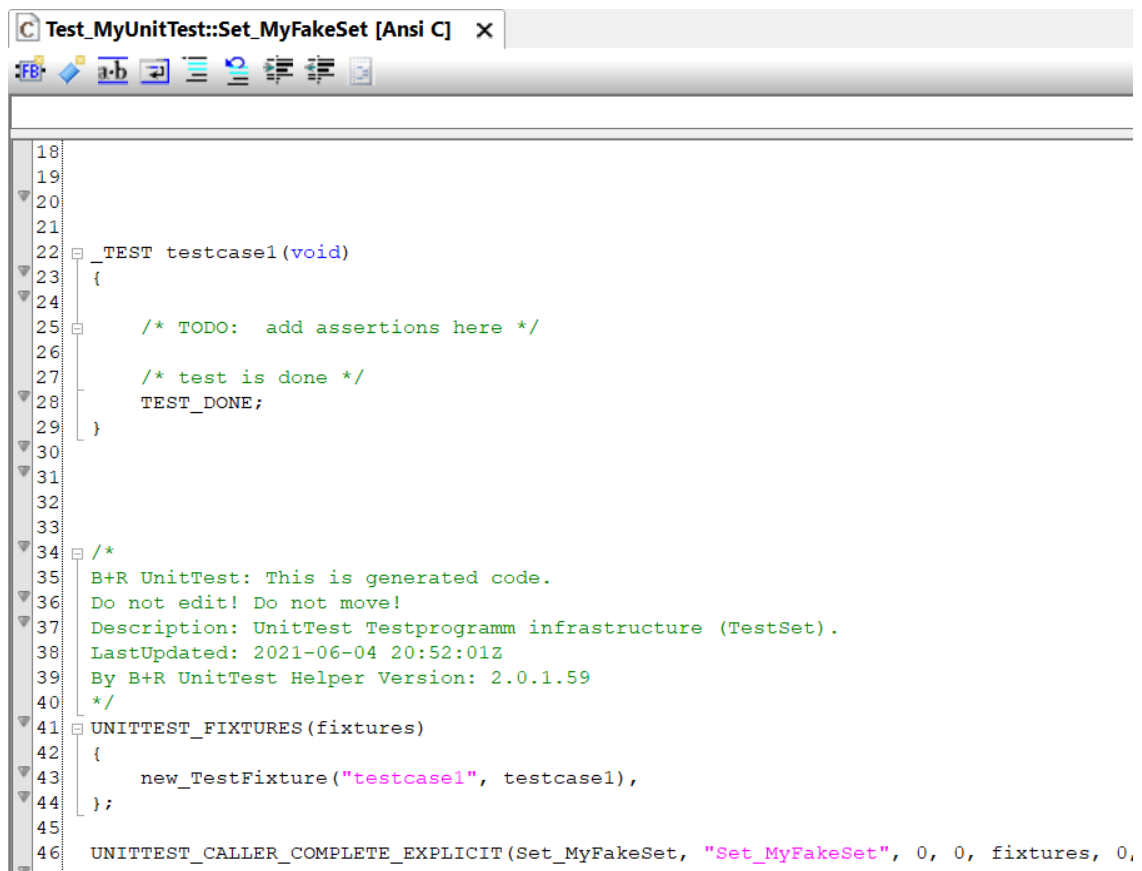
11. Verify the code that was added in autoGeneratedFile.c

```
Test_MyUnitTest::autoGeneratedFile [Ansi C] X
/*
 * B+R UnitTest: This is generated code.
 * Do not edit! Do not move!
 * Description: UnitTest Testprogramm infrastruc
 * LastUpdated: 2021-06-04 20:52:01Z
 * By B+R UnitTest Helper Version: 2.0.1.59
 */
#include "UnitTest.h"

UNITTEST_TESTSET_DECLARATION Set_MyFakeSet;

UNITTEST_TESTSET_FIXTURES(utTestSets)
{
    new_TestSet(Set_MyFakeSet),
};
UNITTEST_TESTSET_HANDLER();
```

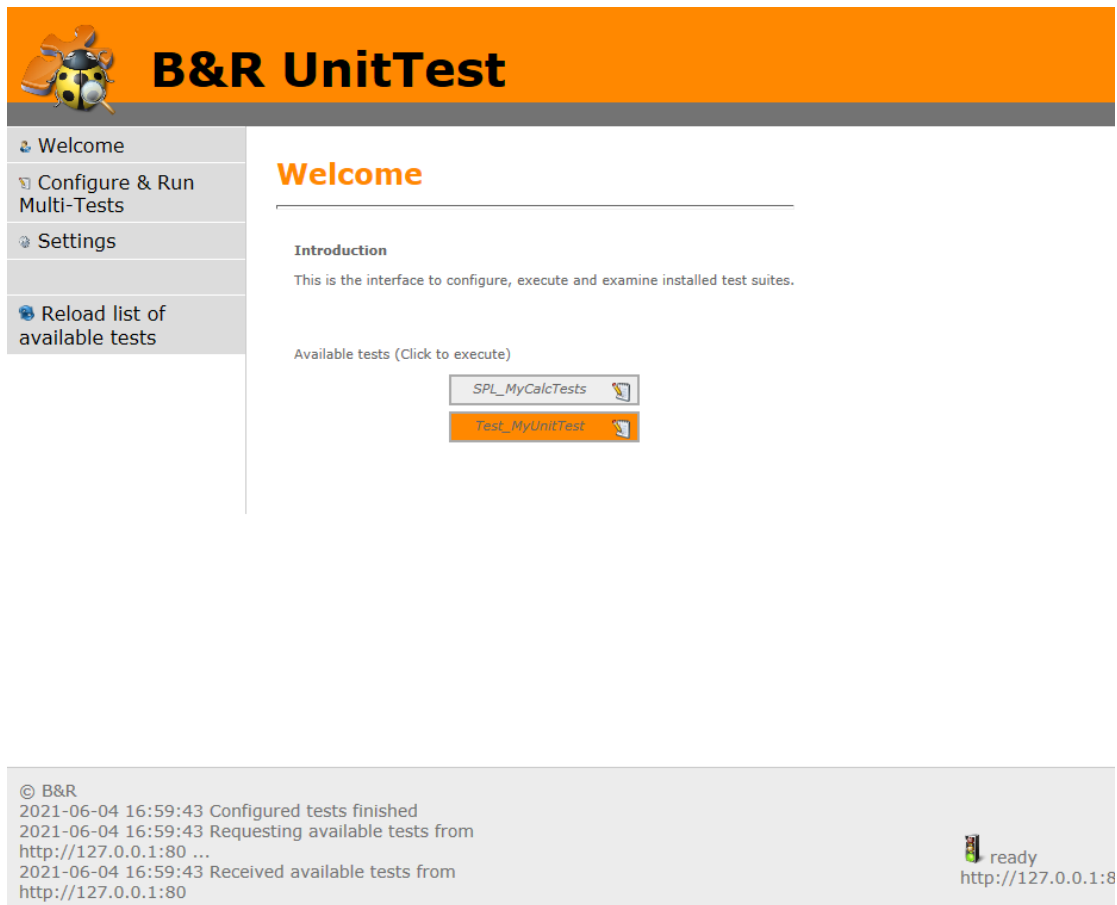
12. Verify the code that was added in Set_MyFakeSet.c



The screenshot shows a code editor window titled "Test_MyUnitTest::Set_MyFakeSet [Ansi C]". The code is written in C++ and includes a test case function and a fixture setup. The test case function is named `_TEST testcase1(void)` and contains a placeholder for assertions and a `TEST_DONE` statement. The fixture setup function is named `UNITTEST_FIXTURES(fixtures)` and creates a new `TestFixture` for the test case. The code is annotated with a B+R UnitTest version 2.0.1.59.

```
18
19
20
21
22 _TEST testcase1(void)
23 {
24
25     /* TODO: add assertions here */
26
27     /* test is done */
28     TEST_DONE;
29 }
30
31
32
33
34 /*
35 B+R UnitTest: This is generated code.
36 Do not edit! Do not move!
37 Description: UnitTest Testprogramm infrastructure (TestSet).
38 LastUpdated: 2021-06-04 20:52:01Z
39 By B+R UnitTest Helper Version: 2.0.1.59
40 */
41 UNITTEST_FIXTURES(fixtures)
42 {
43     new_TestFixture("testcase1", testcase1),
44 };
45
46 UNITTEST_CALLER_COMPLETE_EXPLICIT(Set_MyFakeSet, "Set_MyFakeSet", 0, 0, fixtures, 0,
```

13. Transfer and run your test

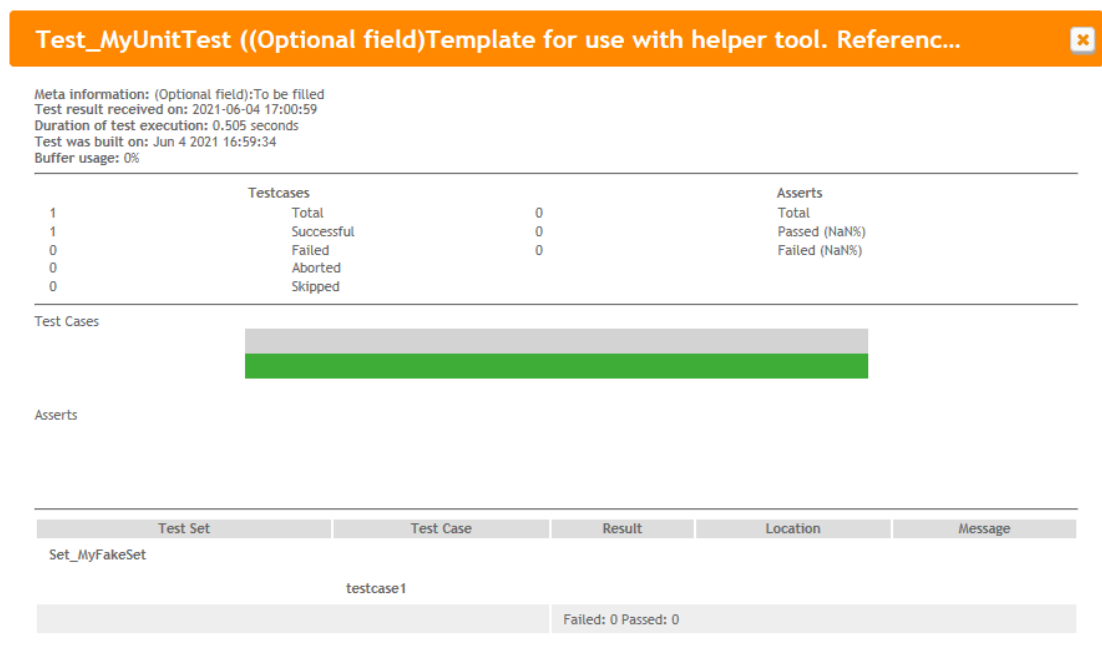


The screenshot shows the B&R UnitTest application interface. It has an orange header with a bee logo and the title "B&R UnitTest". A left sidebar contains navigation links: "Welcome", "Configure & Run Multi-Tests", "Settings", and "Reload list of available tests". The main content area is titled "Welcome" and includes an "Introduction" section stating it's an interface for configuring, executing, and examining test suites. Below this, it lists "Available tests (Click to execute)" with two buttons: "SPL_MyCalcTests" and "Test_MyUnitTest". At the bottom, a log window displays system messages and a status bar shows a "ready" icon and the URL "http://127.0.0.1:8080".

© B&R
2021-06-04 16:59:43 Configured tests finished
2021-06-04 16:59:43 Requesting available tests from
http://127.0.0.1:80 ...
2021-06-04 16:59:43 Received available tests from
http://127.0.0.1:80 ...
2021-06-04 16:59:43 Tests are available

ready
http://127.0.0.1:8080

14. Analyze the result



The screenshot shows the "Test_MyUnitTest" result window. It has an orange title bar with the text "Test_MyUnitTest ((Optional field)Template for use with helper tool. Referenc...". The main content area displays meta-information, a summary table, and a detailed table of test results.

Meta information: (Optional field):To be filled
Test result received on: 2021-06-04 17:00:59
Duration of test execution: 0.505 seconds
Test was built on: Jun 4 2021 16:59:34
Buffer usage: 0%

Testcases		Asserts
1	Total	0
1	Successful	0
0	Failed	0
0	Aborted	
0	Skipped	

Test Cases

Asserts

Test Set	Test Case	Result	Location	Message
Set_MyFakeSet	testcase1	Failed: 0 Passed: 0		

The test is now added and registered by the system. We could develop new tests by adding test cases and asserts in the test set.

6 Tips and tricks

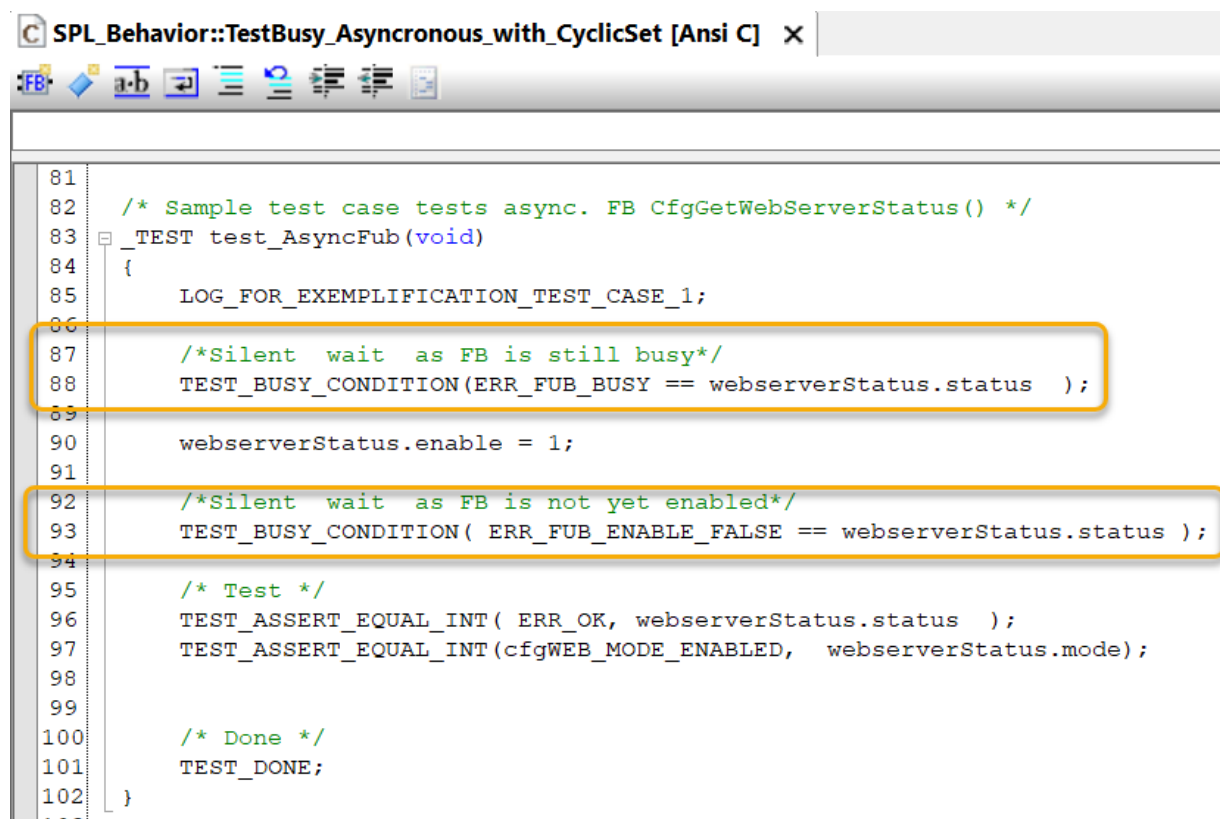
This guide will list some very useful way to use the unit test to make the user more efficient.

6.1 Asynchronous function block

It's typically easier to test a function than a function block. In a test, the developer typically has a function call followed by an assert. This usually works because the functions process the arguments immediately and the return of the function is known right away.

When using a function block, sometimes multiple scans are required before the desired output is set. This can be solved with TEST_BUSY_CONDITION

An exemple of TEST_BUSY_CONDITION is provided in the sample.



```
SPL_Behavior::TestBusy_Asyncronous_with_CyclicSet [Ansi C] X
81
82  /* Sample test case tests async. FB CfgGetWebServerStatus() */
83  _TEST test_AsyncFub(void)
84  {
85      LOG_FOR_EXEMPLIFICATION_TEST_CASE_1;
86
87      /*Silent wait as FB is still busy*/
88      TEST_BUSY_CONDITION(ERR_FUB_BUSY == webserverStatus.status );
89
90      webserverStatus.enable = 1;
91
92      /*Silent wait as FB is not yet enabled*/
93      TEST_BUSY_CONDITION( ERR_FUB_ENABLE_FALSE == webserverStatus.status );
94
95      /* Test */
96      TEST_ASSERT_EQUAL_INT( ERR_OK, webserverStatus.status );
97      TEST_ASSERT_EQUAL_INT(cfgWEB_MODE_ENABLED, webserverStatus.mode);
98
99
100     /* Done */
101     TEST_DONE;
102 }
```

It's not in the sample, but it's a very good idea to add a TEST_ABORT_CONDITION when using a TEST_BUSY_CONDITION. In the example above, the test would run indefinitely if the condition is not met. Therefore, another TEST_ABORT_CONDITION should be put before the TEST_BUSY_CONDITION in order to handle errors from the function block or timeout.

An exemple is provided in the pseudo-code below:

```
TEST_ABORT_CONDITION(MyFunctionBlock.Error || CycleCounter > 100);
TEST_BUSY_CONDITION(MyFunctionBlock.Busy);
```

6.2 Cycle counter

It's very handy to have a cycle counter when testing function block. This can be easily done by modifying the Template files.

utTemplate_automated::variables.var [Variable Declaration]

Name	Type	& R
<pre> ***** * COPYRIGHT -- ***** * Program: utTemplate_automated * File: variables.var * Author: - * Created: - ***** * Local variables of program utTemplate_automated ***** </pre>		
Describe the test		
<div> Testsuite </div>	UtMgrTestSuite	
Variables used in the tests		
<div> CycleCounter </div>	UDINT	

TestSet_Templates::Set_Complete [Ansi C]

FB

SETUP_TEST(void)

```

16
17  #include "UnitTest.h"
18
19  _CYCLIC_SET()
20  {
21
22      CycleCounter = CycleCounter + 1 ;
23      /* TODO:  add code running cyclically while test set is active here */
24      return;
25  }
26
27
28  _SETUP_SET(void)
29  {
30      /* TODO:  add code running before test set here */
31      TEST_DONE;
32  }
33
34  _TEARDOWN_SET(void)
35  {
36      /* TODO:  add code running after test set here */
37      TEST_DONE;
38  }
39
40
41  SETUP_TEST(void)
42  {
43      CycleCounter = 0;
44      /* TODO:  add code running before test case here */
45      TEST_DONE;
46  }

```

6.3 Test location

It's useful to have multiple UnitTest folder in your project. It can make sense to have one folder per module that contains library in your project. This way, if you export your module, you will also export the unit test along with it.

6.4 Integration test

This section is for advanced users only. As mentioned, it's possible to do integration test, but it requires a proper setup. The easiest way to do it is with application module. An application module can be copied in the test configuration along with a test task. This test task is mimicking the input/output of the module via .vvm mapping. Therefore, each unit inside the module has been unit tested and the unit as a whole is tested as well.