05

IMPORTING TEST CASES

Objectives

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

- Export a test case template from testIDEA suitable for use in Excel
- Import test cases created in Excel into testIDEA

testIDEA

iSYSTEM

05

Contents

IMPORTING TEST CASES

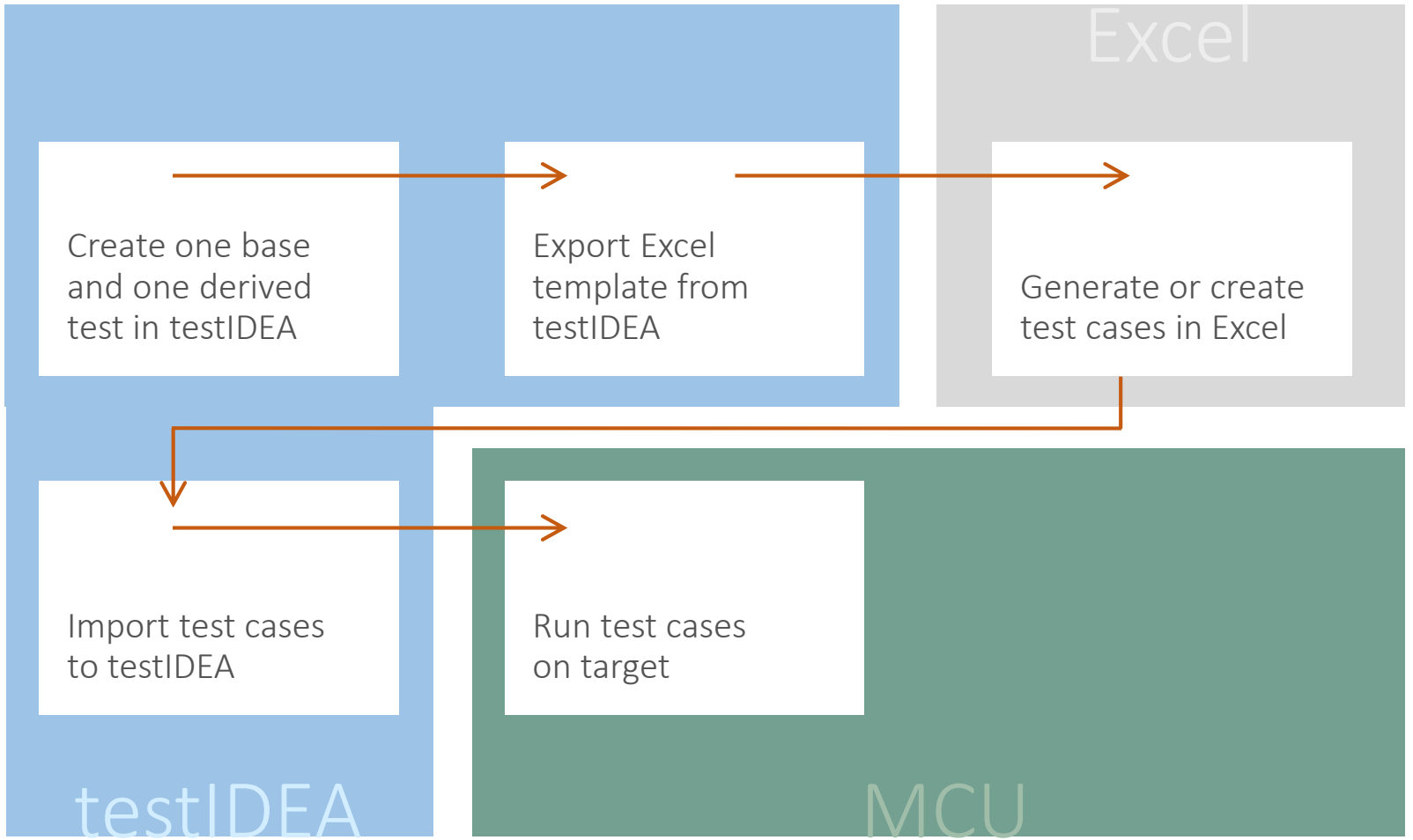
testIDEA

1 PROCESS FOR LARGE NUMBERS OF TEST CASES

In this unit we will test the same function that we tested in Unit 04 using a process that can handle a large numbers of test cases.

It starts by constructing an Excel (or similar) template for the test vectors via testIDEA's export option. From there, it is possible to use the various inbuilt table calculation options of Excel to quickly create test vectors.

Once the test vectors are completed, the Excel data is re-imported into testIDEA, allowing us to execute the tests on the microcontroller target.



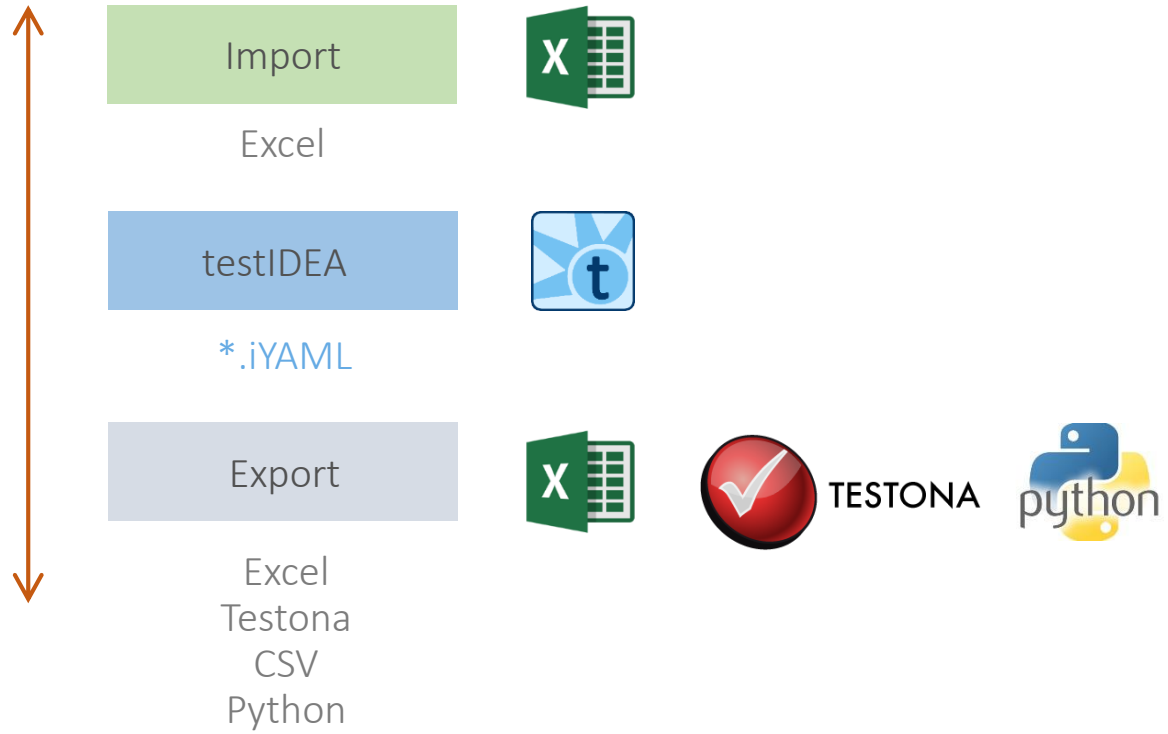
2 FILES OF THE TESTING ENVIRONMENT

testIDEA environment - files

The test cases can be imported directly from an Excel file. Excel makes it simple to create test vectors or import existing vectors into a testIDEA template created for Excel.

Once the test vectors have been imported back into testIDEA, they are saved in YAML format in an *.iYAML file.

The tests can be executed from within testIDEA. Integration and automation is also possible by, for example, exporting the tests as a Python script. This can then be used as part of an automated test environment with, e.g. Jenkins.



The export and import functionality of testIDEA require the testIDEA Pro license.

PRO

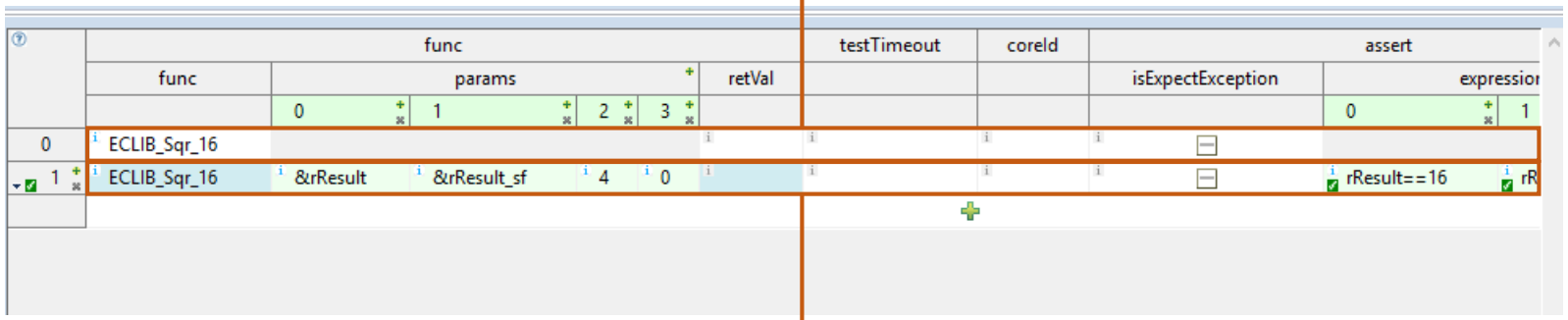
3 GENERATE EXCEL-TEMPLATE - INITIAL SITUATION AND CONCEPT

As a starting point for the creation of our test vectors in Excel, we will take the test vectors from the previous unit and display them in the table view.

This is shown opposite.

testIDEA

Base test



Derived test

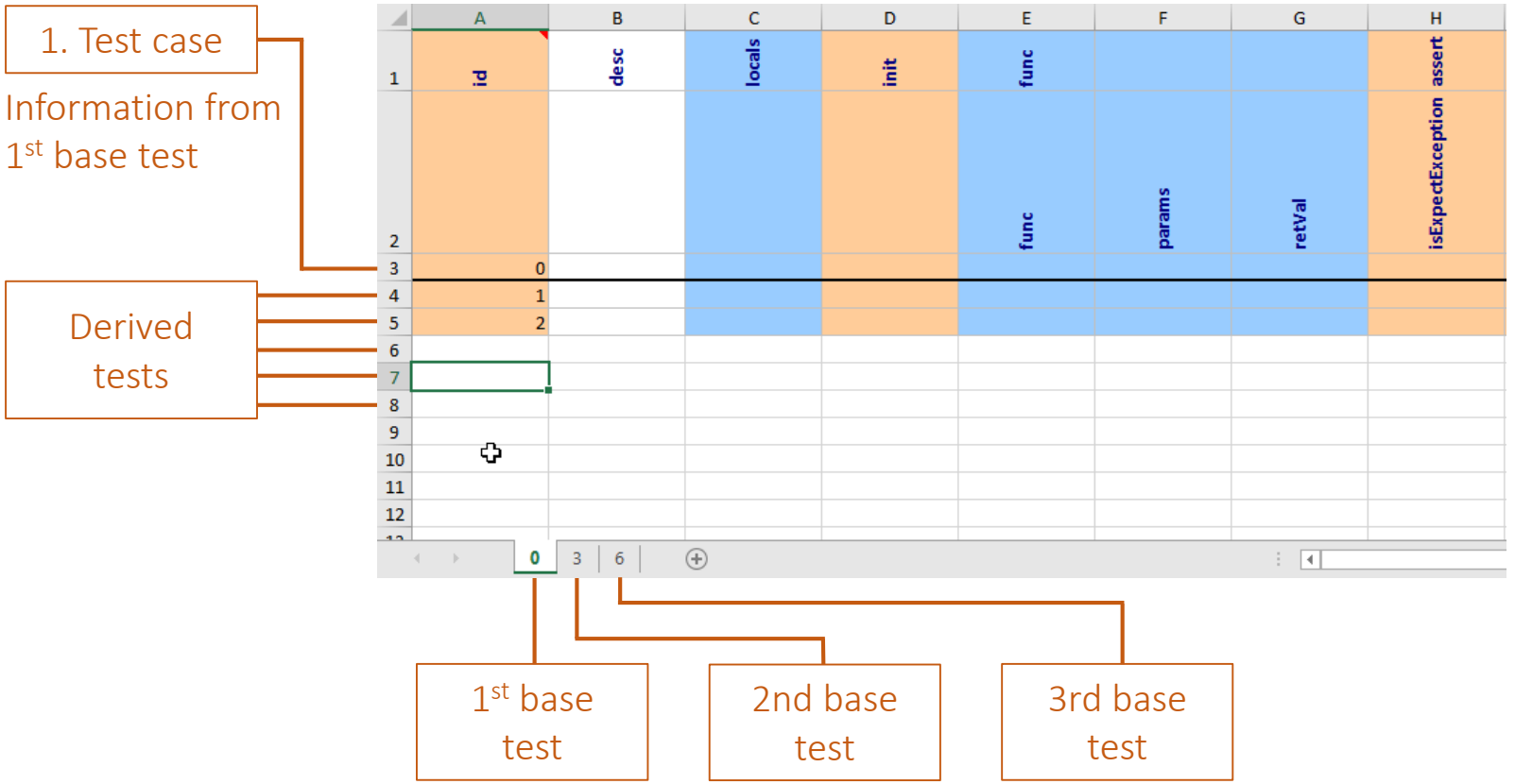
3 GENERATE EXCEL-TEMPLATE - INITIAL SITUATION AND CONCEPT

Test representation in Excel:

Every Excel sheet represents one base test.

The contents of a single Excel-sheet display the individual tests. When tests are derived from a base test, the approach taken up until now with test creation, the thick horizontal black lines divide the derived tests from their base test.

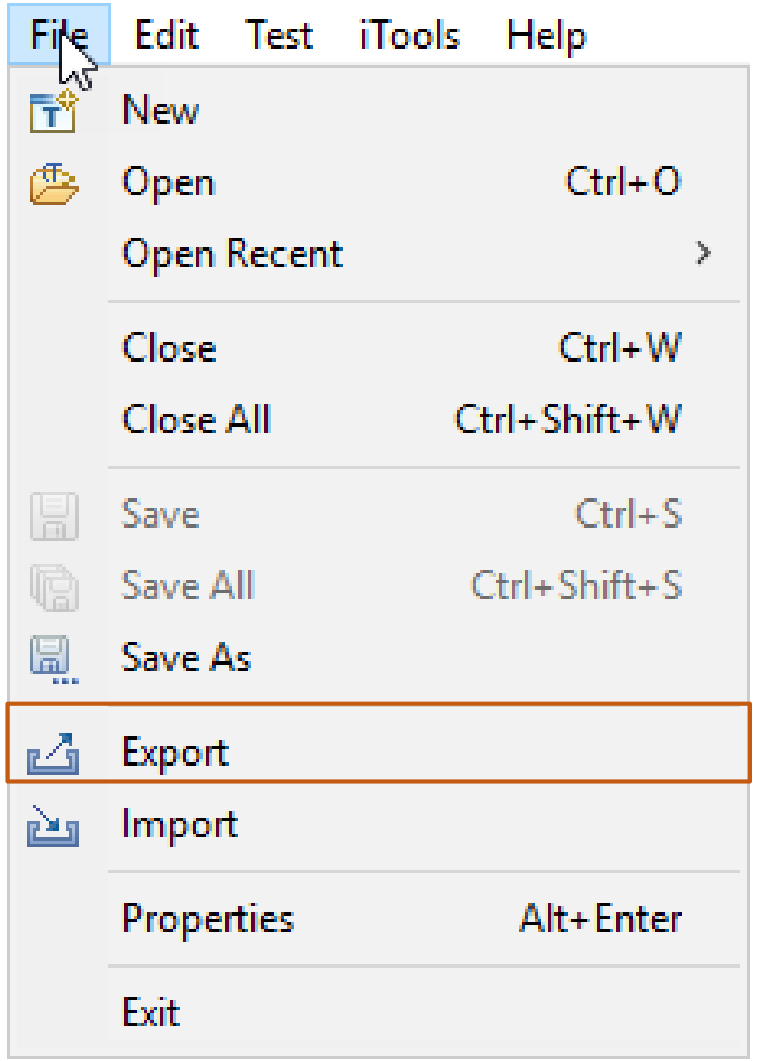
Excel



3 GENERATE EXCEL-TEMPLATE

As we want to create an Excel-template, we start by exporting our two, hand-generated, test vectors to an Excel file.

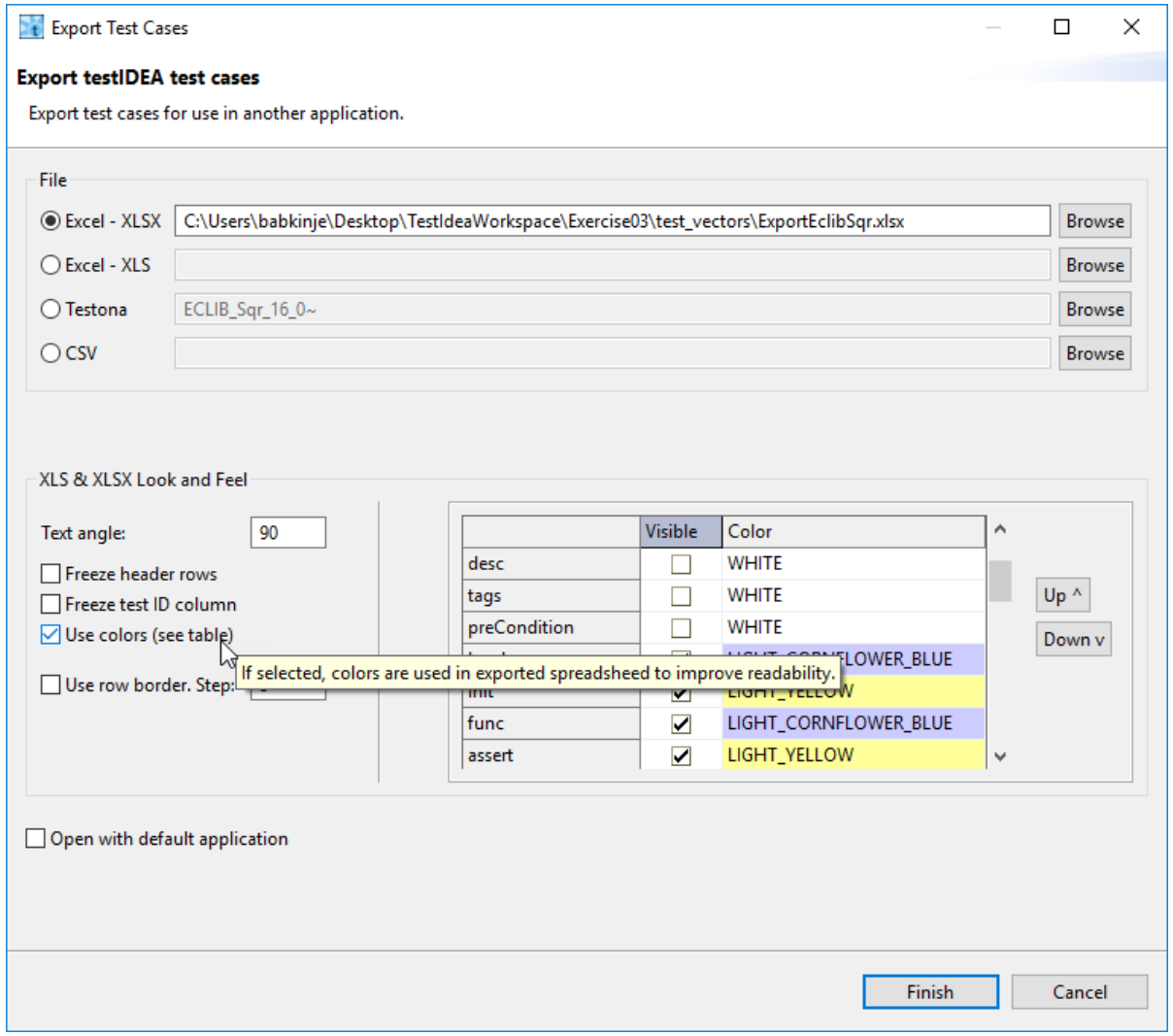
Use the “File“ menu to select the “Export“ option.



3 GENERATE EXCEL-TEMPLATE

In the "*Export Test Cases*" dialogue we can select the file format and the path as well as some of the look-and-feel options.

The use of colors can be helpful if you are planning to create a lot of test vectors, making editing of the entries in Excel easier.



3 GENERATE EXCEL-TEMPLATE

Here we see the exported test vectors within Excel.

We see that line 4 represents our base test with the function name and the locals (declared variables) that are to be inherited.

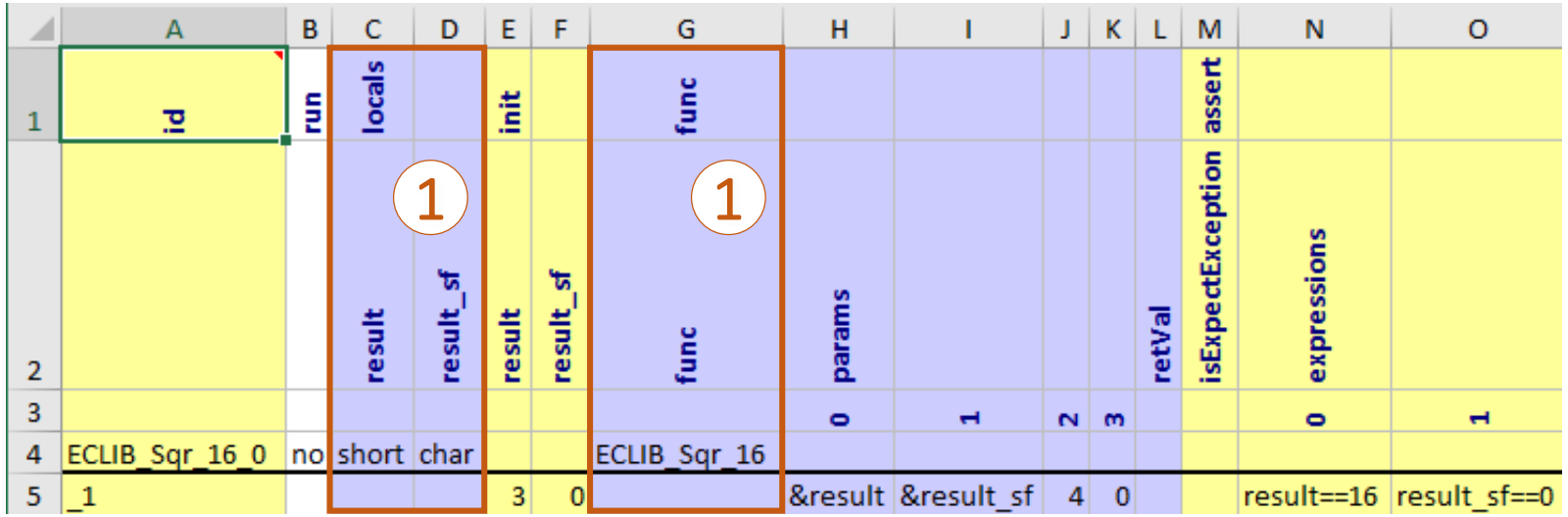
Line 5 contains all the elements of our first derived test: the initialized variables, the parameters and the expected value expressions.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	id	run	locals		init		func						assert		
2			result	result_sf	result	result_sf	func	0 params	1	2	3	retVal	isExpectException	0 expressions	1
3															
4	ECLIB_Sqr_16_0	no	short	char			ECLIB_Sqr_16								
5	_1				3	0		&result	&result_sf	4	0			result==16	result_sf==0

To get this sorting of the test vector (base and derived in a single Excel sheet) you must ***mark only the base test for export.***
If the derived test is also marked, further Excel sheets will be generated for each derived test in addition to the view shown above.

3 GENERATE EXCEL-TEMPLATE - HANDLING

① The Excel cells indicated do not have to be filled as they are inherited.



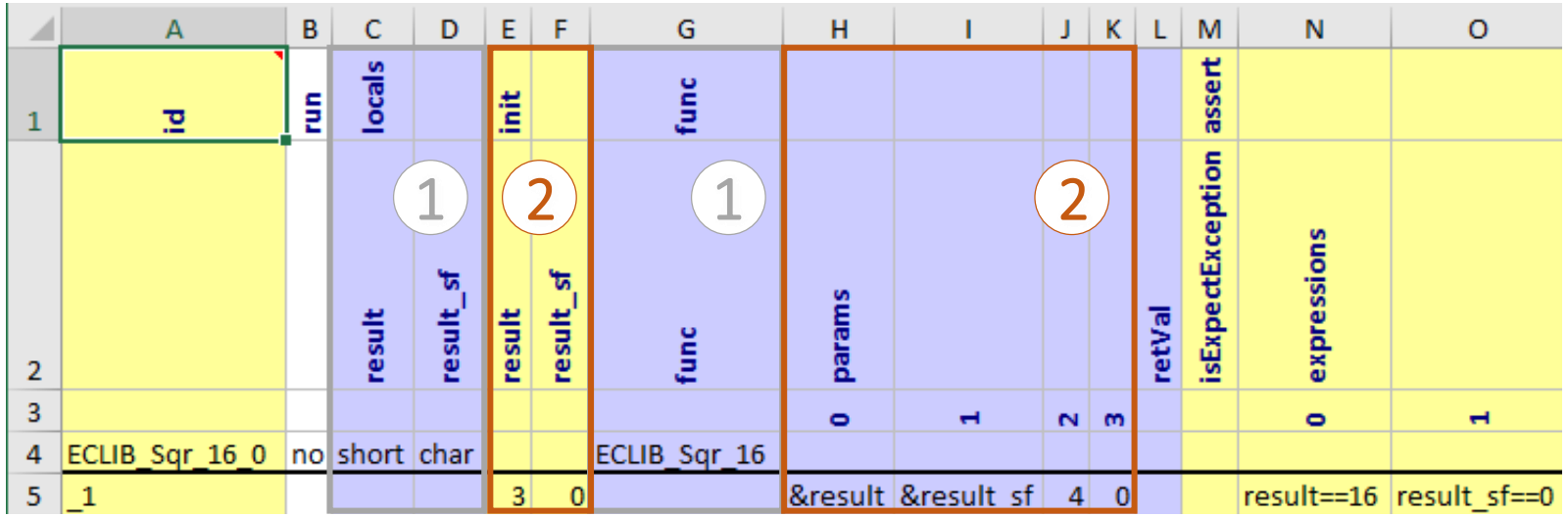
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	id	run	locals		init		func						assert		
2			result	result_sf	result	result_sf	func	0 params	1	2	3	retVal	isExpectException	0 expressions	1
3								0	1	2	3			0	1
4	ECLIB_Sqr_16_0	no	short	char			ECLIB_Sqr_16								
5	_1				3	0		&result	&result_sf	4	0			result==16	result_sf==0

3 GENERATE EXCEL-TEMPLATE - HANDLING

1 The Excel cells indicated do not have to be filled as they are inherited.

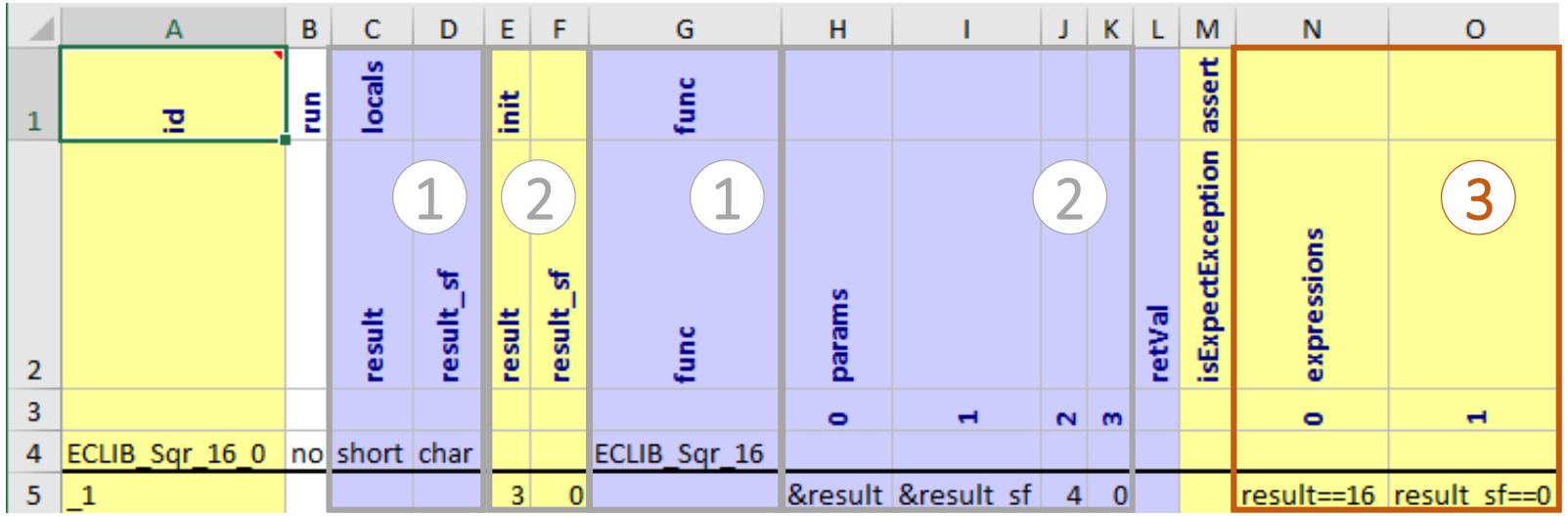
2 The cells containing the test parameters and their initial values have to be filled each time.

This is necessary as a partial *params* inheritance is not supported. In this case, although the *&result* and *&result_sf* are always the same, they have to be entered each time in columns H and I.



3 GENERATE EXCEL-TEMPLATE - HANDLING

3 Finally, the *expressions* (the expected test results) must be filled in.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	id	run	locals		init		func						assert		
2			result	result_sf	result	result_sf	func	0 params	1	2	3	retVal	isExpectException	0 expressions	1
3															
4	ECLIB_Sqr_16_0	no	short	char			ECLIB_Sqr_16								
5	_1				3	0		&result	&result_sf	4	0			result==16	result_sf==0

Now you have a fully functional Excel template.

4 FILL EXCEL-TABLE WITH TEST CASES - INITIAL SITUATION

In order to guarantee the desired test coverage, the following unit test cases have been generated by the software developers.

Included are a Test ID, a test description, input vectors for each unit test and expected results.

These test cases will need to be imported into testIDEA by inserting them into the Excel template we just created.

	A	B	C	D	E	F	G
1			Result of Test	Input Vectors for test			
2	Test ID	Test Description	Expected Result (myResult)	Result Shift Factor (myResult_sf)	Input Value	Input Value Shift Factor	Expected floating point result (not used for testing)
3							
4	ECLIB_Sqr_16.0001	par1 and par2 are zero, all sf are zero	0	0	0	0	0
5	ECLIB_Sqr_16.0002	par1 and par2 are zero, all sf are zero	0	0	0	0	0
6	ECLIB_Sqr_16.0003	par1 and par2 are zero, all sf are zero	0	0	0	0	0
7	ECLIB_Sqr_16.0004	21	10	2	13	3	10,5625
8	ECLIB_Sqr_16.0005	par is zero	0	0	0	0	0,00
9	ECLIB_Sqr_16.0006	par is 5	25	0	5	0	25,00
10	ECLIB_Sqr_16.0007	par is near max	ECLIB_S16_POS_INF	0	32766	0	1.073.610.756,00
11	ECLIB_Sqr_16.0008	par is max	ECLIB_S16_POS_INF	0	32767	0	32.767,00
12	ECLIB_Sqr_16.0009	par is near min	ECLIB_S16_POS_INF	0	-32766	0	1.073.610.756,00
13	ECLIB_Sqr_16.0010	par is min	ECLIB_S16_POS_INF	0	-32767	0	32.767,00
14	ECLIB_Sqr_16.0011	par is NAN	ECLIB_S16_NAN	0	-32768	0	-32.768,00
15	ECLIB_Sqr_16.0012	par is zero	0	42	0	41	0,00
16	ECLIB_Sqr_16.0013	par is 5	0	42	5	41	0,00
17	ECLIB_Sqr_16.0014	par is near max	0	42	32766	41	0,00
18	ECLIB_Sqr_16.0015	par is max	ECLIB_S16_POS_INF	42	32767	41	32.767,00
19	ECLIB_Sqr_16.0016	par is near min	0	42	-32766	41	0,00
20	ECLIB_Sqr_16.0017	par is min	ECLIB_S16_POS_INF	42	-32767	41	32.767,00
21	ECLIB_Sqr_16.0018	par is NAN	ECLIB_S16_NAN	42	-32768	41	-32.768,00
22	ECLIB_Sqr_16.0019	par is zero	0	41	0	43	0,00
23	ECLIB_Sqr_16.0020	par is 5	0	41	5	43	0,00
24	ECLIB_Sqr_16.0021	par is near max	0	41	32766	43	0,00
25	ECLIB_Sqr_16.0022	par is max	ECLIB_S16_POS_INF	41	32767	43	32.767,00
26	ECLIB_Sqr_16.0023	par is near min	0	41	-32766	43	0,00
27	ECLIB_Sqr_16.0024	par is min	ECLIB_S16_POS_INF	41	-32767	43	32.767,00
28	ECLIB_Sqr_16.0025	par is NAN	ECLIB_S16_NAN	41	-32768	43	-32.768,00
29	ECLIB_Sqr_16.0026	par is zero	0	43	0	42	0,00
30	ECLIB_Sqr_16.0027	par is 5	0	43	5	42	0,00
31	ECLIB_Sqr_16.0028	par is near max	0	43	32766	42	0,00

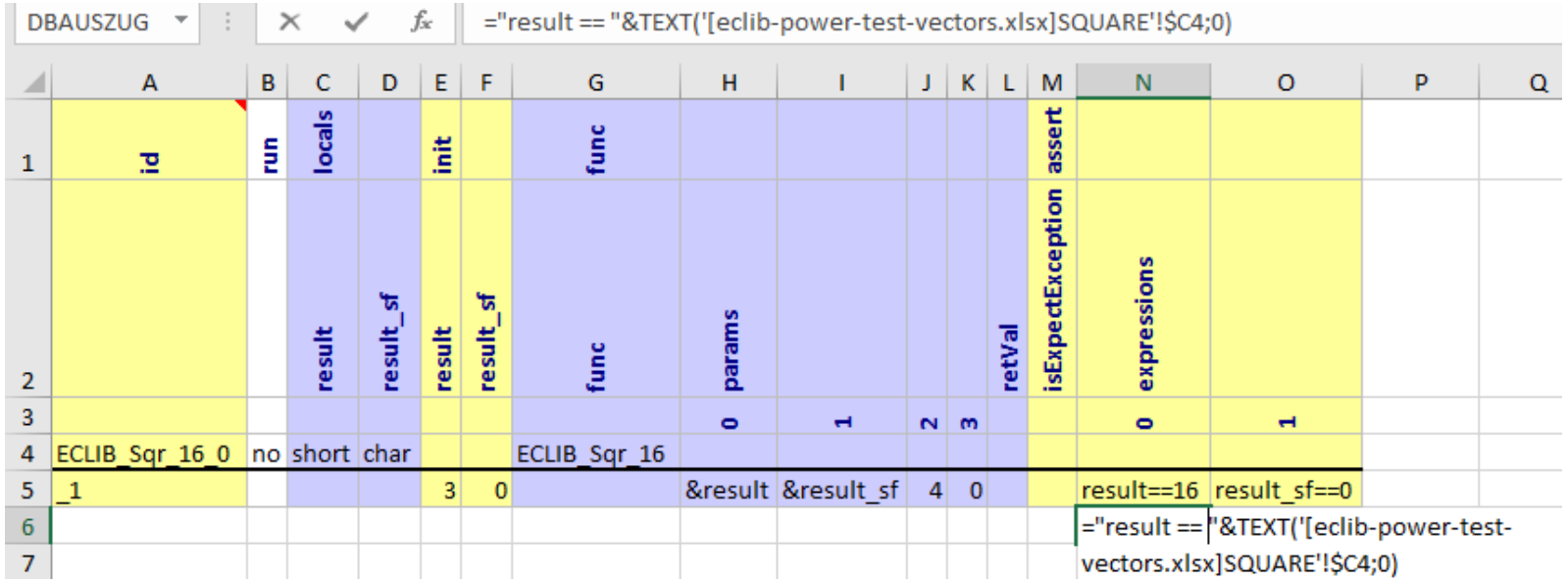
SQUARE

4 FILL EXCEL-TABLE WITH TEST CASES

The testIDEA Excel template can now be extended by using a simple *TEXT()* import from the Excel spreadsheet seen on the previous page.

The Excel *TEXT()* function simply copies the content of the cell referenced (in this case, cells in a different Excel file) into the cell as text. This ensures that there are no issues with numerical representation that can sometimes occur.

Here, the “result==“ string is combined with the desired result from row 4 of the test case Excel spreadsheet.



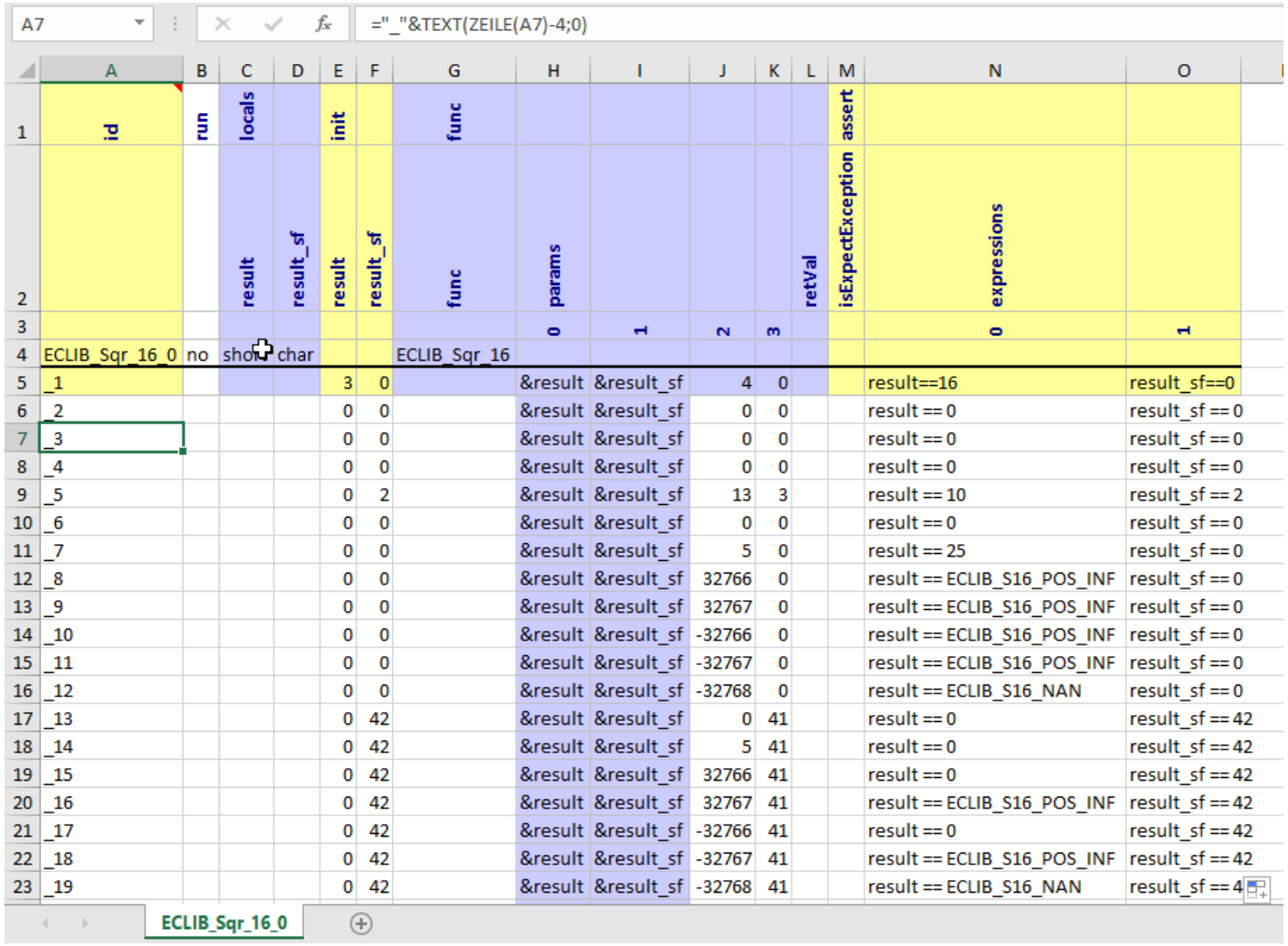
Example - Reference to the calculated expected results:

= “result==“&TEXT(’[eclib–power–test–vectors.xlsx]SQUARE’!$C4;0)

4 FILL EXCEL-TABLE WITH TEST CASES

After adding the references and extrapolating the entries, the file will contain the 100 or more test cases.

This makes it simple to create large numbers of test vectors for testIDEA using commonly available tools and existing test cases.



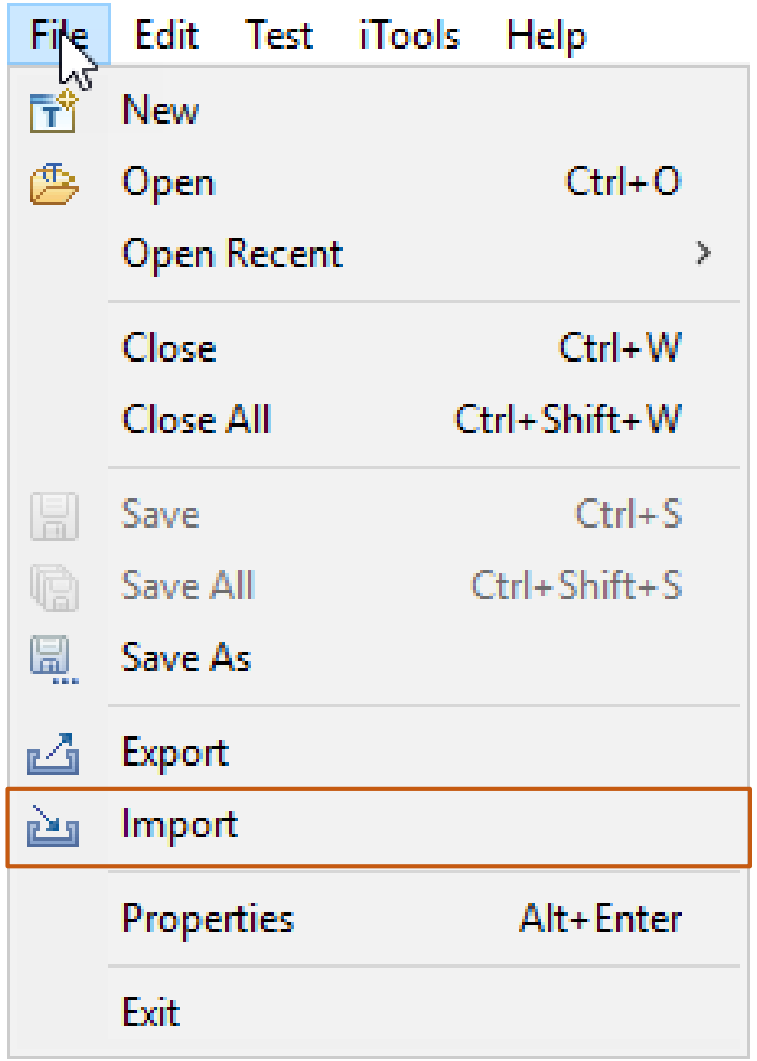
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	id	run	locals		init		func						assert		
2			result	result_sf	result	result_sf	func	params				retVal	isExpectException	expressions	
3								0	1	2	3			0	1
4	ECLIB_Sqr_16_0	no	short	char			ECLIB_Sqr_16								
5	_1				3	0		&result	&result_sf	4	0			result==16	result_sf==0
6	_2				0	0		&result	&result_sf	0	0			result == 0	result_sf == 0
7	_3				0	0		&result	&result_sf	0	0			result == 0	result_sf == 0
8	_4				0	0		&result	&result_sf	0	0			result == 0	result_sf == 0
9	_5				0	2		&result	&result_sf	13	3			result == 10	result_sf == 2
10	_6				0	0		&result	&result_sf	0	0			result == 0	result_sf == 0
11	_7				0	0		&result	&result_sf	5	0			result == 25	result_sf == 0
12	_8				0	0		&result	&result_sf	32766	0			result == ECLIB_S16_POS_INF	result_sf == 0
13	_9				0	0		&result	&result_sf	32767	0			result == ECLIB_S16_POS_INF	result_sf == 0
14	_10				0	0		&result	&result_sf	-32766	0			result == ECLIB_S16_POS_INF	result_sf == 0
15	_11				0	0		&result	&result_sf	-32767	0			result == ECLIB_S16_POS_INF	result_sf == 0
16	_12				0	0		&result	&result_sf	-32768	0			result == ECLIB_S16_NAN	result_sf == 0
17	_13				0	42		&result	&result_sf	0	41			result == 0	result_sf == 42
18	_14				0	42		&result	&result_sf	5	41			result == 0	result_sf == 42
19	_15				0	42		&result	&result_sf	32766	41			result == 0	result_sf == 42
20	_16				0	42		&result	&result_sf	32767	41			result == ECLIB_S16_POS_INF	result_sf == 42
21	_17				0	42		&result	&result_sf	-32766	41			result == 0	result_sf == 42
22	_18				0	42		&result	&result_sf	-32767	41			result == ECLIB_S16_POS_INF	result_sf == 42
23	_19				0	42		&result	&result_sf	-32768	41			result == ECLIB_S16_NAN	result_sf == 4

ECLIB_Sqr_16_0

5 IMPORT EXCEL FILE

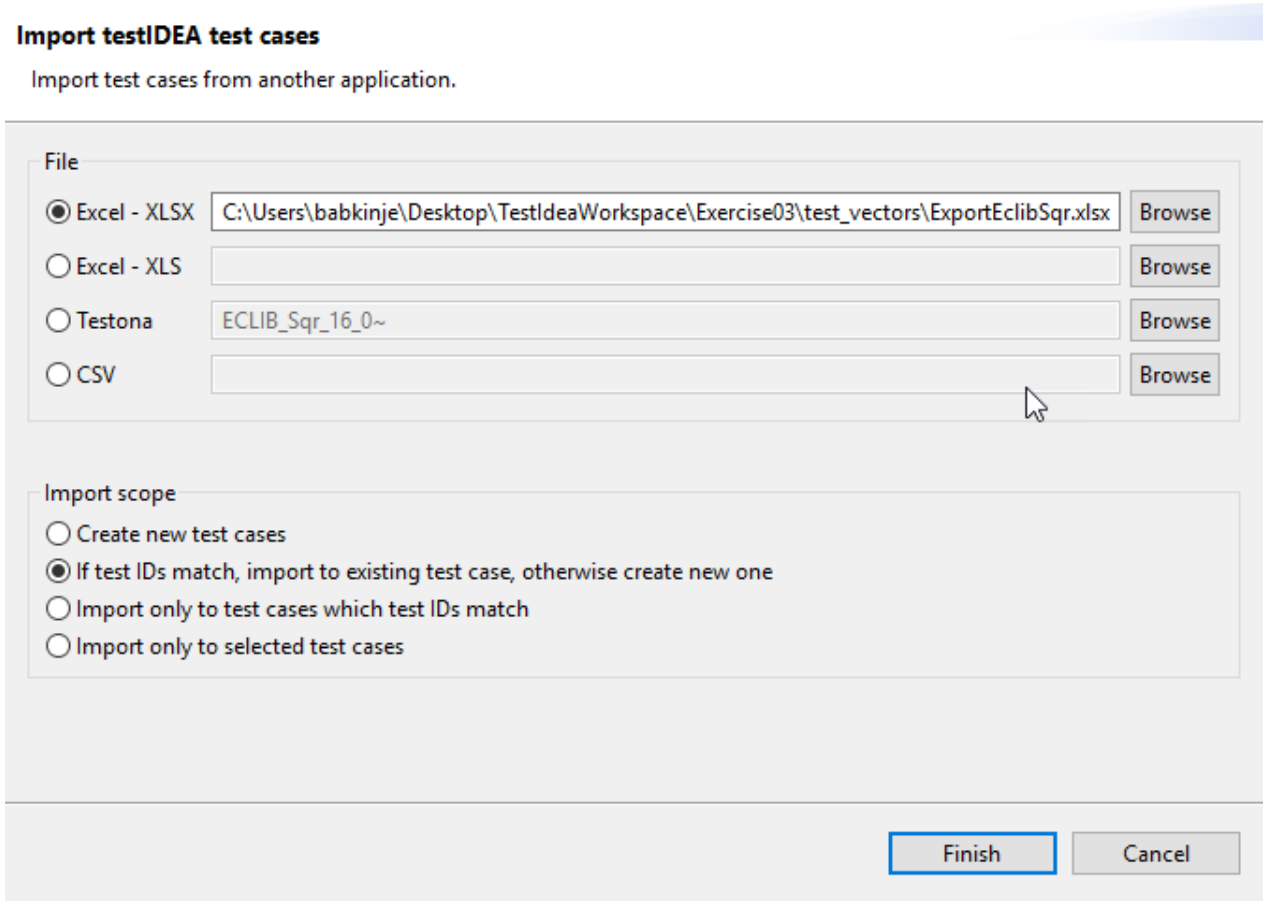
Once the Excel spreadsheet test vectors have been generated we can import them back into testIDEA.

Simply navigate to the “File“ menu and select the “Import“ option.



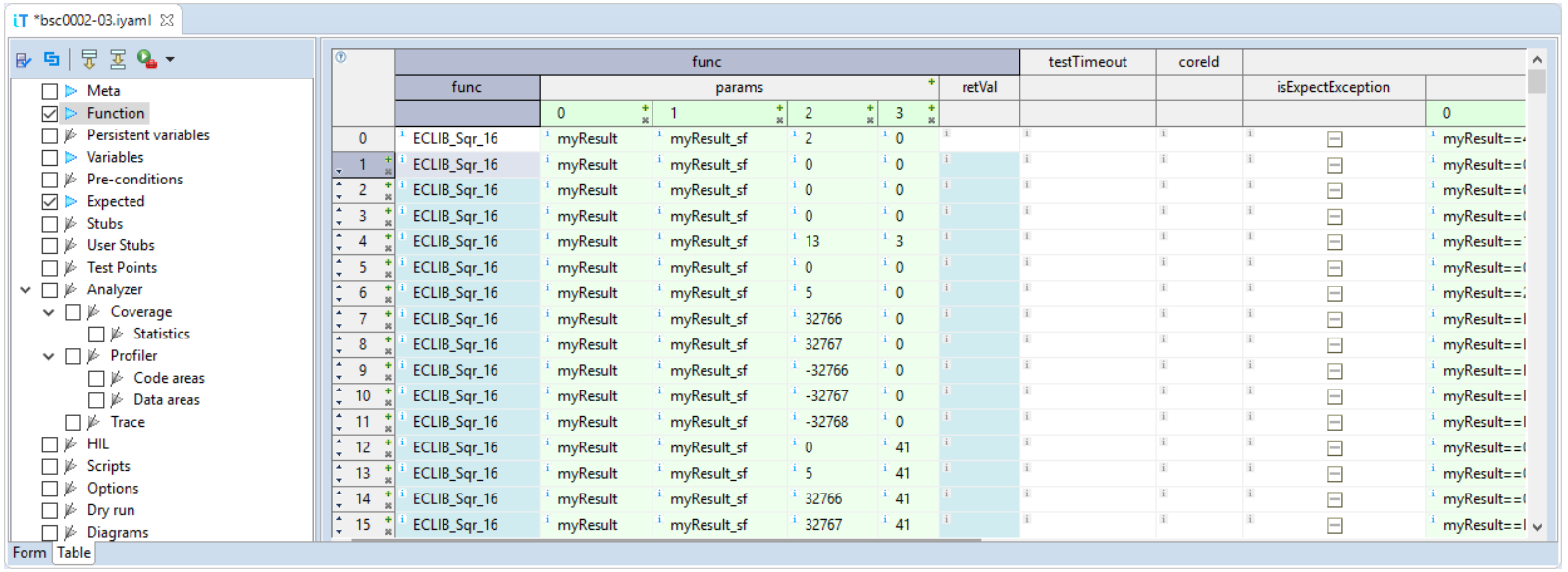
5 IMPORT EXCEL FILE

The *"Import testIDEA test cases"* dialog opens, allowing us to define the document for import and required import scope.



5 IMPORT EXCEL FILE

Once completed, the testIDEA view of the same test cases looks as shown opposite.

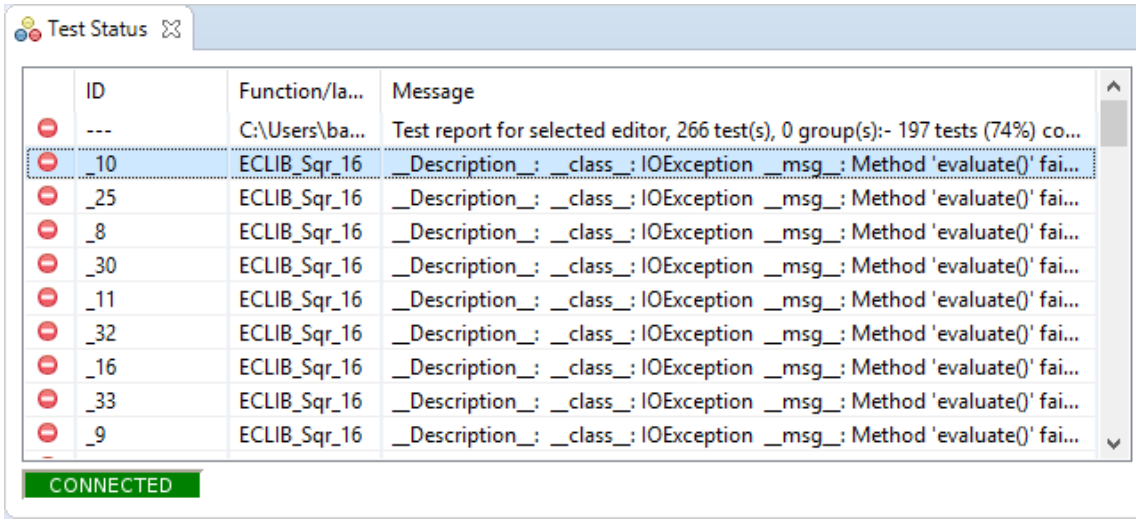


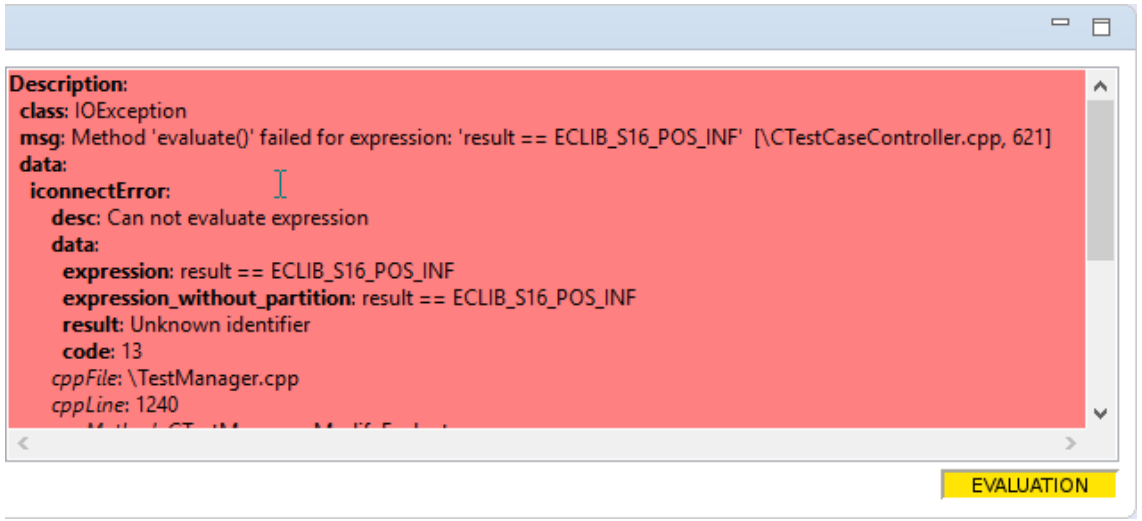
5 IMPORT EXCEL FILE

Importing and executing these tests into testIDEA at this point, for this particular code, results in about half the tests passing and the rest failing.

This is because the test rely upon the *#define* of NAN, POS_INF and NEG_INF.

#define are preprocessor values and do not appear as symbols in the ELF file - therefore we need to somehow substitute them.





6 ADD PERSISTENT VARIABLE

A simple approach to handling the problem of missing *#define* definitions is to define them as *Persistent variables* and assign them the required values for the duration of the testing.

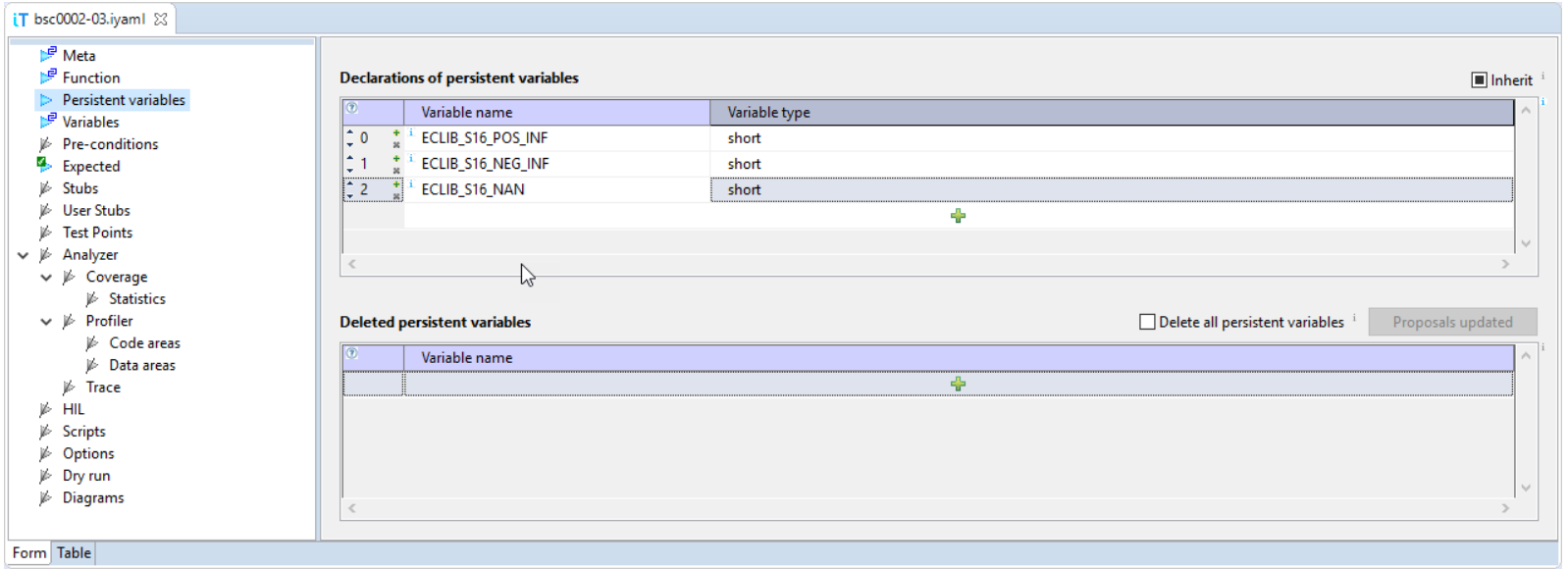
Persistent variables are analogous to global variables in a C or C++ application in that they are available to all functions and methods. Additionally, by using them, we can continue to develop legible test cases using understandable symbols rather than replacing these symbols with their numeric representation.

Wie geht man grundsätzlich mit persistent variablen um

6 ADD PERSISTENT VARIABLE

The three *#define* of our EC-LIB© example must first be declared as persistent variables in our **first derived test**. Therefore we navigate to the “Persistent variables“ form and add the new variables to replace the *#define*.

*Do not declare persistent variables in the base test as they will be added to **all** tests, resulting in testIDEA trying to initialize them for each test, which will fail as there can only be one instance of each symbol.*
It is also recommended to copy the exact expression directly from the source code so as to avoid typing errors.

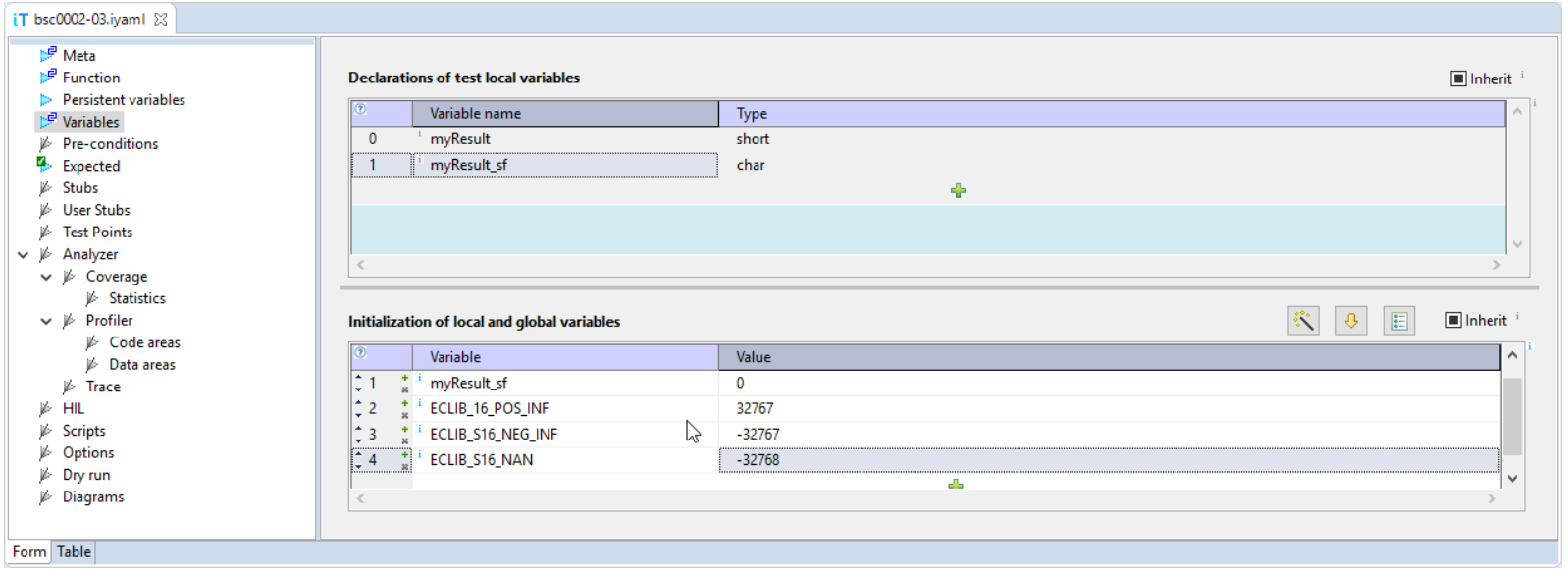


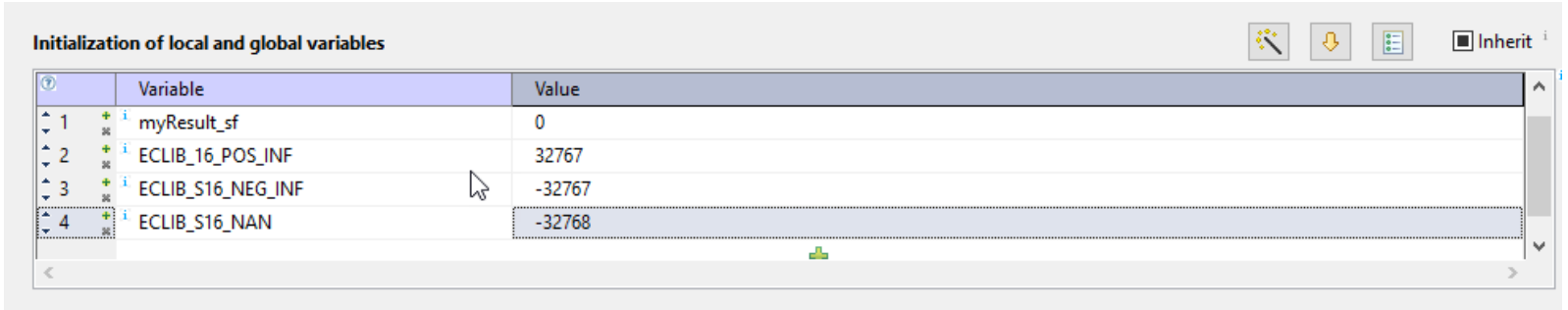
6 ADD PERSISTENT VARIABLE

We also have to initialize our persistent variables.

Switch to the “Variables“ form and add them with their accompanying values manually to the **initialization list** (not the declaration list). This is shown opposite.

As these are persistent, and since they will not be changed during the tests, this only needs to be undertaken for the first derived test.



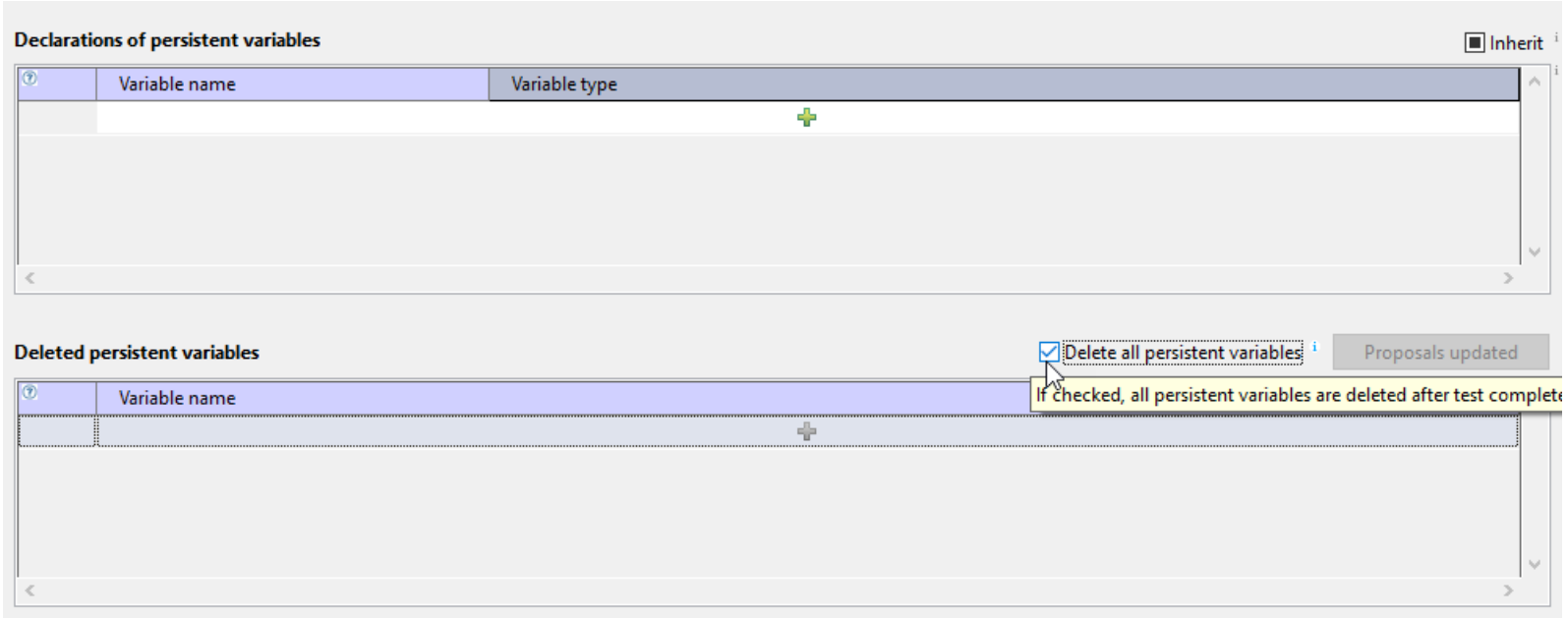


6 ADD PERSISTENT VARIABLE

In the last derived test it is recommend to delete the persistent variables created for the tests, since they have lifetime across all test cases being executed.

This can be performed in the “Persistent variables“ form by ticking the check box “Delete all persistent variables”.

Failing to do so may cause subsequent tests to fail if they too attempt to declare persistent variables with the same names.



SUMMARY

testIDEA

7 SUMMARY

- *#define*s can be declared and initialized as persistent variables. Due to their persistent nature, it is recommended to delete them in the final test of a set of tests.
- Prepare an Excel template by first creating a single base and derived test and exporting it in Excel format. This Excel template can then be expanded upon and re-imported into testIDEA.
- The structure of base and derived tests are transferred to the Excel template when the base test **only** is selected in testIDEA. If the derived test was also selected, a new Excel sheet will be created for each derived test.