Getting started with OneTest Embedded Eclipse IDE

OneTest Embedded is delivered with some examples. For the Eclipse IDE, they are in `<installation folder>/examplesEclipse`. The following tutorial uses the example `MUIproj`. It will demonstrate the following features:

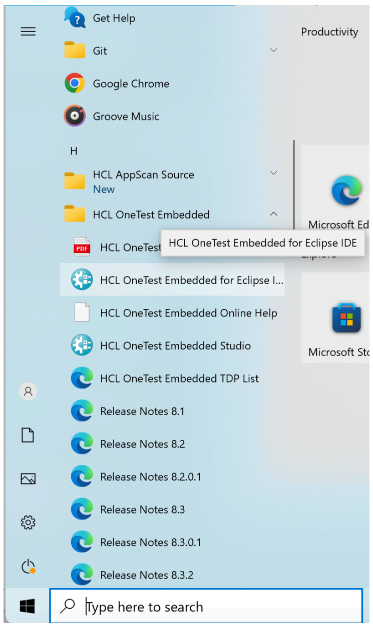
- The application build
- Code coverage
- MISRA rules review
- The call graph visualization
- The test generation
- The stub creation
- The execution of the test

1 Import the project into Eclipse

1.1 Start Eclipse IDE

⇒ **On Windows**:

- Before stating, set the environment variables `HCL_LICENSING_URL` and `HCL_LICENSING_ID` with the information provided by HCL
- Open the Windows start menu and select the menu **`HCL OneTest Embedded for Eclipse IDE`** in the group **`HCL OneTest Embedded`**

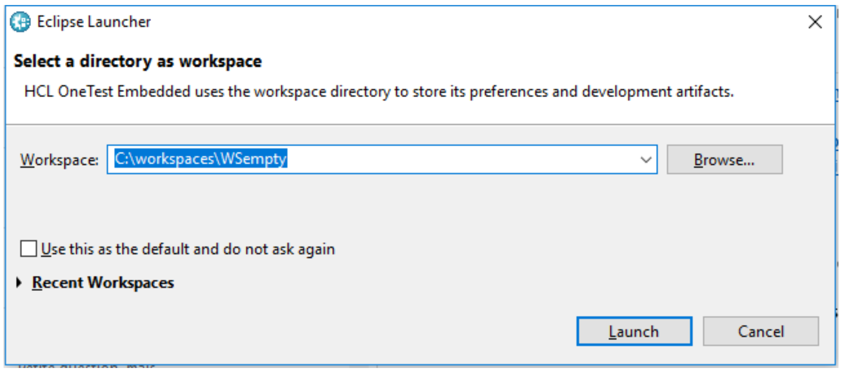


⇒ **On Linux**:

- In the installation folder, edit the file `testrtinit.sh` and update the following environment variables
  - `TESTRTDIR` with the correct installation folder

        - `HCL_LICENSING_URL` and `HCL_LICENSING_ID` with the information provided by HCL
    - Execute the command: `. testrtinit.sh`
    - Then execute the command: `. start_visualtest.sh &`

⇒ Then, create your own workspace (alternatively you can select an existing workspace).


Eclipse Launcher

Select a directory as workspace

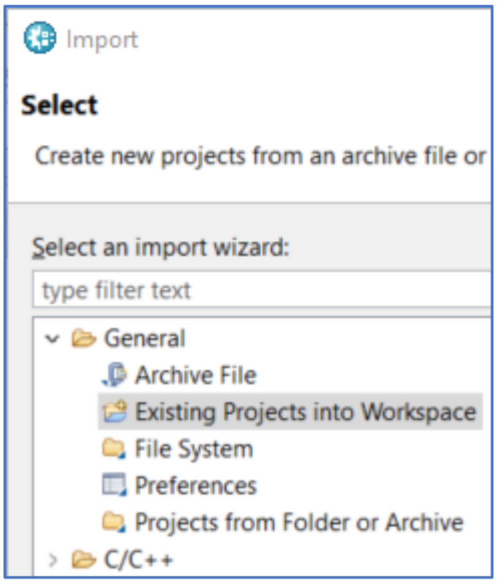
HCL OneTest Embedded uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\workspaces\WSempty

1.2 Import the project MUIproj

⇒ Select the menu **`File > Import...`**

⇒ In the opened wizard, select **`General > Existing Projects into Workspace`** and click on **`Next`**


Import

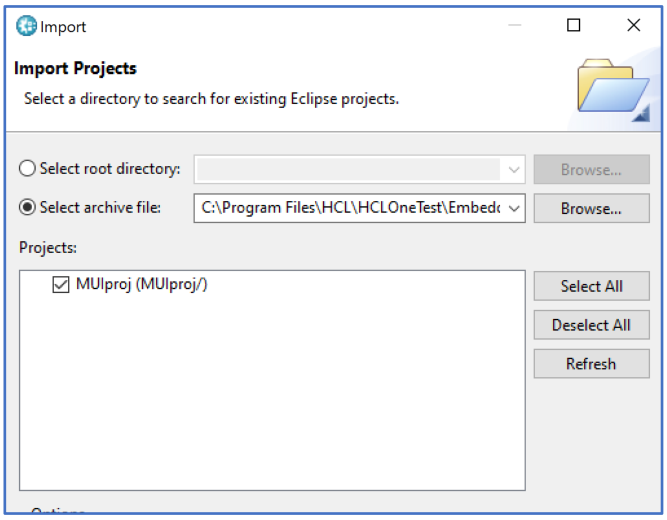
Select

Create new projects from an archive file or

Select an import wizard:

type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
Projects from Folder or Archive
C/C++

⇒ Then click on **`Select archive file`**, click on **`Browse...`** on the same line and select the file **`MUIproj.zip`** in the folder **`<installation folder>/examplesEclipse`**.

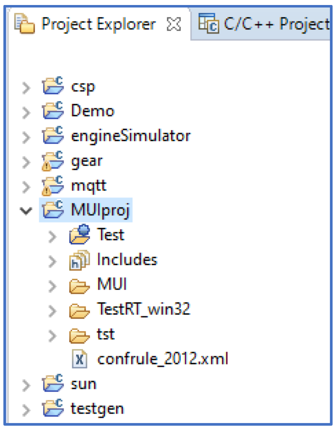


Select archive file: C:\Program Files\HCL\HCLOneTest\Embedd

Projects: ☑ MUIproj (MUIproj/)

⇒ Then click on **Finish**

A new project **MUIproj** is created. You can see it in the project explorer (if this view is not already open, you can open it by selecting the menu **Window > Show View > Project Explorer**)



2 Build and execute the application

OneTest Embedded comes with many Target Deployment Port (a.k.a. TDP) for different compilers (for more information about Target Deployment Port, please click here). This project has been initially for **C Visual Studio 2019**. If you have not this compiler already installed, or if you are on Linux, you need to change the Target Deployment Port.

When you install OneTest Embedded on your laptop, the installer checks your compilers installation and create the following TDP for you:

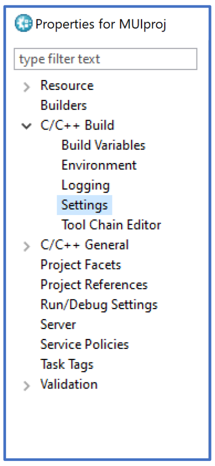
- **C GNU** if it finds a gcc native compiler (Cygwin or MinGW)
- **C Visual** if it finds a Microsoft Visual compilers

We will update this project with one of them.

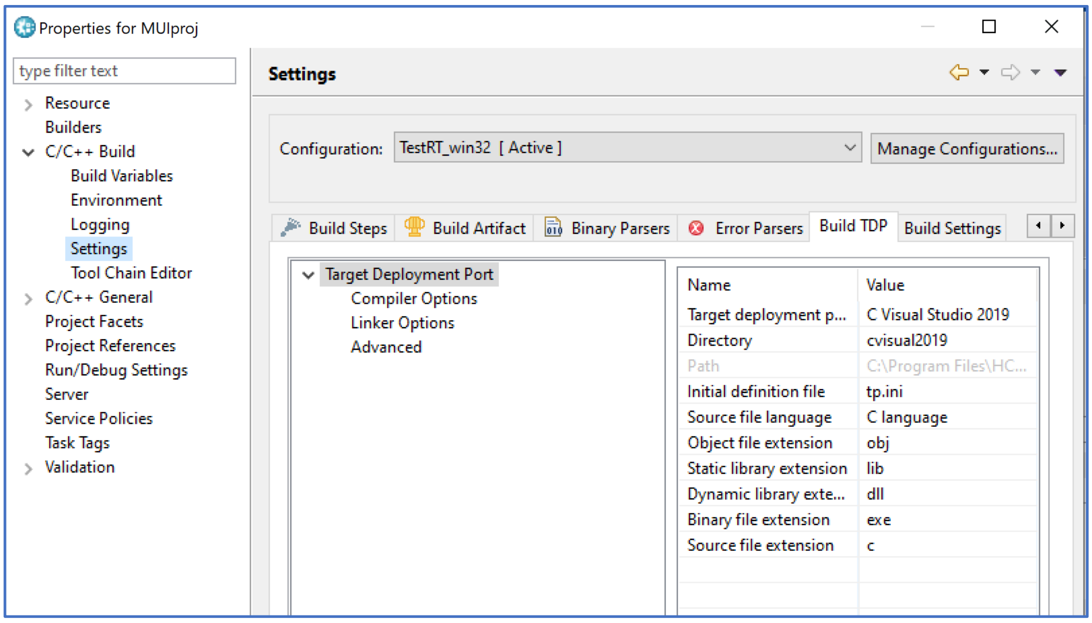
2.1 Modify the TDP in the settings

⇒ In the project explorer, right-click on the project **MUIproj** and select the menu **Properties**

⇒ In the left menu tree in the wizard, select **C/C++ Build > Settings**



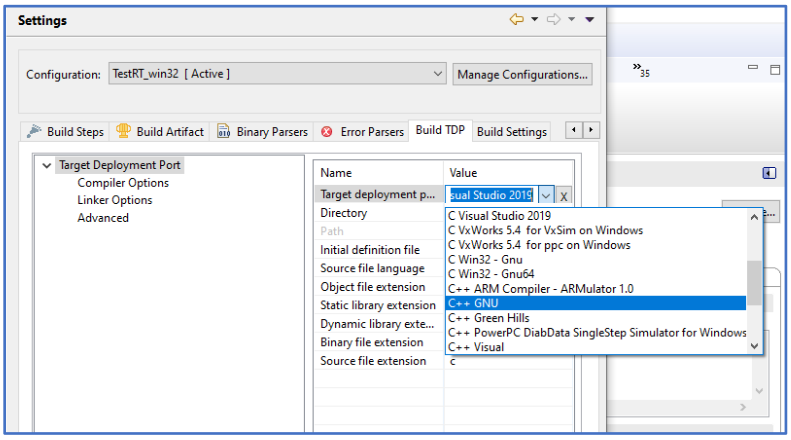
⇒ In the right panel of the wizard, select the tab `Build TDP` (if this tab is not displayed, increase the width of the wizard or use the right arrow to make it appears).



Configuration: TestRT_win32 [Active]

Name	Value
Target deployment p...	C Visual Studio 2019
Directory	cvisual2019
Path	C:\Program Files\HC...
Initial definition file	tp.ini
Source file language	C language
Object file extension	obj
Static library extension	lib
Dynamic library exte...	dll
Binary file extension	exe
Source file extension	c

⇒ In `Target Deployment Port` property, click on `C Visual Studio 2019` to make the dropdown appear and select `C GNU` (or `C Visual` depending on the compiler you have on your machine)



Configuration: TestRT_win32 [Active]

Name	Value
Target deployment p...	sual Studio 2019
Directory	
Path	
Initial definition file	
Source file language	
Object file extension	
Static library extension	
Dynamic library exte...	
Binary file extension	
Source file extension	c

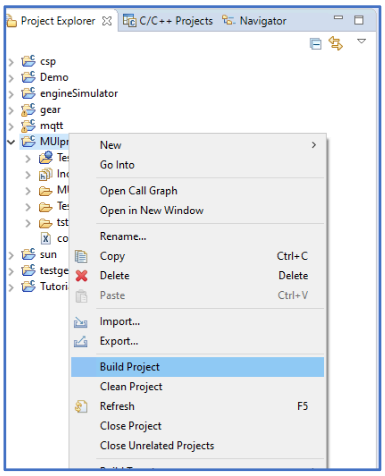
C Visual Studio 2019
C VxWorks 5.4 for VxSim on Windows
C VxWorks 5.4 for ppc on Windows
C Win32 - Gnu
C Win32 - Gnu64
C++ ARM Compiler - ARMulator 1.0
C++ GNU
C++ Green Hills
C++ PowerPC DiabData SingleStep Simulator for Windows
C++ Visual

⇒ Then click on `Apply and Close`

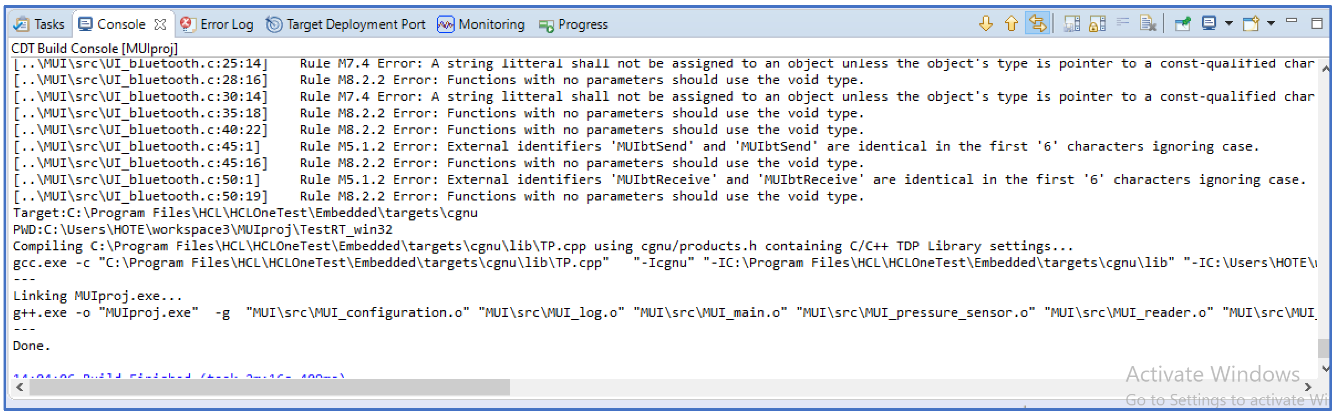
2.2 Build the project

Now this project can be built using this TDP. By default, the build will be done with **Coverage** and **MISRA Code Review** options engaged.

⇒ In the project explorer, right-click on the project `MUIproj` and select the menu `Build Project`



The console view should display the build log. At the end, the build should be completed until the link phase with success (it could take some minutes).

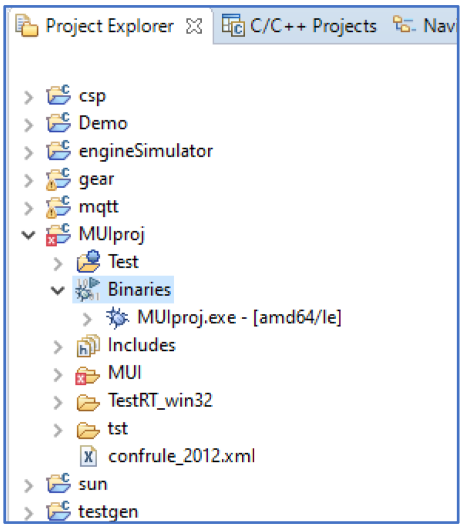


```
[..\MUI\src\UI_bluetooth.c:25:14]    Rule M7.4 Error: A string litteral shall not be assigned to an object unless the object's type is pointer to a const-qualified char
[..\MUI\src\UI_bluetooth.c:28:16]    Rule M8.2.2 Error: Functions with no parameters should use the void type.
[..\MUI\src\UI_bluetooth.c:30:14]    Rule M7.4 Error: A string litteral shall not be assigned to an object unless the object's type is pointer to a const-qualified char
[..\MUI\src\UI_bluetooth.c:35:18]    Rule M8.2.2 Error: Functions with no parameters should use the void type.
[..\MUI\src\UI_bluetooth.c:40:22]    Rule M8.2.2 Error: Functions with no parameters should use the void type.
[..\MUI\src\UI_bluetooth.c:45:1]     Rule M5.1.2 Error: External identifiers 'MUIbtSend' and 'MUIbtSend' are identical in the first '6' characters ignoring case.
[..\MUI\src\UI_bluetooth.c:45:16]    Rule M8.2.2 Error: Functions with no parameters should use the void type.
[..\MUI\src\UI_bluetooth.c:50:1]     Rule M5.1.2 Error: External identifiers 'MUIbtReceive' and 'MUIbtReceive' are identical in the first '6' characters ignoring case.
[..\MUI\src\UI_bluetooth.c:50:19]    Rule M8.2.2 Error: Functions with no parameters should use the void type.
Target:C:\Program Files\HCL\HCLOneTest\Embedded\targets\cgnu
PWD:C:\Users\HOTE\workspace3\MUIproj\TestRT_win32
Compiling C:\Program Files\HCL\HCLOneTest\Embedded\targets\cgnu\lib\TP.cpp using cgnu/products.h containing C/C++ TDP Library settings...
gcc.exe -c "C:\Program Files\HCL\HCLOneTest\Embedded\targets\cgnu\lib\TP.cpp"   "-Icgnu" "-IC:\Program Files\HCL\HCLOneTest\Embedded\targets\cgnu\lib" "-IC:\Users\HOTE\
---
Linking MUIproj.exe...
g++.exe -o "MUIproj.exe"  -g  "MUI\src\MUI_configuration.o" "MUI\src\MUI_log.o" "MUI\src\MUI_main.o" "MUI\src\MUI_pressure_sensor.o" "MUI\src\MUI_reader.o" "MUI\src\MUI_
---
Done.
```

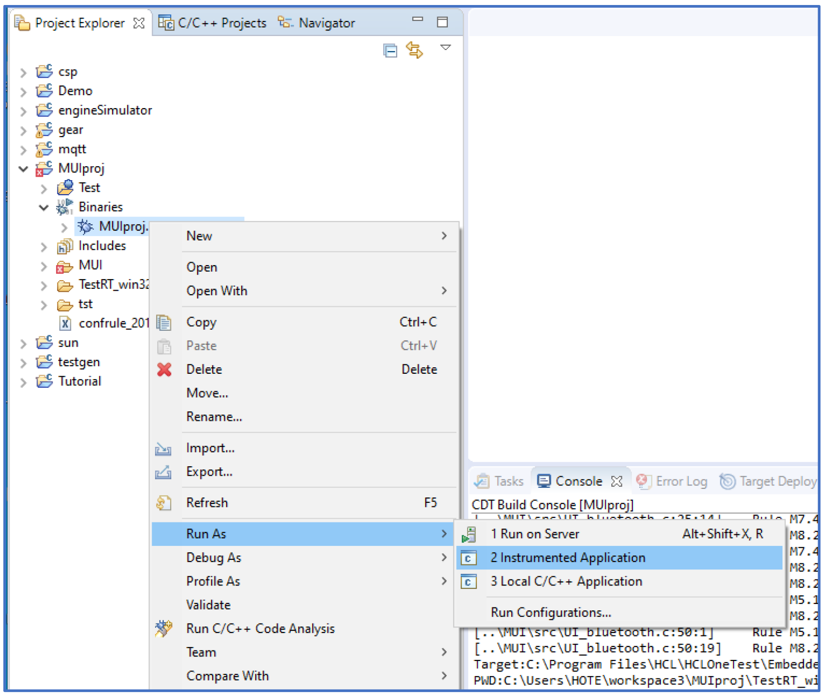
2.3 Execute the application

This application contains a simple main that can be executed.

⇒ In the project explorer, open the node `Binaries` in the `MUIproj` project



⇒ Right-click on `MUIproj.exe` and select the menu `Run as > Instrumented Application`



CDT Build Console [MUIproj]

```
[..\MUI\src\UI_bluetooth.c:50:1]    Rule M5.1
[..\MUI\src\UI_bluetooth.c:50:19]   Rule M8.2
Target:C:\Program Files\HCL\HCLOneTest\Embedde
PWD:C:\Users\HOTE\workspace3\MUIproj\TestRT_wi
```

This menu will execute the just-compiled application and then will launch different tools to generate reports depending on the settings.

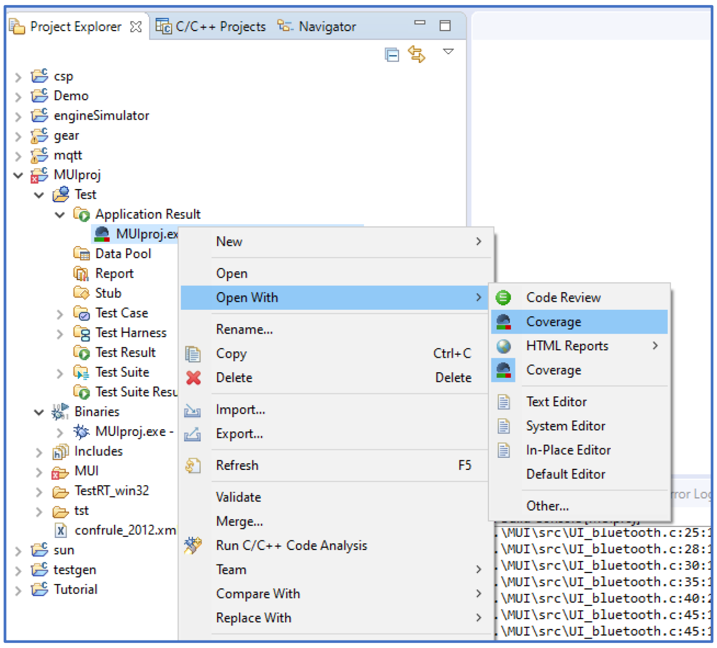
Note: if Eclipse prompts you in case of errors in the project, ignore it and click on `Yes`.

2.4 See code coverage report

After the execution, all the reports are gathered in a single file that you can find under the virtual node `Test`.

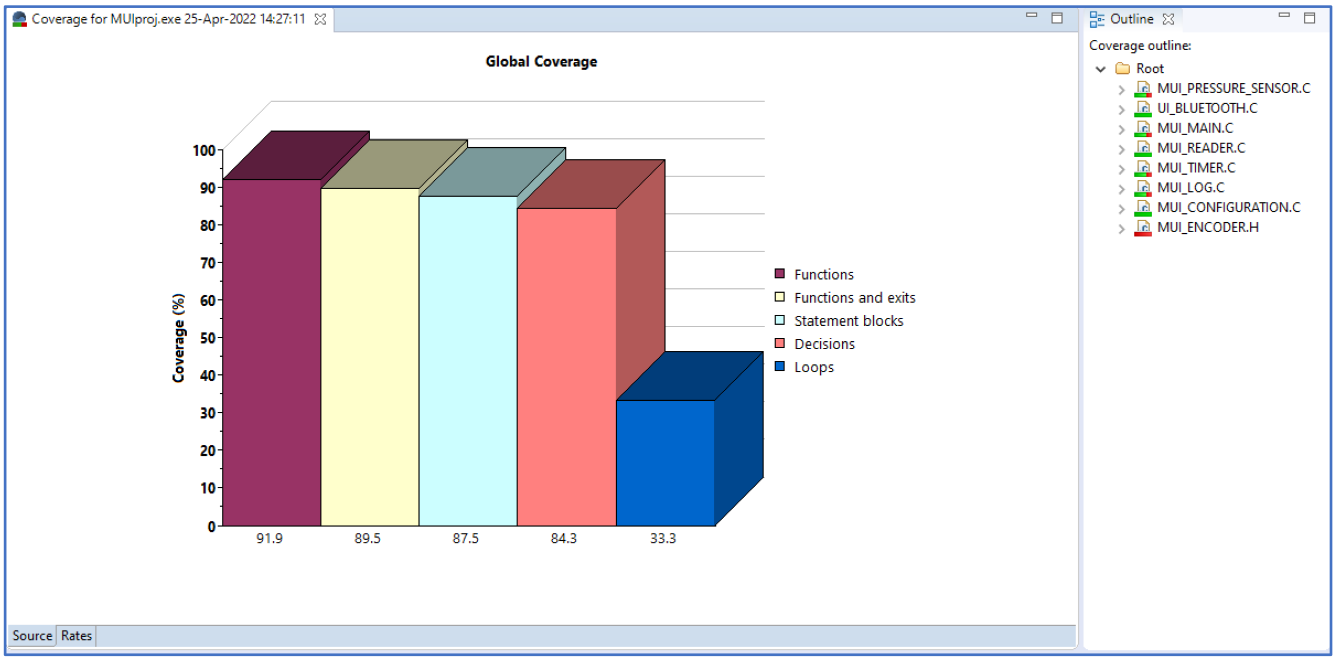
⇒ In the project explorer, open the node `Test > Application Result` in the `MUIproj` project. There is a new file called `MUIproj.exe` with the date of the execution and a status.

⇒ Right-click on this file and select the menu `Open With > Coverage`



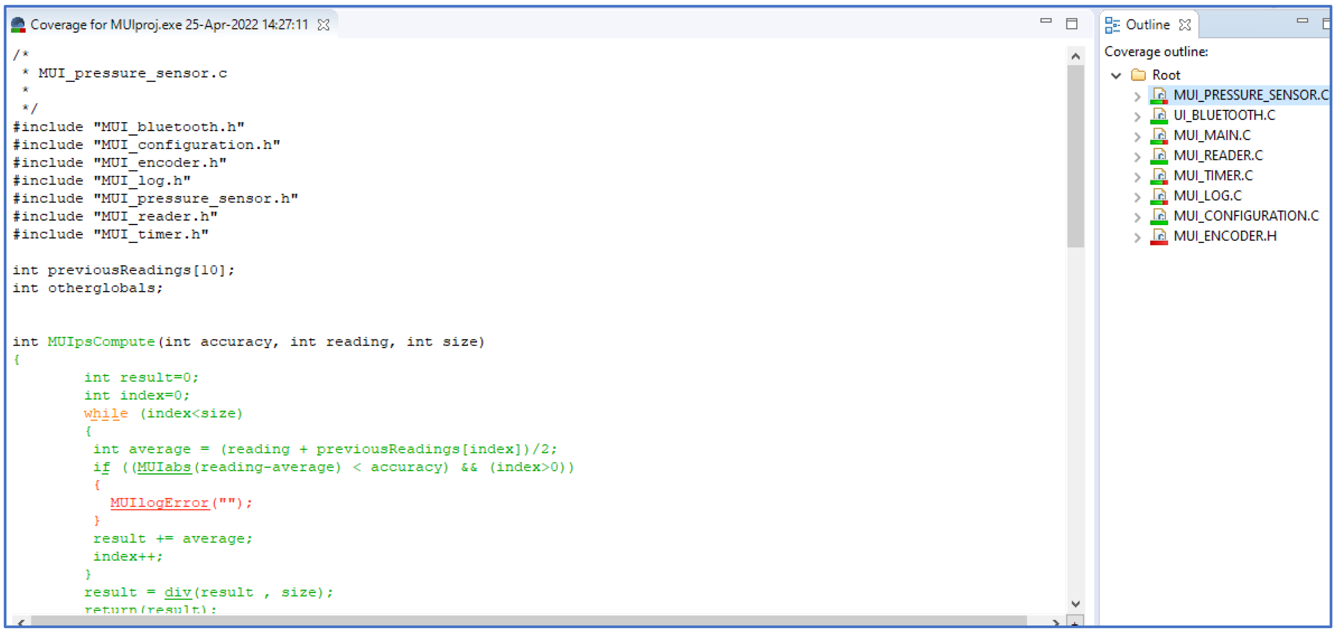
```
\MUI\src\UI_bluetooth.c:25:
\MUI\src\UI_bluetooth.c:28:
\MUI\src\UI_bluetooth.c:30:
\MUI\src\UI_bluetooth.c:35:
\MUI\src\UI_bluetooth.c:40:
\MUI\src\UI_bluetooth.c:45:
\MUI\src\UI_bluetooth.c:45:
```

The coverage viewer is opened now, showing a graph with the different coverage level percentages (for more information about the coverage levels, click here and here).



The outline view (here on the right) allows you to navigate on the source code for each compilation unit.

⇒ Click on one of them. A copy of the source code is now displayed with different colors:
- **Green**: the code has been covered
- **Red**: the code has been partially covered
- **Orange**: the code is partially covered
- **Black**: not code



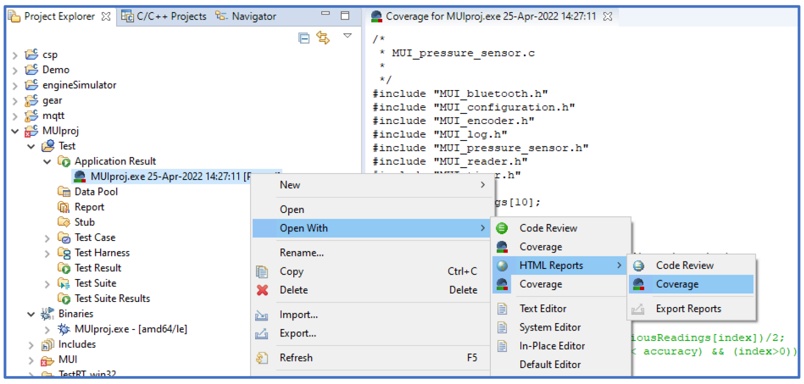
```c
/*
 * MUI_pressure_sensor.c
 *
 */
#include "MUI_bluetooth.h"
#include "MUI_configuration.h"
#include "MUI_encoder.h"
#include "MUI_log.h"
#include "MUI_pressure_sensor.h"
#include "MUI_reader.h"
#include "MUI_timer.h"

int previousReadings[10];
int otherglobals;


int MUIpsCompute(int accuracy, int reading, int size)
{
        int result=0;
        int index=0;
        while (index<size)
        {
         int average = (reading + previousReadings[index])/2;
         if ((MUIabs(reading-average) < accuracy) && (index>0))
         {
           MUIlogError("");
         }
         result += average;
         index++;
        }
        result = div(result , size);
        return(result);
```

For more details on this page, please click here. A similar report exists in HTML. You can open it in a browser:

⇒ Right-click on the same file and select the menu `Open With > HTML Reports > Coverage`

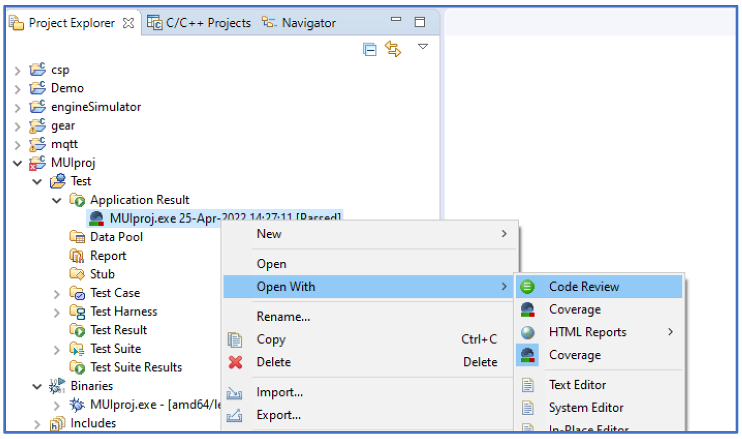


```
/*
 * MUI_pressure_sensor.c
 *
 */
#include "MUI_bluetooth.h"
#include "MUI_configuration.h"
#include "MUI_encoder.h"
#include "MUI_log.h"
#include "MUI_pressure_sensor.h"
#include "MUI_reader.h"
```

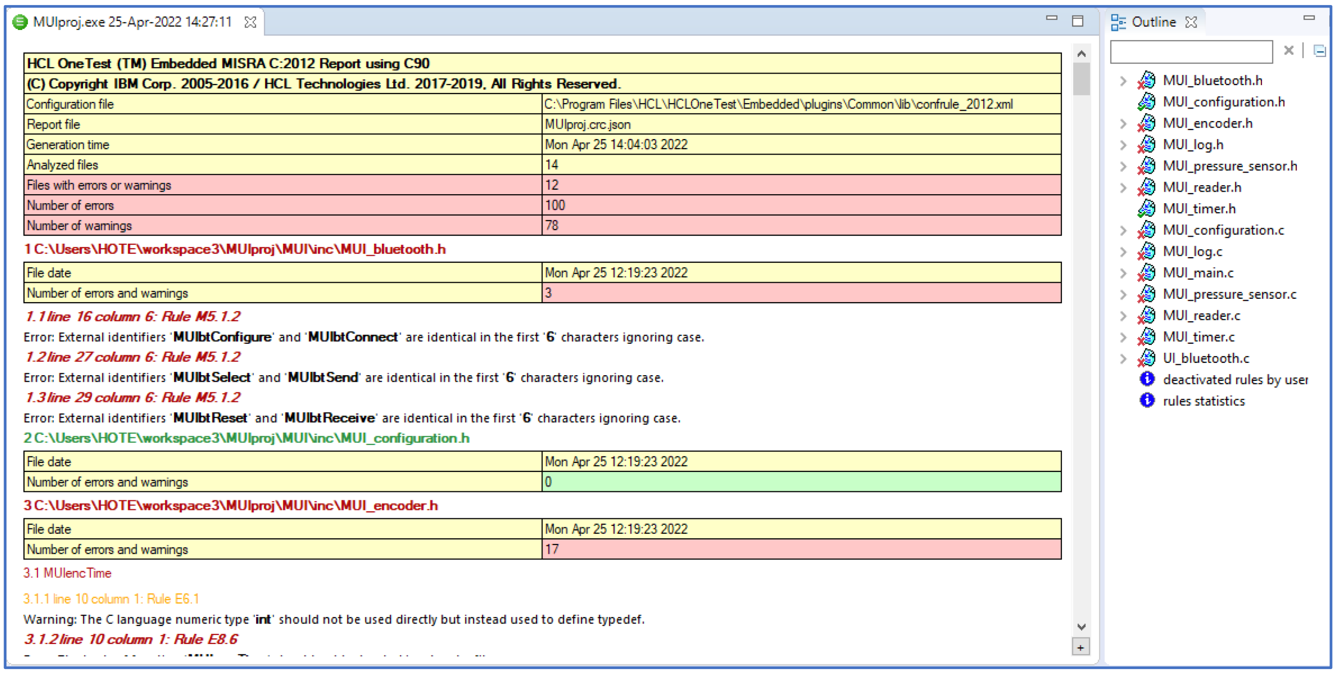
2.5 See MISRA code review report

OneTest Embedded supports MISRA C 2004 (click here for the detailed description of the MISRA C 2004 rules) & MISRA C 2012 (click here for the detailed description of the MISRA C 2012 rules). The report generated by this feature could be opened in a similar way.

⇒ Right-click on the node `MUIproj.exe` in the `Application Result` node this and select the menu `Open With > Code Review`



The code review viewer is opened now. The outline view allows you to navigate in the different files of the application (.h and .c). The central panel displays the rules that have been raised during the analysis for the selected file. If you click on a rule, the source code editor will open in the selected file, at the line where the error has been found.



HCL OneTest (TM) Embedded MISRA C:2012 Report using C90	
(C) Copyright IBM Corp. 2005-2016 / HCL Technologies Ltd. 2017-2019. All Rights Reserved.	
Configuration file	C:\Program Files\HCL\HCLOneTest\Embedded\plugins\Common\lib\confrule_2012.xml
Report file	MUIproj.crc.json
Generation time	Mon Apr 25 14:04:03 2022
Analyzed files	14
Files with errors or warnings	12
Number of errors	100
Number of warnings	78

1 C:\Users\HOTE\workspace3\MUIproj\MUI\inc\MUI_bluetooth.h

File date	Mon Apr 25 12:19:23 2022
Number of errors and warnings	3

1.1 line 16 column 6: Rule M5.1.2

Error: External identifiers '**MUIbtConfigure**' and '**MUIbtConnect**' are identical in the first '**6**' characters ignoring case.

1.2 line 27 column 6: Rule M5.1.2

Error: External identifiers '**MUIbtSelect**' and '**MUIbtSend**' are identical in the first '**6**' characters ignoring case.

1.3 line 29 column 6: Rule M5.1.2

Error: External identifiers '**MUIbtReset**' and '**MUIbtReceive**' are identical in the first '**6**' characters ignoring case.

2 C:\Users\HOTE\workspace3\MUIproj\MUI\inc\MUI_configuration.h

File date	Mon Apr 25 12:19:23 2022
Number of errors and warnings	0

3 C:\Users\HOTE\workspace3\MUIproj\MUI\inc\MUI_encoder.h

File date	Mon Apr 25 12:19:23 2022
Number of errors and warnings	17

3.1 MUIencTime

3.1.1 line 10 column 1: Rule E6.1

Warning: The C language numeric type '**int**' should not be used directly but instead used to define typedef.

3.1.2 line 10 column 1: Rule E8.6

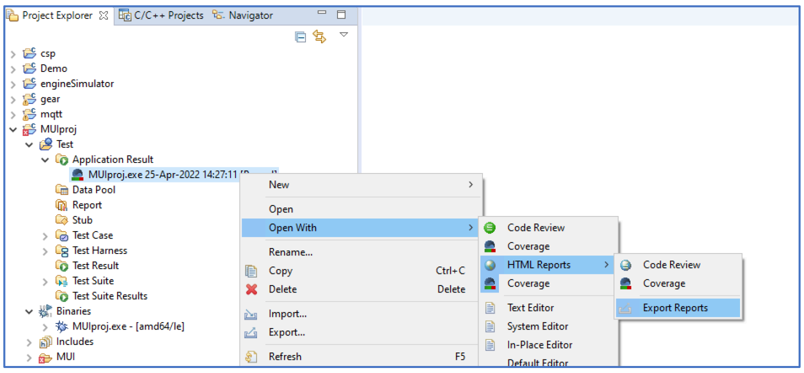
A similar report exists in HTML. You can open it in a browser:

⇒ Right-click on the same file and select the menu `Open With > HTML Reports > Code Review`

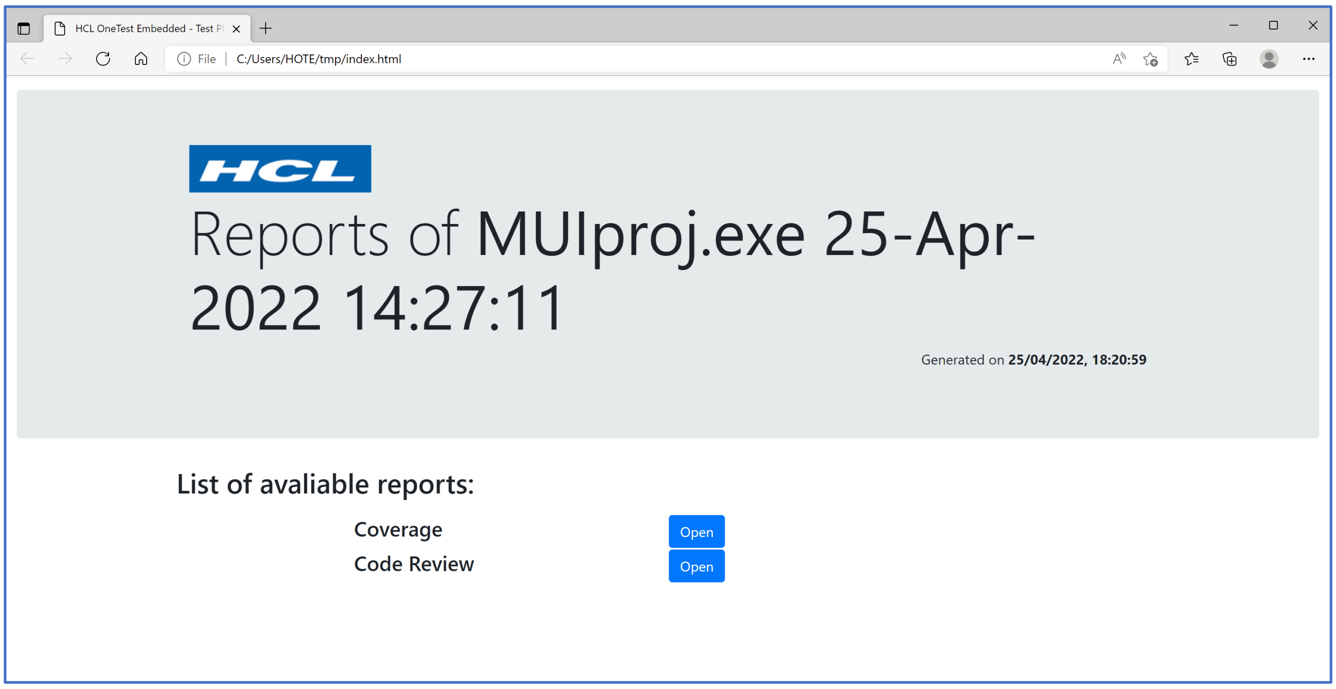
2.6 Export the HTML reports

As described earlier, all the reports, including the HTML reports, are store in a single file (a zip file). You can easily extract only the HTML reports in a folder with an index that lists all the exported files.

⇒ Right-click on the result file and select the menu `Open With > HTML Reports > Export Reports`



⇒ Select a folder on your disk (or create a new one) and click OK. Then, a browser opens with the index as following:


HCL OneTest Embedded - Test P

File | C:/Users/HOTE/tmp/index.html

HCL

Reports of MUIproj.exe 25-Apr-2022 14:27:11

Generated on **25/04/2022, 18:20:59**

List of avaliable reports:

Coverage	Open
Code Review	Open

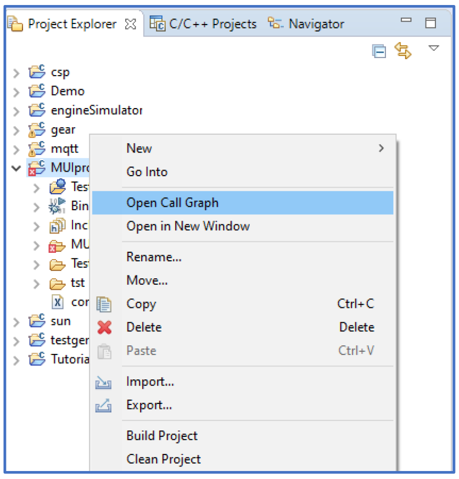
3 Create a test case

The next sections will show you how to create, update and execute a test case with OneTest Embedded.

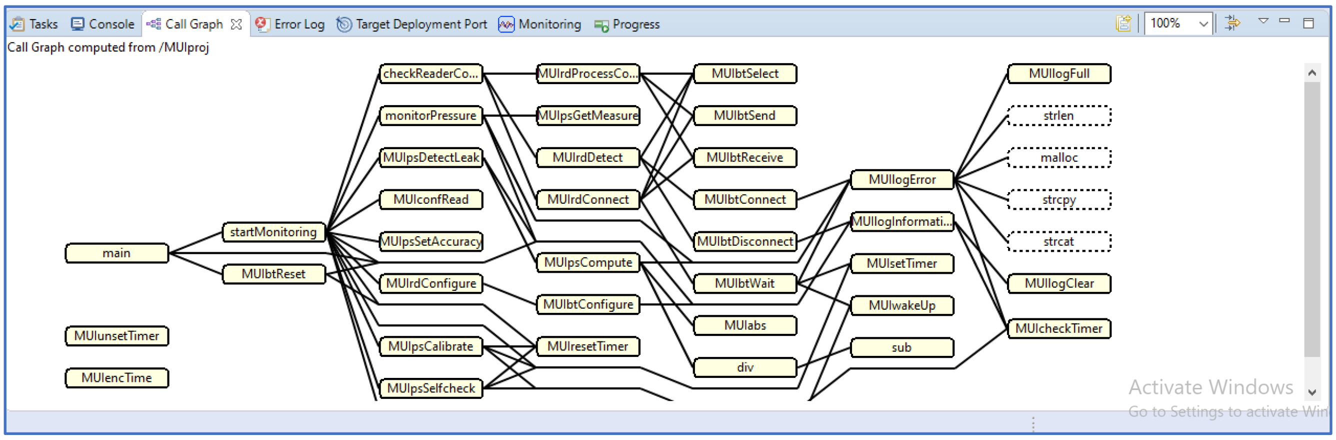
3.1 Open the call graph

There are many ways to create a test case. The one we will explore use the call graph.

⇒ In the project explorer, right-click on the project **MUIproj** and select the menu **Open Call Graph**

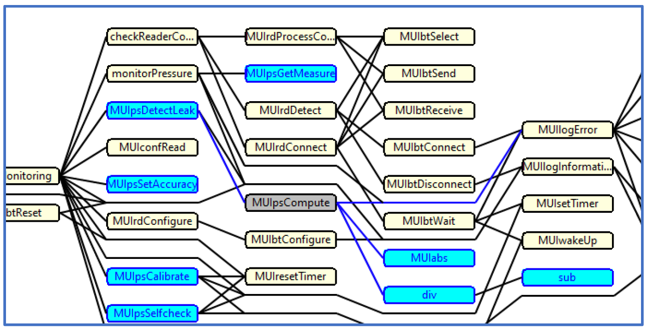


This action will open a new view containing the call graph of the application:



The nodes of this graph are the functions of the application. Nodes with dotted line are functions for which we don't have the source code (in this example, these are functions in `libc`). The lines between 2 nodes are the calls between functions. The top level function is at the left of the graph (in this example it is the function `main`) and low level functions on the right.

⇒ Click on one node inside the call graph (in the following example, the node **MUIpsCompute** has been selected)



Now the call graph highlights the following:

- The grey node is the selected function
- The blue lines on the right of this node are calls to other functions
- The blue lines on the left of this node are link to caller functions
- The blue nodes are functions that are in the same compilation unit

⇒ Double click on a node, the corresponding source code will open in the editor.

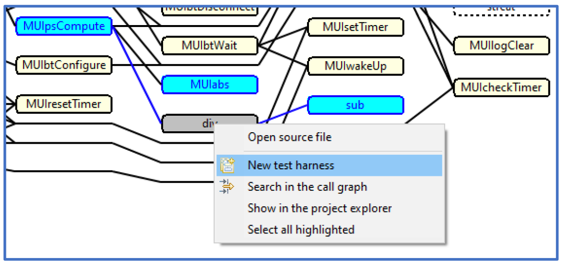
3.2 Create a test

We will now create a test for the function **`div`**. In OneTest Embedded, a test is composed of 2 parts:

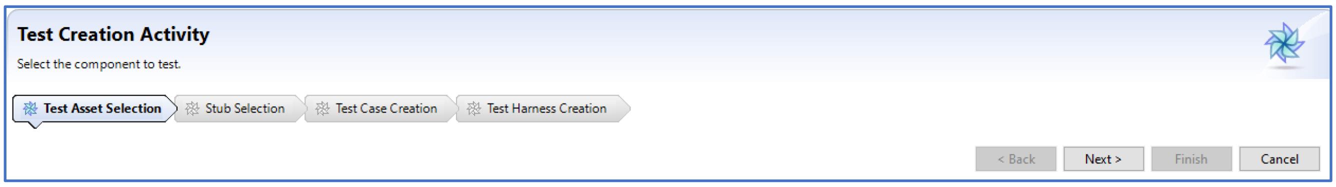
- A test case: it is the test itself. It contains:
  - The call of the function under test.
  - A table with the initial values and the expected values of the parameters, the global variables and local variables that you can add in this test case.
  - The stub behaviors relative to this test case.
  - Optionally, code that will be added at the beginning (`#include` for example) and one or several requirements.
- A test harness: this is the container of the test cases (one test harness can contain several test cases). It defines how the test will be built to become an executable. It contains:
  - The list of the files under test.
  - Additional source files that can be added to the test harness when linking. This option is useful for software integration test. It is also possible to add object files and libraries.
  - The build settings.
  - Optionally, code that will be added at the beginning (`#include` for example) and one or several requirements (in such case, these requirements will cover all the test cases of the test harness).

In OneTest Embedded, test cases and test harnesses are files with the extensions `.test_case` and `.test_harness`. There are compressed XML files.

⇒ Right-click on the node **`div`** and select the menu **`New test harness`**



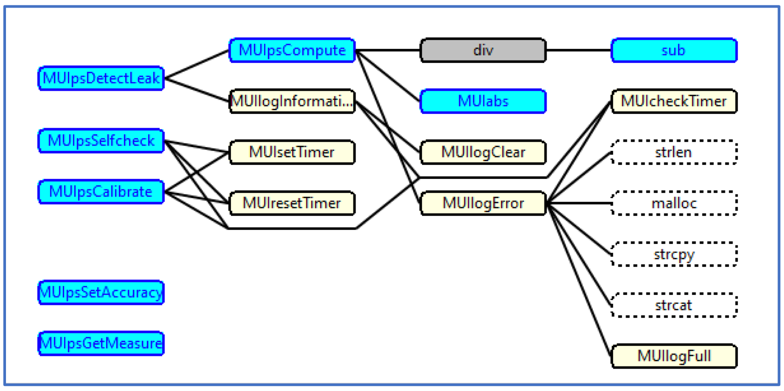
A new panel appears at the top of the call graph, called **Test Creation Activity**:


Test Creation Activity

Select the component to test.

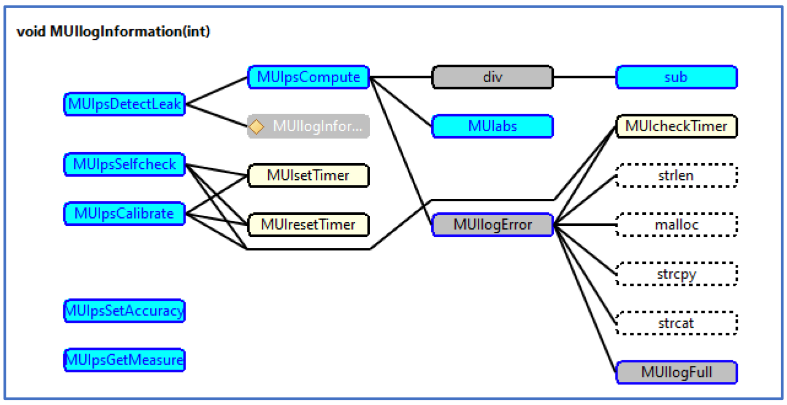
This is a wizard to help us to create a first test case.

⇒ Click on **`Next`**. This is the second page of the wizard. Now the call graph is reduced only to the functions that are in the same compilation unit (in blue) and the functions that are called by previous one:



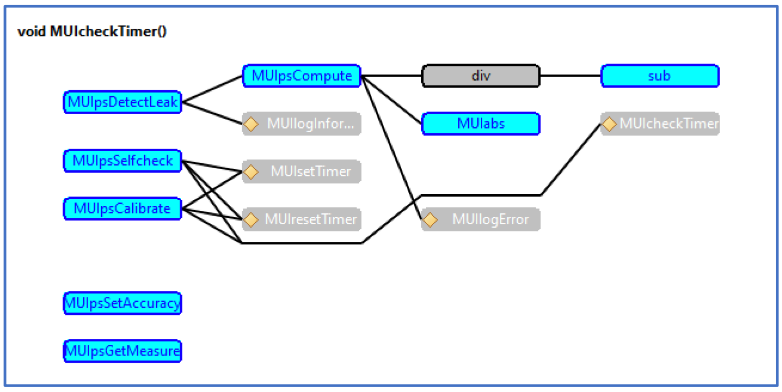
The objective of this step is to take into account only the functions that are required in your test harness to link properly with your compilation unit without error. By default, the test harness will be linked only with the compilation unit of the function under test. So, all the referenced functions (nodes in yellow) will generate an error at the link phase as they will be missing. The way to avoid that is to stub them.

⇒ Click on the node **`MUILogInformation`**



This node is now displayed as a stub, and the node **`MUILogClear`** disappears from the call graph because its caller will be stubbed.

⇒ Continue the sub selection by clicking on the nodes **`MUIsetTimer`**, **`MUIresetTimer`**, **`MUILogError`** and **`MUIcheckTimer`**.



At the end, all the yellow nodes should have been supressed and only the blue one are ramaining.

⇒ Click on **`Next`**. The next page is the test case name. By default, it is the name of the function under test. Let's go with the default.

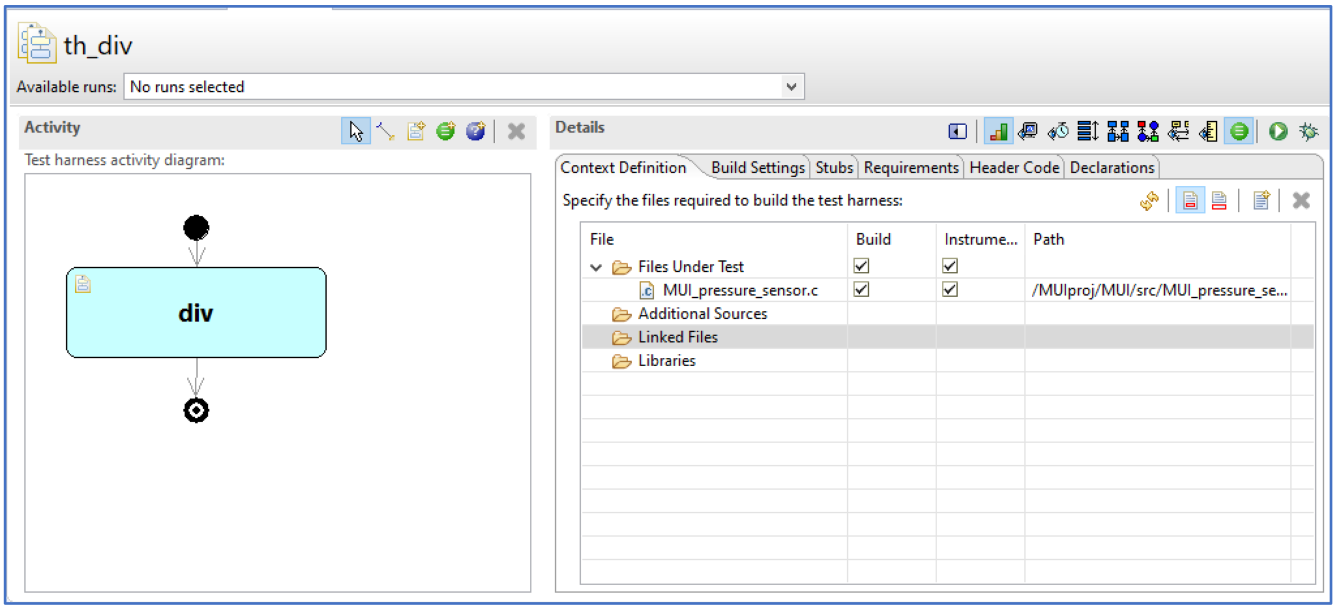
⇒ Click on **`Next`**. The next page is the test harness name. By default, it is the name of the function under test with the suffix `th_`. Let's go with the default.

⇒ Click on **`Finish`**.

3.3 Edit a test

At the end of the wizard, a test harness that includes one test case is created. This test harness should be open. It contains 2 parts:

- On the left panel, an activity diagram that allows to chain the execution of the test cases
- On the right panel, the configuration of the test harness



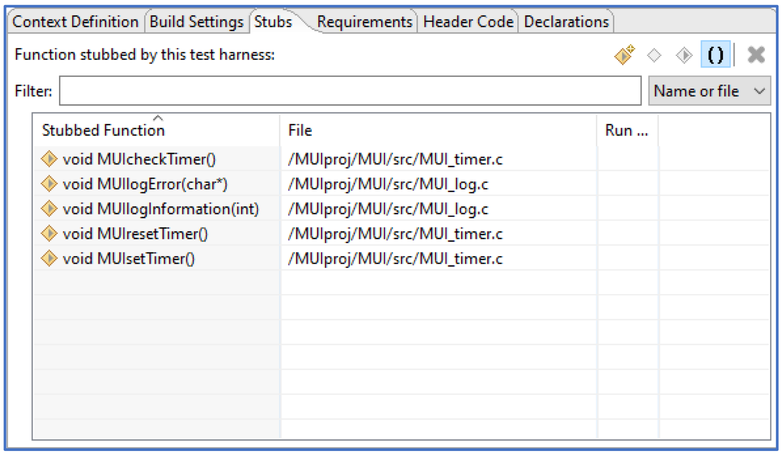
Specify the files required to build the test harness:

File	Build	Instrume...	Path
Files Under Test	✓	✓	
MUI_pressure_sensor.c	✓	✓	/MUIproj/MUI/src/MUI_pressure_se...
Additional Sources			
Linked Files			
Libraries			

For now, let's have a look only on the following ones:

- **Context definition**: Only one file has been added as files under test because we have stubbed all the external calls.

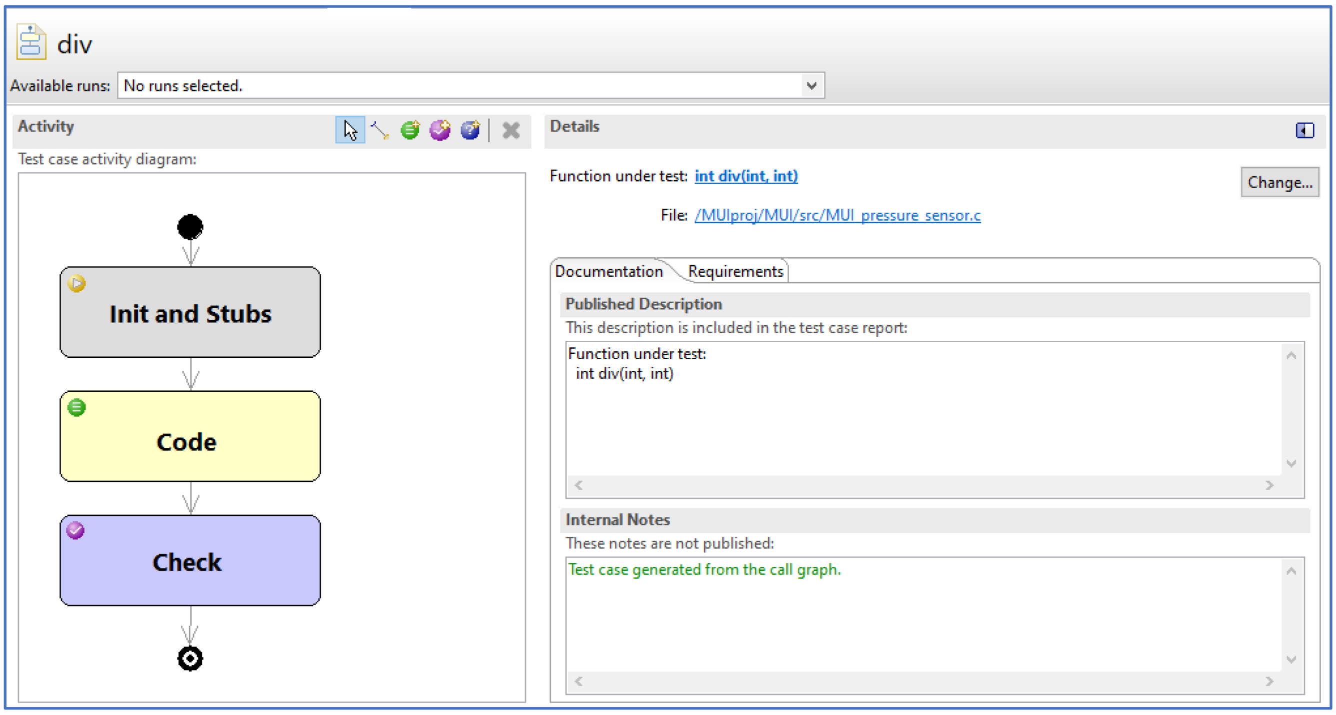
  Note that this icon indicates that the files under test will be included in the code generated for the test harness. This will help you to have the visibility on static variables and static functions that are hidden from external compilation units.
- **Stubs**: You can see here the 5 selected stubbed functions.



Function stubbed by this test harness:

Stubbed Function	File	Run ...
void MUIcheckTimer()	/MUIproj/MUI/src/MUI_timer.c	
void MUIlogError(char*)	/MUIproj/MUI/src/MUI_log.c	
void MUIlogInformation(int)	/MUIproj/MUI/src/MUI_log.c	
void MUIresetTimer()	/MUIproj/MUI/src/MUI_timer.c	
void MUIsetTimer()	/MUIproj/MUI/src/MUI_timer.c	

⇒ To open the test case, double click on the box `div` in the activity diagram. It contains 2 parts:
  - On the left panel, an activity diagram that display the different phases of the test case
  - On the right panel, the configuration of the phase that is selected on the activity diagram (the default one when opening the test case is its general description, that you can open when clicking on the background of the activity diagram)



Function under test: int div(int, int)

File: /MUIproj/MUI/src/MUI_pressure_sensor.c

Published Description

This description is included in the test case report:

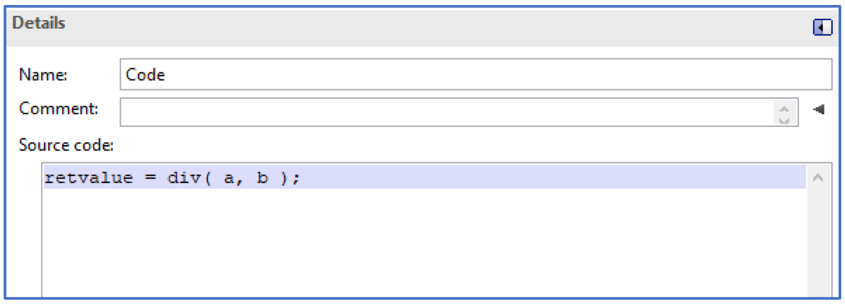
Function under test:
int div(int, int)

Internal Notes

These notes are not published:

Test case generated from the call graph.

⇒ Click on the **Code** box in the activity diagram: this shows you the generated code for calling the function under test. It could be edited.



Name: Code

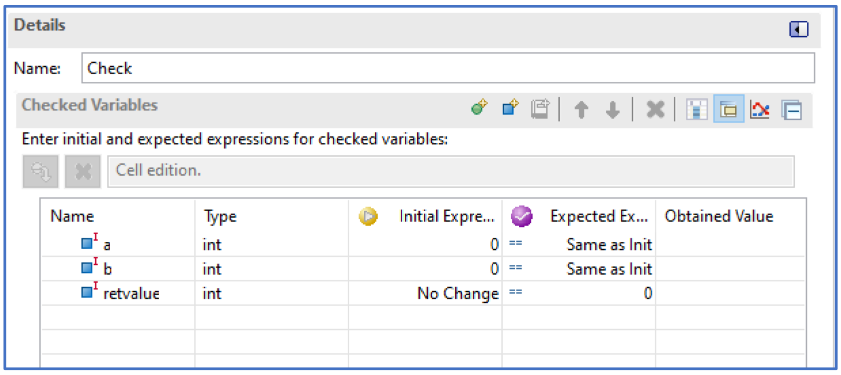
Comment:

Source code:

```
retvalue = div( a, b );
```

The variables `retValue`, `a` and `b` will be created locally in the test harness and used as parameters.

⇒ Click on the **Check** box in the activity diagram



Name: Check

Enter initial and expected expressions for checked variables:

Name	Type	Initial Expre...		Expected Ex...	Obtained Value
a	int	0	==	Same as Init	
b	int	0	==	Same as Init	
retvalue	int	No Change	==	0	

This table displays all the variables used for the test (parameters and global variables) with:

- Their type
- Their initial expression (i.e. the value before calling the function under test). `No Change` means that this variable will be not initialized before calling the function under test.
- Their expected expression (i.e. the expected value after calling the function under test). `Same as Init` mean that the expected value is the same that the initialized value.

- Their obtained expression (i.e. the actual value after calling the function under test). This column is empty for now. It will be automatically filled after an execution.

Be default, the wizard generates a test case with all the parameters to 0 (or `null` in case of pointers). Let's modify this test case for having the following division 50/7 that should give 7 as result.

⇒ Click on the cell `initial expression` of the variable `a` and enter the value `50`. Press `enter` to validate this value, or click on the icon .

⇒ Click on the cell `initial expression` of the variable `b`, enter the value `7` and validate it. Do not modify the expected result of `retValue` for now.

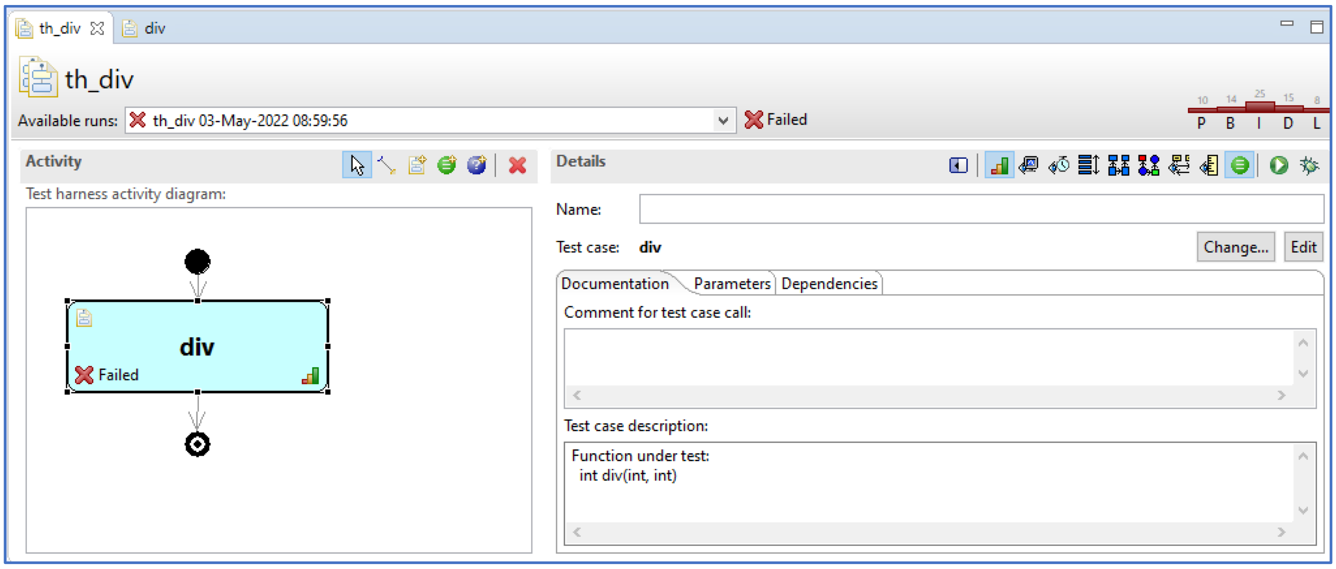
3.4 Execute a test

⇒ Go back in the test harness by clicking on the tab `th_div` in the editor panel.

⇒ Click on the icon on the top left of the details panel. The console view should display build log… This should take less than 1mn.

3.5 View a test result

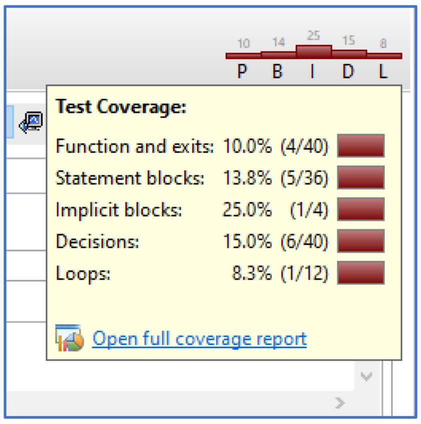
At the end of the execution, the test harness editor is updated to display the result as following:



Function under test:
  int div(int, int)

- The filed **`Available runs`** is updated with the last execution result and its status is displayed (**`Failed`** in this case).
- The activity diagram is updated with the status of each test case (**`Failed`** in this case).
- A coverage summary is displayed on the top right of the panel.

⇒ You can hover over this coverage summary to display a more detailed information



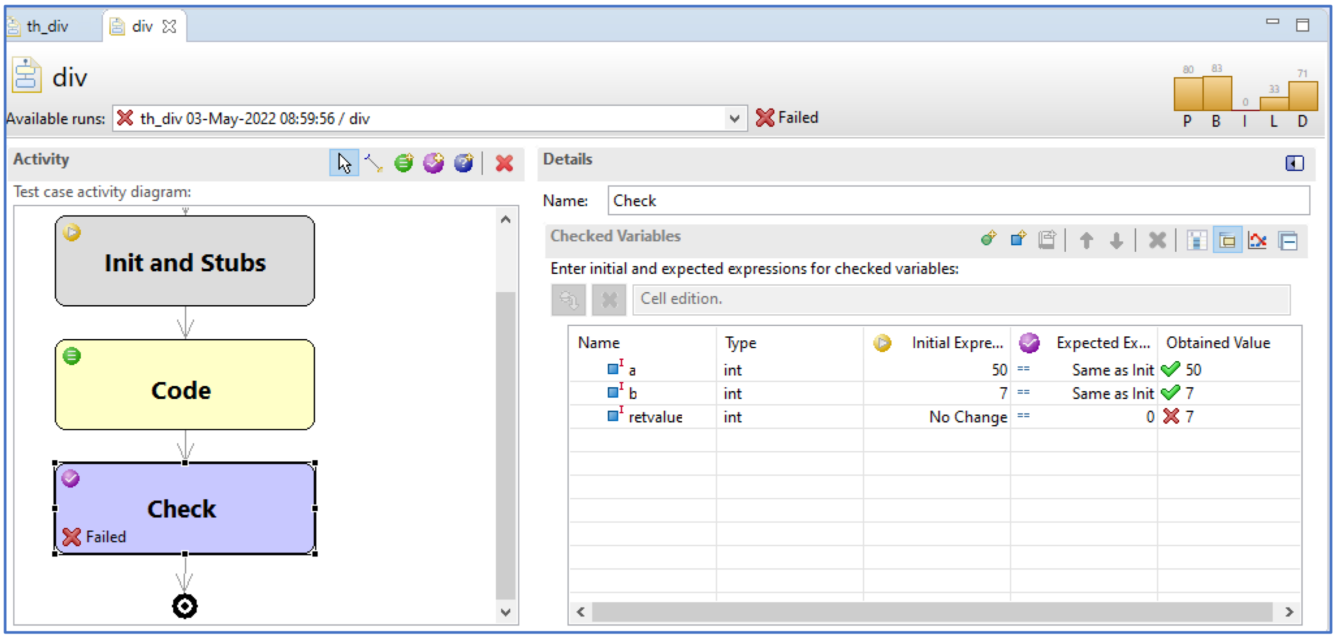
Test Coverage:

Function and exits:	10.0% (4/40)	🟥
Statement blocks:	13.8% (5/36)	🟥
Implicit blocks:	25.0% (1/4)	🟥
Decisions:	15.0% (6/40)	🟥
Loops:	8.3% (1/12)	🟥

Open full coverage report

⇒ Then click on the link **Open full coverage report** to open the coverage viewer with the information relative to the file under test only.

⇒ Double-click on the test case **div** inside the activity diagram. You will go back in the test case editor which display additional information relative to the last runs:



div

Available runs: th_div 03-May-2022 08:59:56 / div   Failed

Name: Check

Enter initial and expected expressions for checked variables:

Name	Type	Initial Expre...		Expected Ex...	Obtained Value
a	int	50	==	Same as Init	✓ 50
b	int	7	==	Same as Init	✓ 7
retvalue	int	No Change	==	0	✗ 7

- The filed **Available runs** is updated with the last execution result and its status is displayed (**Failed** in this case).
- The activity diagram is updated with the status of the **Check** box (**Failed** in this case).
- A coverage summary is displayed on the top right of the panel.
- The column **Obtained value** is updated with the true values read during the execution and their status.

3.6 Fix the test case

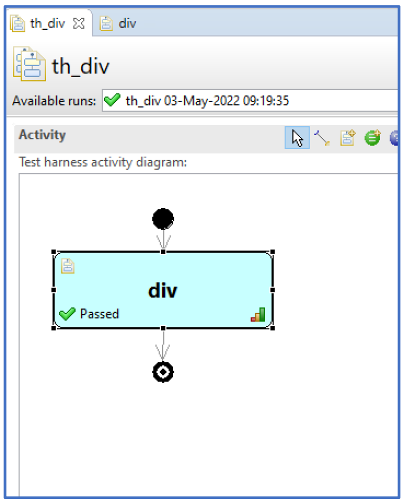
We can see that the value of `retvalue` is wrong.

⇒ Fix the expected value of `retvalue` to `7`

⇒ Save the test case `div`

⇒ Go back in the test harness and re-run the test

Now, it is passed:


th_div
div
th_div
Available runs: th_div 03-May-2022 09:19:35
Activity
Test harness activity diagram:
div
Passed

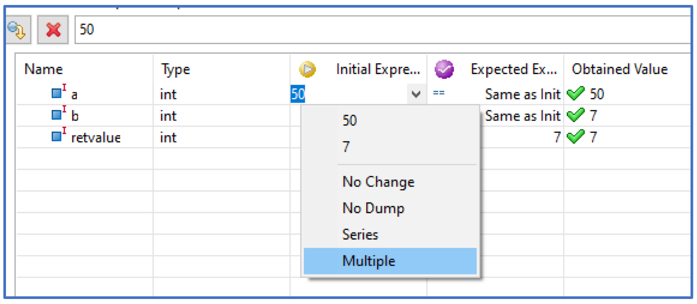
3.7 Update the test case with multiple input values

OneTest Embedded allows you to define not only a single value but multiple values for an input variable, and also for the expected values of an output parameter. For multiple input values, you can:

1. Give a list of values from a min value to a max value with a defined step,
2. Give a list of values as a list,
3. Give a list of values that come from a datapool.

We will illustrate the second case here, with several values on parameter `a` that should modify output value of `retvalue`.

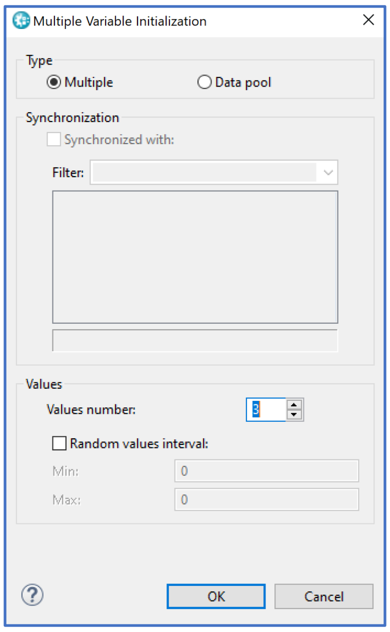
⇒ Click on the input Expression of the variable `a` and select menu **`Multiple`**.



50

Name	Type	Initial Expre...	Expected Ex...	Obtained Value
a	int	50	== Same as Init	✓ 50
b	int		Same as Init	✓ 7
retvalue	int		7	✓ 7

⇒ In the dialog box, select **`Multiple`** option, and the **`Values number`** to `3`.



Multiple Variable Initialization

Type: [✓ Multiple] Data pool

Synchronization

Synchronized with:

Filter:

Values

Values number: 3

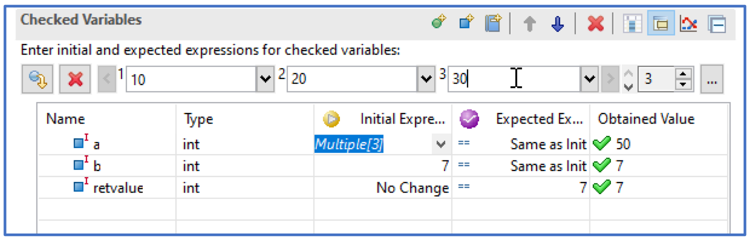
Random values interval:

Min: 0

Max: 0

OK Cancel

⟹ Then click on **OK**. A new bar appears on top of the table. Enter the values `10`, `20` and `30` in the 3 available fields.



Checked Variables

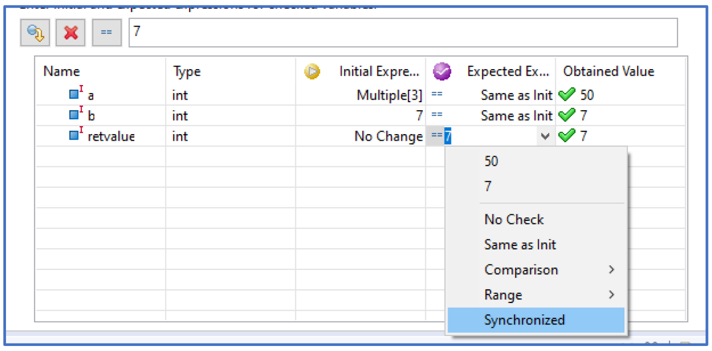
Enter initial and expected expressions for checked variables:

1 10 2 20 3 30 3

Name	Type	Initial Expre...	Expected Ex...	Obtained Value
a	int	Multiple[3]	Same as Init	✓ 50
b	int	7	Same as Init	✓ 7
retvalue	int	No Change	7	✓ 7

⟹ Press `enter` to validate these values, or click on the icon .

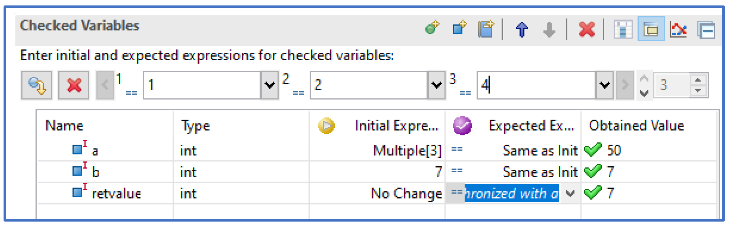
⟹ Click on the Expected Expression of the variable `retvalue` and select menu **Synchronized**.



7

Name	Type	Initial Expre...	Expected Ex...	Obtained Value
a	int	Multiple[3]	Same as Init	✓ 50
b	int	7	Same as Init	✓ 7
retvalue	int	No Change	7	✓ 7

⟹ A new bar appears on top of the table. Enter the values `1`, `2` and `4` in the 3 available fields.



Checked Variables

Enter initial and expected expressions for checked variables:

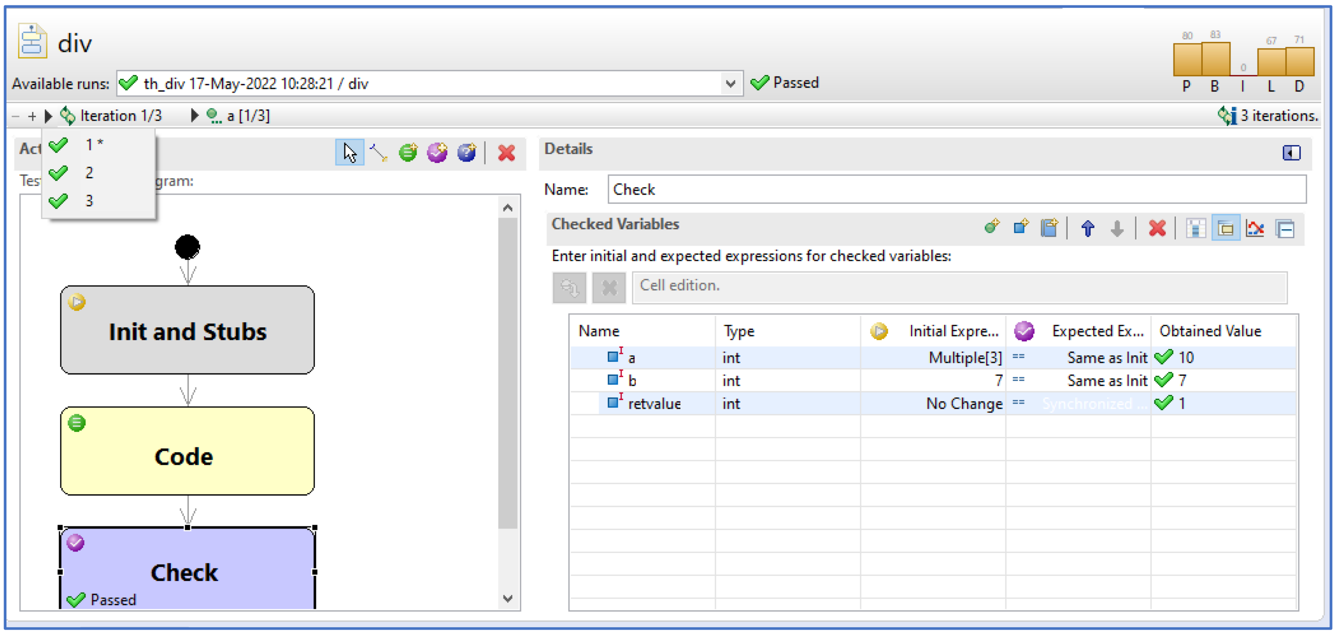
1 1 2 2 3 4 3

Name	Type	Initial Expre...	Expected Ex...	Obtained Value
a	int	Multiple[3]	Same as Init	✓ 50
b	int	7	Same as Init	✓ 7
retvalue	int	No Change	hronized with d	✓ 7

⟹ Press `enter` to validate these values, or click on the icon .

⇒ Execute this test case Click by going back on the test harness and clicking on the icon on the top left of the details panel.

⇒ At the end of the execution, the test case is updated with the obtained values. You can navigate in the 3 iterations using the breadcrumb bar on the top of the test case:



div

Available runs: th_div 17-May-2022 10:28:21 / div ✓ Passed

Iteration 1/3 ▸ a [1/3] — 3 iterations.

Details

Name: Check

Checked Variables

Enter initial and expected expressions for checked variables:

Cell edition.

Name	Type	Initial Expre...		Expected Ex...	Obtained Value
a	int	Multiple[3]	==	Same as Init	✓ 10
b	int	7	==	Same as Init	✓ 7
retvalue	int	No Change	==	Synchronized ...	✓ 1