Visit us at www.hitex.de, www.hitex.co.uk or www.hitex.com

hitex
EMBEDDED TOOLS & SOLUTIONS

Hitex Germany (Head Quarters)
Greschbachstr. 12 ☎ +049-721-9628-0
D-76229 Karlsruhe Fax +049-721-9628-149
E-mail: Sales@hitex.de

Hitex USA
2062 Business Center ☎ 800-45-HITEX
Drive, Suite 230 ☎ +1-949-863-0320
Irvine, CA 92612 Fax +1-949-863-0331
E-mail: Info@hitex.com

Hitex UK
Warwick University ☎ +44-24-7669-2066
Science Park Fax +44-24-7669-2131
GB Coventry CV47EZ E-mail: Info@hitex.co.uk

Hitex Asia
25 International ☎ +65-6566-7919
Business Park, #04-62A Fax +65-6563-7539
German Centre E-mail: Sales@hitexasia.com.sg
Singapore 609916

Embedding Software Quality

Tutorial

Pointers, Random Input,

… and Unknown Expected Output

TESSY is a tool to automate unit/module testing of Embedded Software

This application note describes how pointers are handled, how random test input can be generated, and how to proceed if the expected output is unknown or cannot determined easily.

We recommend reading for those who want to get more familiar with TESSY. However, a basic understanding of TESSY is prerequisite.

Architecture:	TESSY
Author:	Frank Buechner / Hitex GmbH
Revision:	11/2022 - 004

Preface

In order to keep you up-to-date with the latest developments on our products, we provide Application Notes containing additional topics, special hints, examples and detailed procedures etc.

For more information on our products, software and hardware revisions, as well as our update service, contact www.hitex.de, www.hitex.co.uk or www.hitex.com.

Contents

1 The Test Object Selectionsort()

1.1 The Source Code of Selectionsort()

As an example for a test object (i.e. a function in the sense of C) we take the function selectionsort() (see Fig. 1).

```
/* Selection sort
 * Input: array: Pointer to an array of integers.
 * Before the call, this array holds the integers to sort.
 * After the call, this array holds the integers in sorted order.
 * Input: size: Number of integers in the array.
 * Return value: Number of swaps performed during sorting.
 */

int selectionsort(int *array, int size)
{
    int i, j, lowindex;
    int count = 0;
    for (i = 0; i < size - 1; i++)
    {
        lowindex = i;
        for (j = size - 1; j > i; j--)
        {
            if (array[j] < array[lowindex])
            {
                lowindex = j;
            }
        }
        if (i != lowindex)
        {
            int tmp = array[i];
            array[i] = array[lowindex];
            array[lowindex] = tmp;
            count++;
        }
    }
    return count;
}
```

Fig. 1 Source code of the test object selectionsort()

The test object selectionsort() gets a pointer to an array of integers as first parameter. These integers are to be sorted. After the call to selectionsort(), the array holds the integers in sorted order.

The second parameter "size" of selectionsort() indicates the number of integers in the array to sort. selectionsort() can expect that at least "size" elements are present in the array.

The return value of selectionsort() indicates the number of swaps that were necessary to sort the array.

1.2 Testing Selectionsort()

The following was accomplished using TESSY V3.2.11. The compiler used is the GNU C compiler included in the TESSY installation. You do not need a cross compiler to reproduce the following example.

1.2.1 Preparations

1.2.1.1 Selecting the Compiler

First, we select the GNU C compiler (included in the TESSY installation) to execute the test.

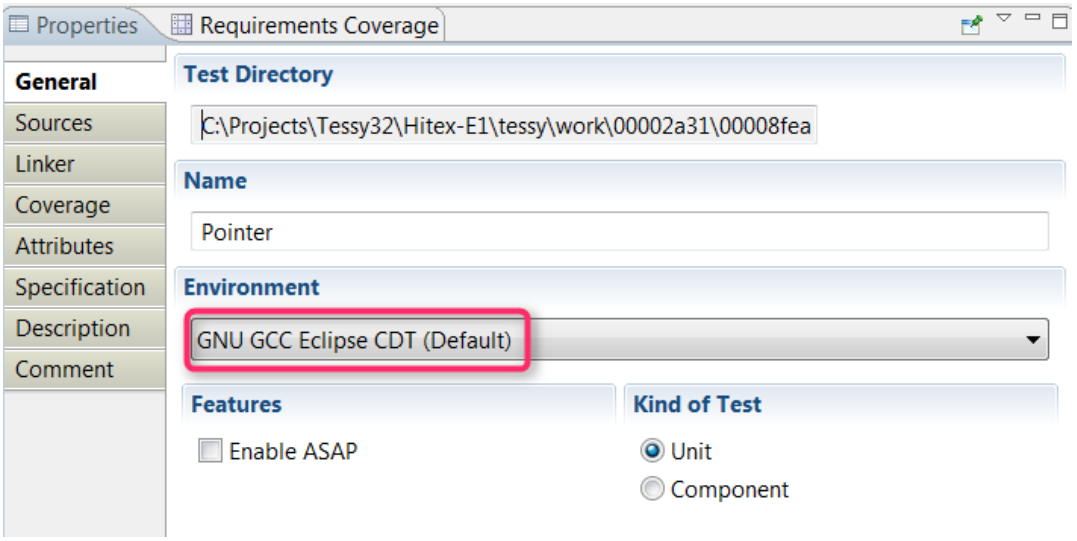


Fig. 2 Selecting the GNU C compiler

1.2.1.2 Selecting the Source Module

Second, we open the C source module (selectionsort.c) which contains the implementation of selectionsort(), as shown in Fig. 1.

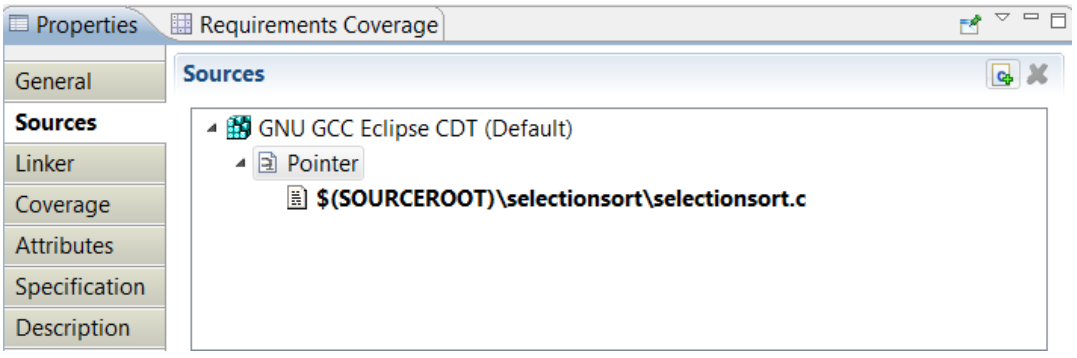


Fig. 3 Selecting the source file

1.2.2 The Interface of Selectionsort()

We let TESSY analyze selectionsort.c. In doing so, TESSY automatically determines the interface of selectionsort().

Interface of 'selectionsort'

Search for interface elements

	Passing	Target Passing	Data Format	Use In Report
External Functions				
Local Functions				
External Variables				
Global Variables				
Parameter				
int * array	IN	INOUT	None	Yes
int size	IN		None	Yes
Return				
int	OUT		None	Yes
Unused				
External Functions				
Local Functions				
External Variables				
Global Variables				
Types				
Structs and Unions				
Enums				
Defines				

Fig. 4 The interface of selectionsort(), automatically determined by TESSY

TESSY has determined that the pointer to the array is only input ("IN"), i.e. this pointer is not changed inside the function selectionsort(). However, the contents of the memory the pointer points to is both input and output ("INOUT").

The second parameter "size" of selectionsort() is also only input ("IN").

The return value of selectionsort() is output ("OUT"). (The return value is always output, if present.)

All passing directions were determined automatically by TESSY and we do not have to adjust anything manually.

1.2.3 The Difficulty of the Interface

Obviously, selectionsort() assumes that it gets a pointer pointing to something useful, in our case to the array holding the unsorted integers. This assumes that memory for that array was defined or allocated prior to the call to selectionsort(). Furthermore, the unsorted integers are assumed to be present in this memory area prior to the call to selectionsort().

Normally, definition and initialization of the array will be done in the parts of the application surrounding selectionsort(). During unit testing, these surrounding parts are not present. Therefore, TESSY has to rebuild this functionality in order to enable unit testing of selectionsort(). TESSY is directed to define and initialize the array during test case definition (see below).

1.2.4 Defining a Test Case

The interface determines the structure of a test case. Hence, after TESSY has determined the interface, we can go ahead and define a first test case. This is done in the **Test Items** view of TESSY's Test Data Editor perspective.

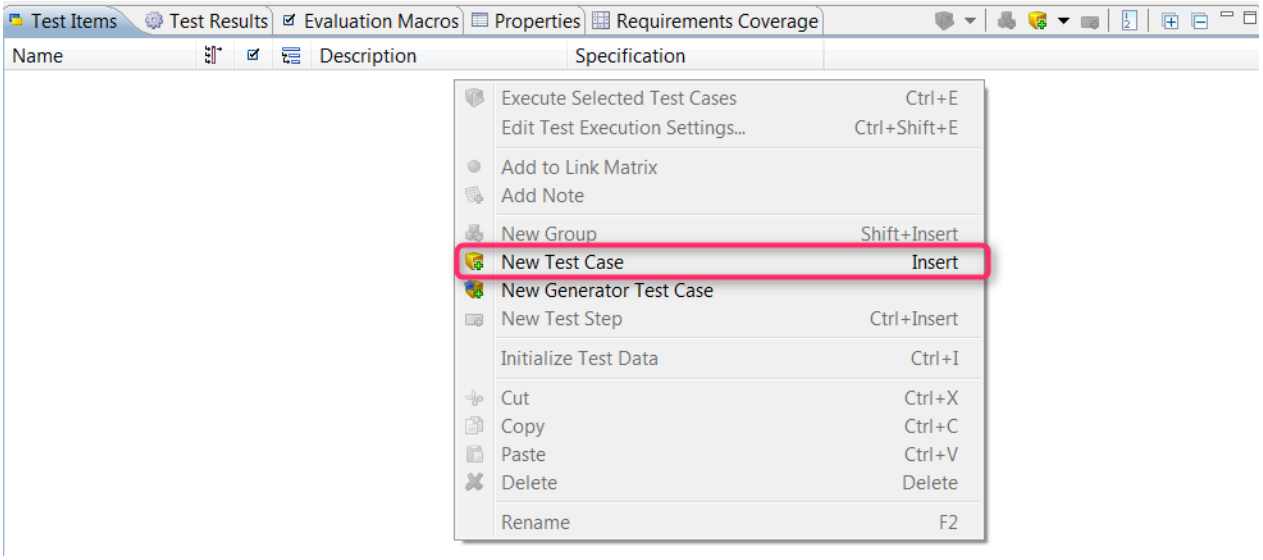


Fig. 5 Inserting a test case using the context menu of the Test Items view

After the test case is created, a column is displayed in the Test Data view of the Test Data Editor perspective. This column needs to be filled with test data at the marked locations.

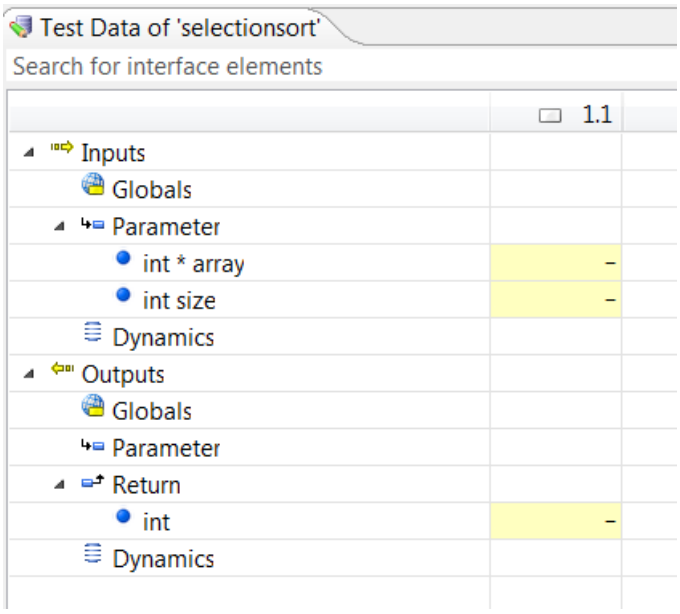


Fig. 6 The first test case, still empty

1.2.4.1 Define Dynamic Memory

We now direct TESSY to define memory for the pointer to point to. We do this by selecting the parameter **int *array**. Then we open the context menu and select **Create Pointer Target Value**.

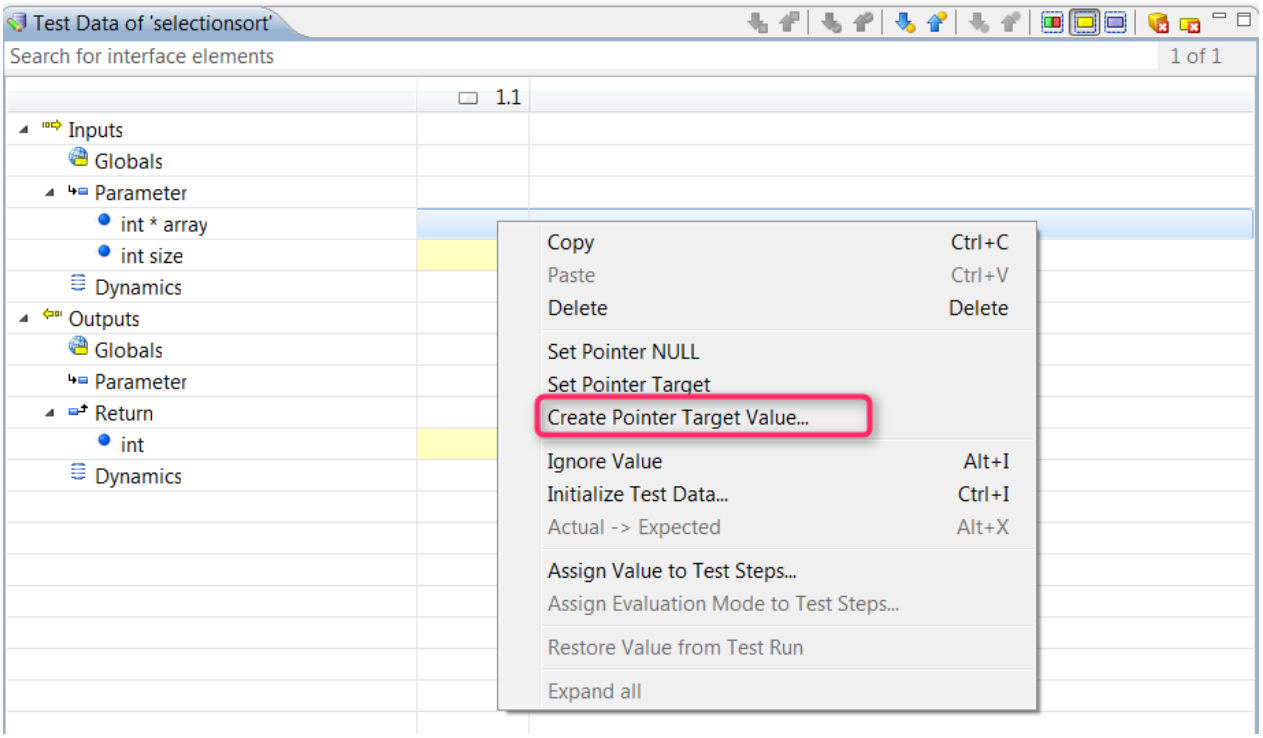


Fig. 7 Creating a memory area for the pointer to point to

After selecting **Create Pointer Target Value**, TESSY opens a dialog (Fig. 8).

(It would be an interesting test case to set the value of the pointer to NULL, what is easily possible by selecting an appropriate entry from the context menu. But test case specification is not the topic of this application note.)

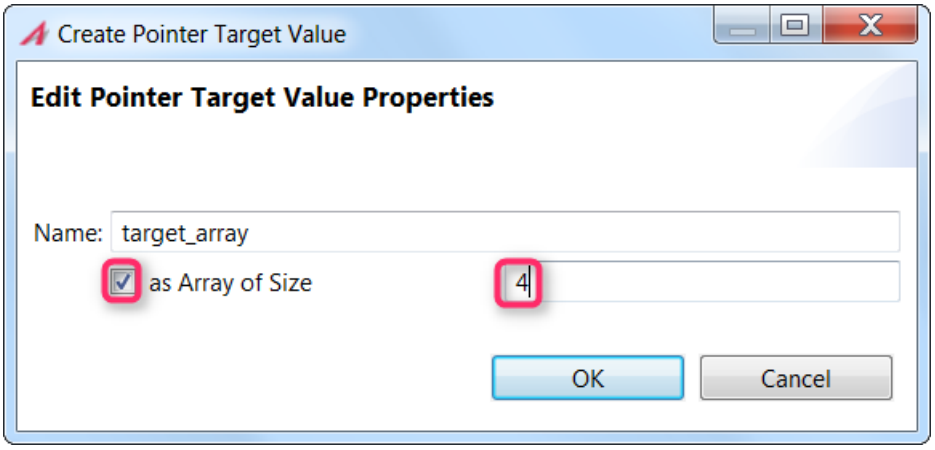


Fig. 8 Specifying the memory area to be created by TESSY

TESSY suggests the name “target_array”, but you can change this to another name if you want. Because the pointer shall not point to a scalar data item, but to an array of items (i.e. integers), you must enable the **as Array of Size** check box. Furthermore, you must specify an appropriate value for the number of items in the array, e.g. 4.

After clicking on **OK**, the Test Data view displays two dynamic memory areas (Fig. 9).

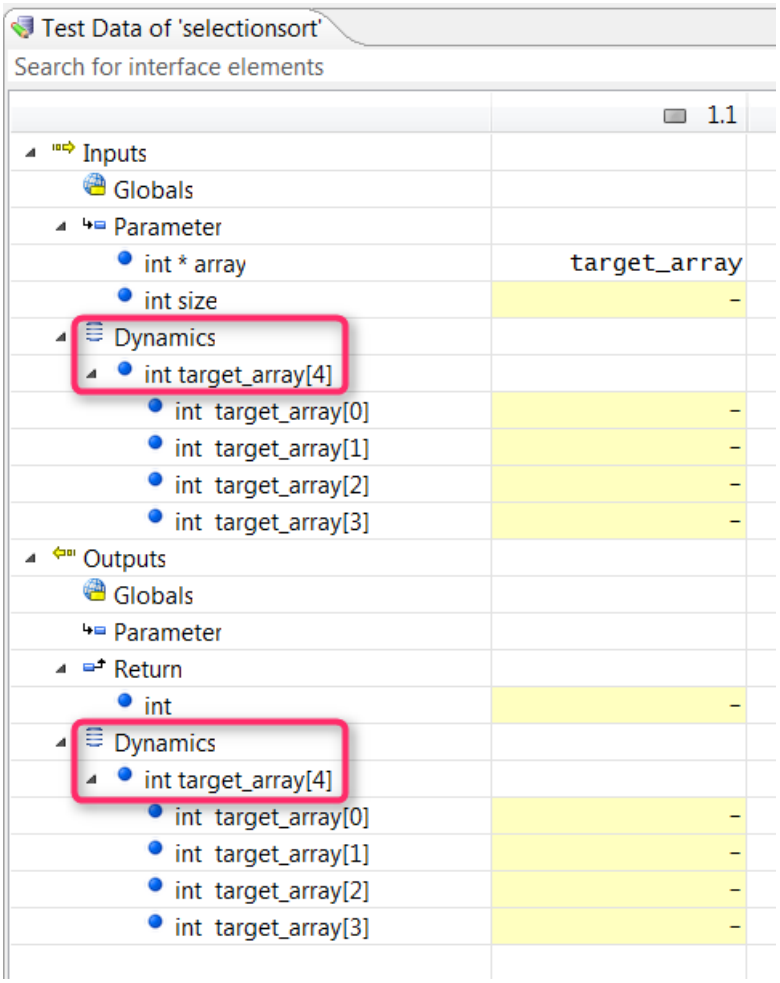


Fig. 9 Two dynamic memory areas are displayed in the Test Data Editor (TDE)

From the interface information of selectionsort(), TESSY knows that the dynamic array is not only input, but also output. Therefore, the dynamic memory area is not only displayed in the Inputs section (upper part), but also in the Outputs section (lower part).

1.2.4.2 Initializing with Random Values

You may now go ahead and type in values for the elements (i.e. integers) of the dynamic memory area. However, you may also let TESSY initialize the values, e.g. with random test input data. This may save a lot of work.

Use the context menu again and select **Initialize Test Data**.

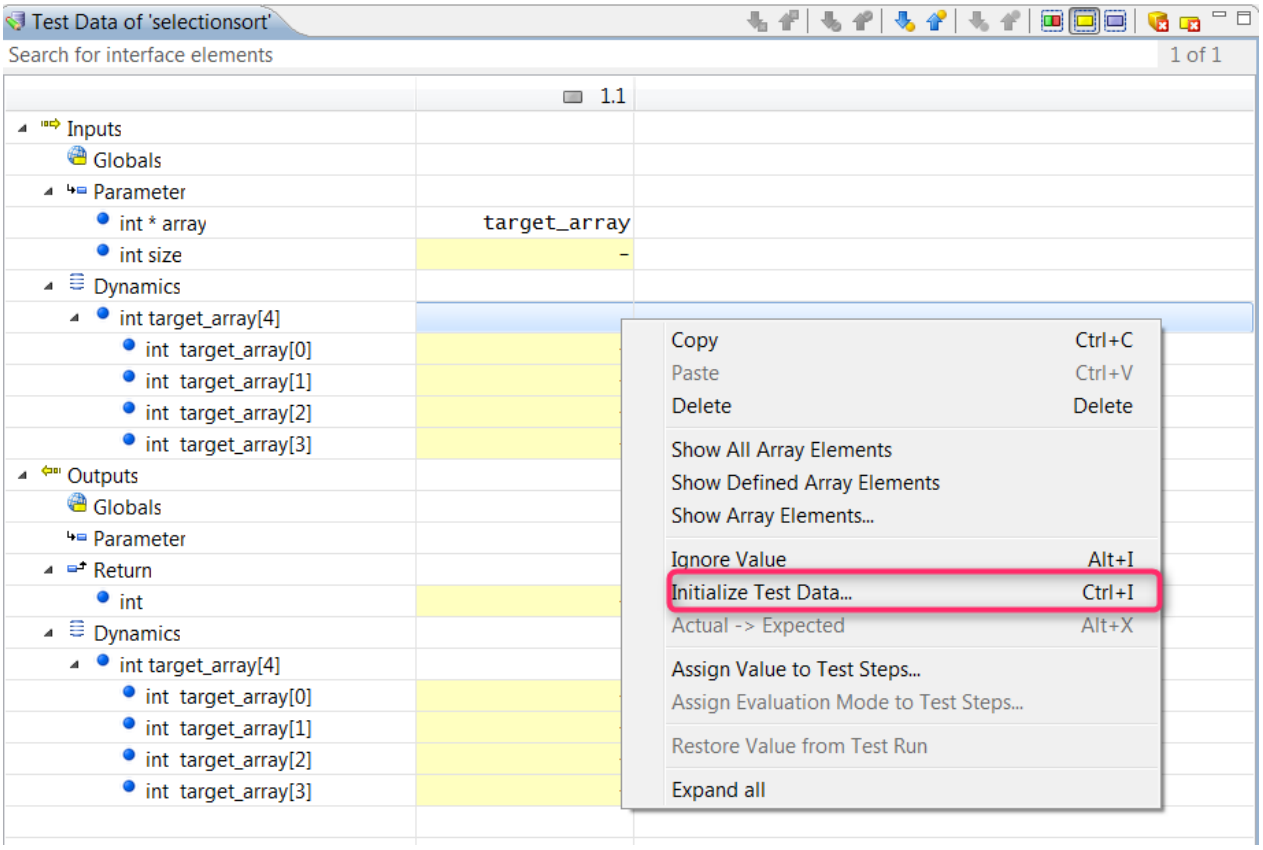


Fig. 10 Start initializing of the dynamic memory area

The following dialog opens (Fig. 11).

Select **Random** and define a range. Check **Initialize all array elements.**

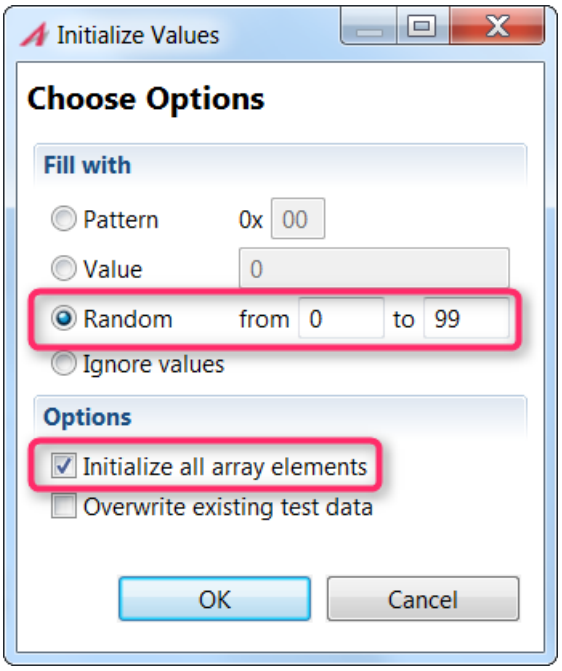


Fig. 11 Initializing the elements of the dynamic memory area with random values

After clicking on **OK** in the **Initialize Values** dialog, the test input data is initialized with random values.

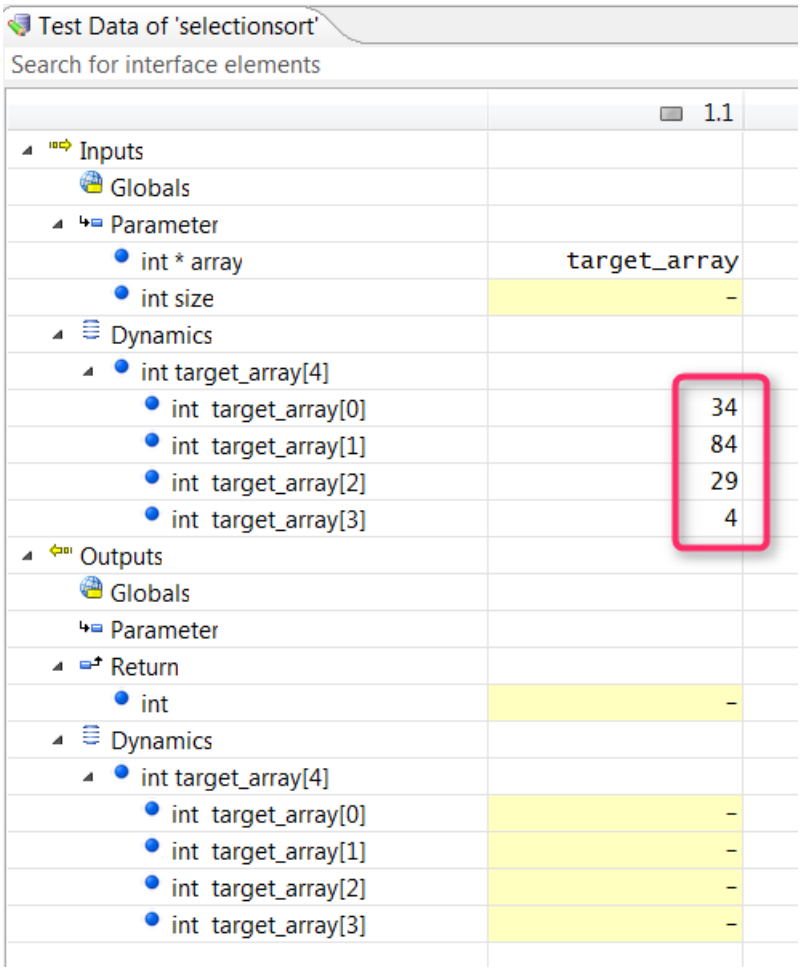


Fig. 12 The dynamic memory area is now initialized with random values

A Word on Using Random Test Input Data

It is tempting to automatically generate random test input data instead of thinking carefully about the necessary test cases and the test data that should be used.

But random test input data has a drawback: You need to check the results.

Unless you check the actual results of a test case with random input test data, this test case will only reveal "rough misbehavior" of the test object, i.e. a crash of the test object or an endless loop.

A solution could be an automatic test oracle. This is an instance that can determine from the input data, if an actual result is correct or not. A test oracle could be a program different to your test object that does the same calculations as your test object in a different way, e.g. by using a different algorithm, e.g. for sorting.

Unless you can make use of a test oracle, you have to check all your tests manually. (In fact, you are the test oracle). In this case, it makes no sense to create rather easily a huge amount of test cases with random input test data and then take the effort to check each actual test result manually, if it is correct or not.

Even if you dispose of a test oracle, some effort is needed to automate test evaluation.

The better way is to first put some effort in test case specification and reduce the number of test cases to a minimal set. During this process, normally the expected results of a certain test case are also specified without further effort.

However, random test input data provides an advantage: Random input test data may be the only way to find unsystematic errors in the software. However, such unsystematic errors should show up when the test coverage is determined after (systematic) unit testing.

1.2.4.3 Initializing "size"

The input parameter "size" is still not set. You have to set it manually to an appropriate value. The value of size indicates the number of items in the array that are to be sorted. Therefore, size must not be higher than the available elements in the dynamic memory area.

(It would be an interesting test case to set "size" to a value higher than the number of items in the array. But test case specification is not the topic of this application note.)

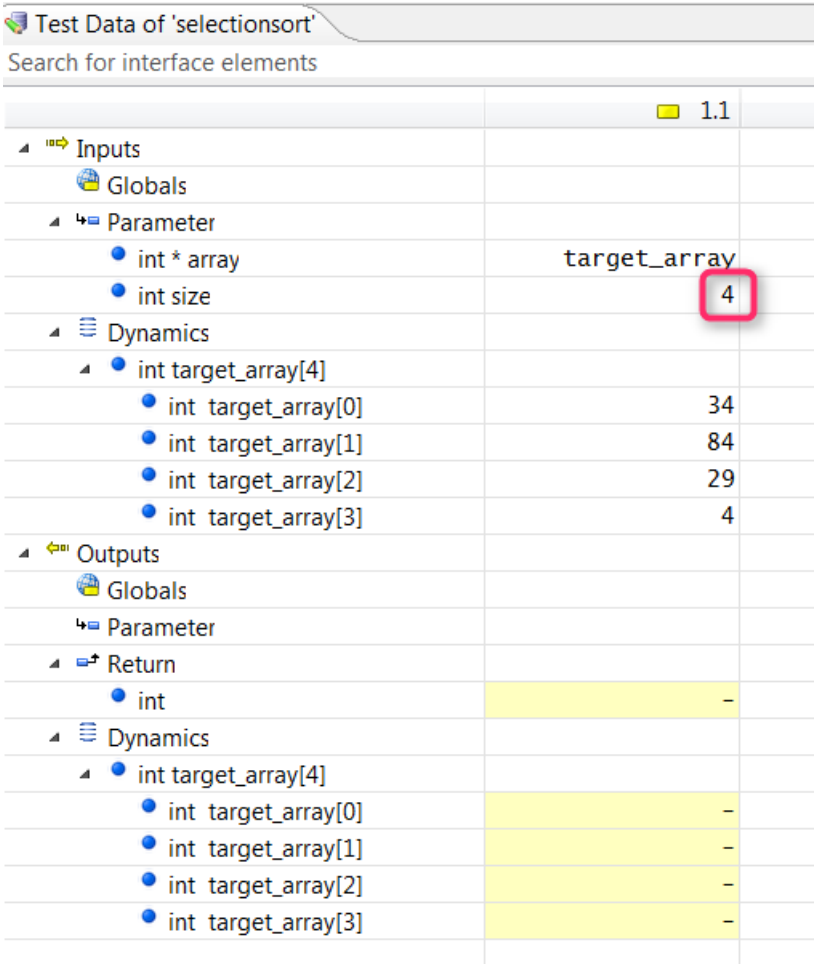


Fig. 13 Four elements are to be sorted

We have now specified all necessary input fields.

1.2.4.4 Specifying the Expected Output

Now we have finalized the definition of the input data in the upper part of the Test Data view. To complete the test case, we should also specify the expected data (output data) for this test case in the lower part of the Test Data view.

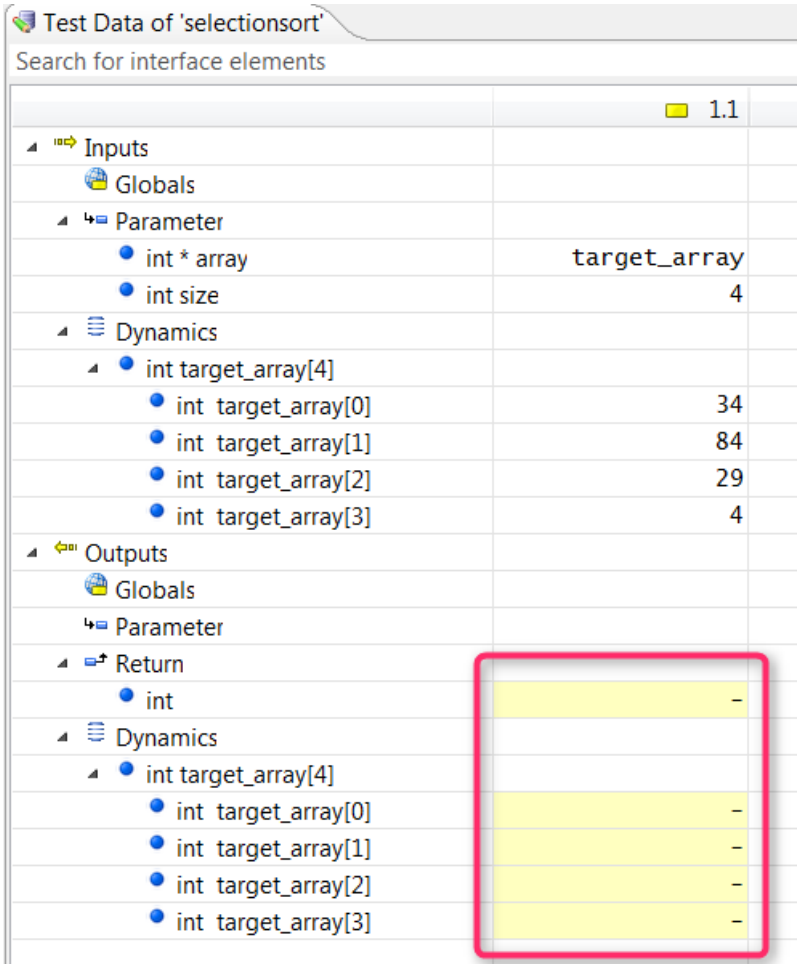


Fig. 14 The expected results are not yet defined

The expected data consist of the sorted array and the number of swaps necessary to create the sorted result.

Unfortunately, we cannot determine the array in sorted order without putting some effort in it. I.e. we could take the values generated at random by TESSY and sort them manually (using pencil and a sheet of paper, or another sorting program, etc.).

However, TESSY allows running a test case without having specified an expected result.

To use this feature for the array, we use the context menu and select **Ignore Values**.

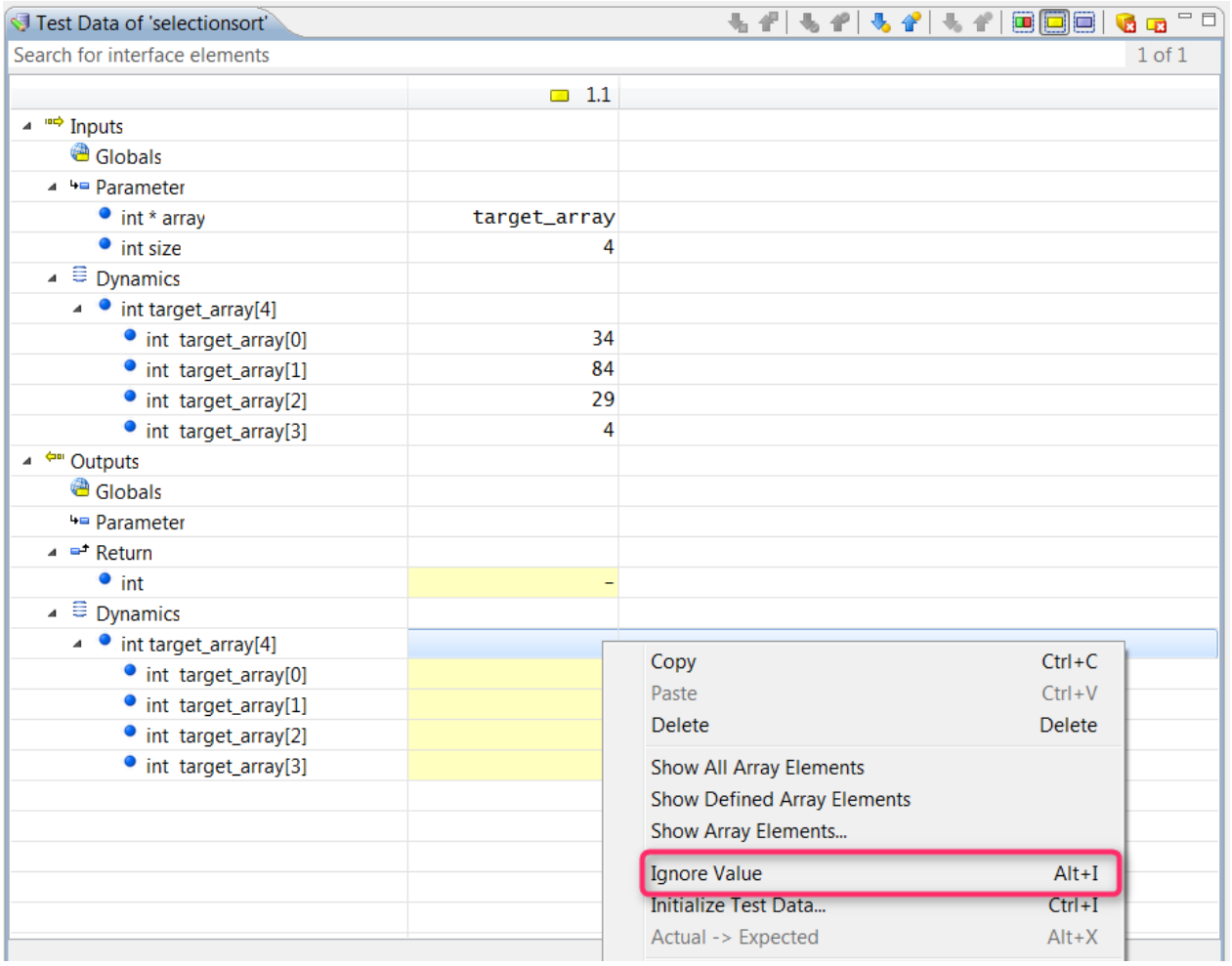


Fig. 15 Selecting **Ignore Value** from the context menu

This results in the display of "*none*" for all items of the array.

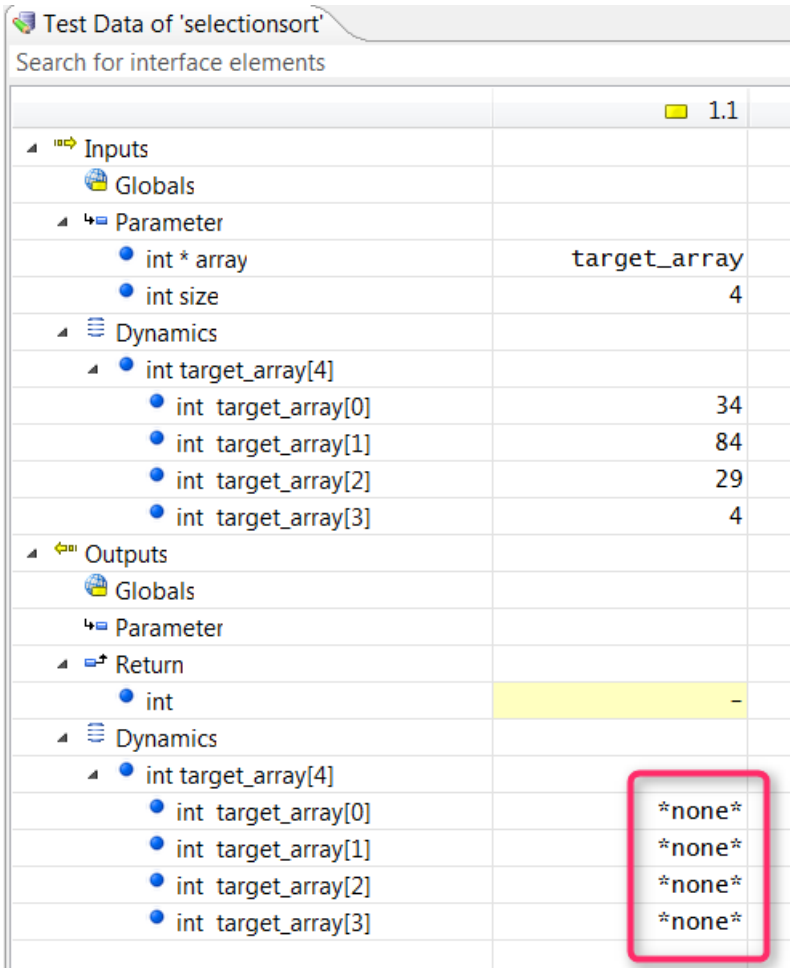


Fig. 16 The special value "*none*" for expected results means: "Take every result as correct result"

In case the special value "*none*" is used as expected result TESSY will take every actual result as correct. In other words: if you specify **Ignore Value** for a variable (i.e. if "*none*" is displayed), this test case will never fail with respect to this variable.

Remark

If **Ignore Value** is specified for an input variable, also the special value "*none*" is displayed. However, the meaning is different: TESSY will not initialize this variable prior to a test case execution.

To complete the test case, we still have to specify an expected value for the return value of selectionsort(). The return value indicates the number of swaps required to sort the array. We cannot determine the correct expected value without putting some effort in it. Therefore, we could take the same approach as for the values in the sorted array: We could select **Ignore Value** from the context menu, what would result in "*none*" to be displayed, what would in turn cause TESSY to accept every possible result as correct.

However, we can leave this field empty. This will also allow running the test case, but regardless of the actual result, this test will fail, due to the unset expected value.

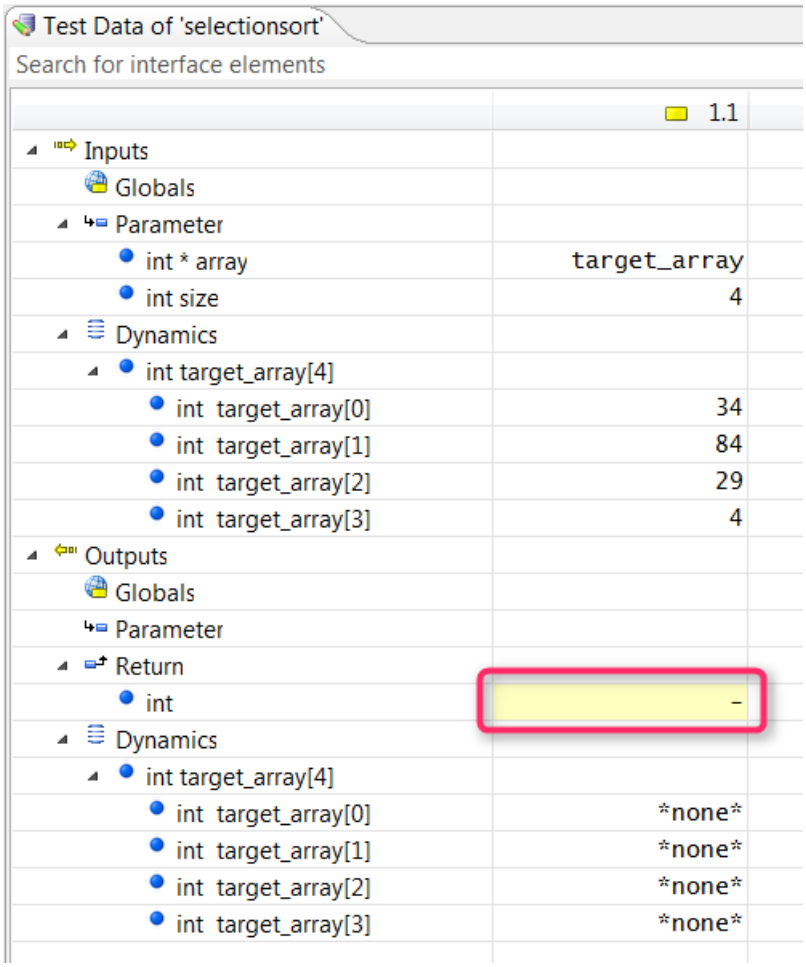


Fig. 17 If the field for an expected result is left empty, the test can be executed, but will fail due to this field.

1.2.5 Running the Test

After saving the test case, we can direct TESSY to execute this test case.

1.2.6 Checking the Test Results

1.2.6.1 Checking the Result of the Return Value

As expected, the test fails due to the missing expected value for the return value. This is indicated by the red symbol in the heading of the test case column.

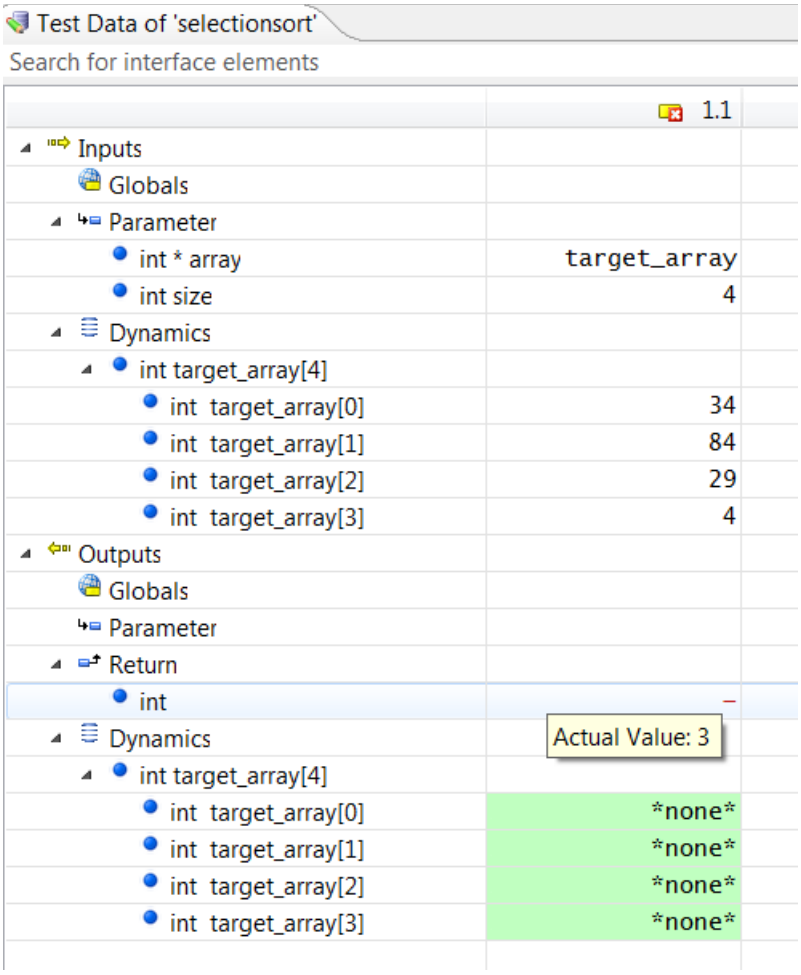


Fig. 18 The test has failed due to the missing expected return value

However, the actual value for the return value is displayed as tooltip. So we know the return value for our specific set of input data, and we can decide if this value is correct or not.

1.2.6.2 Checking Result in the Array

Also the actual values of the array are displayed in tooltip format. We can display the value of each item and decide, if it is correct or not.

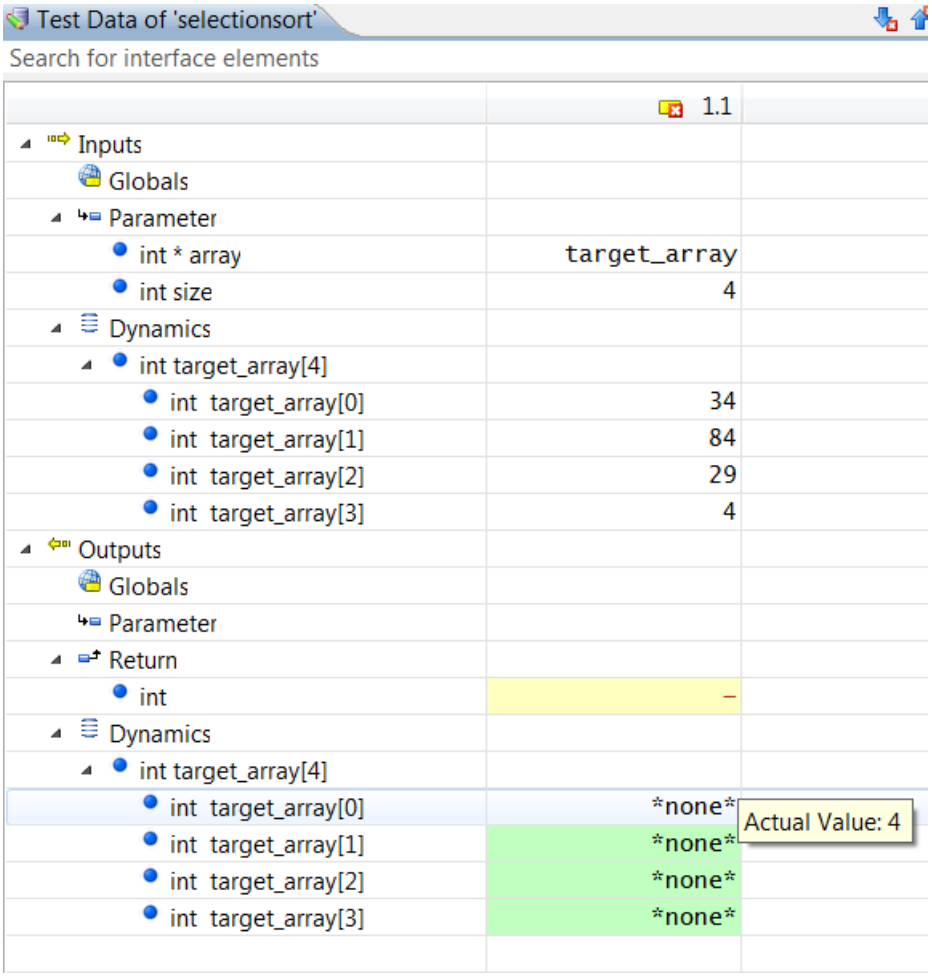


Fig. 19 The actual values of the array are also displayed

The actual value in the first element of the array is 4. This is correct, because 4 is the smallest value of the array elements.

Please note, that all array elements are displayed with green background, i.e. they are treated as correct by TESSY.

1.2.6.3 Checking Results Using Excel

You may also get a better overview of all actual results by exporting the test data of this test case to Microsoft Excel. From the TESSY main menu, select **Tools > Import/Export**.

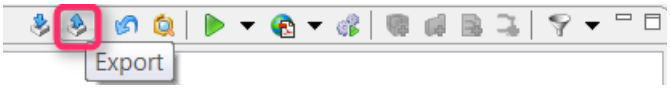


Fig. 20 Start the export to Microsoft Excel

In the dialog, specify a file and the desired action (see below). The extension of the file name (.xlsx) specifies the file format resulting from the export operation.

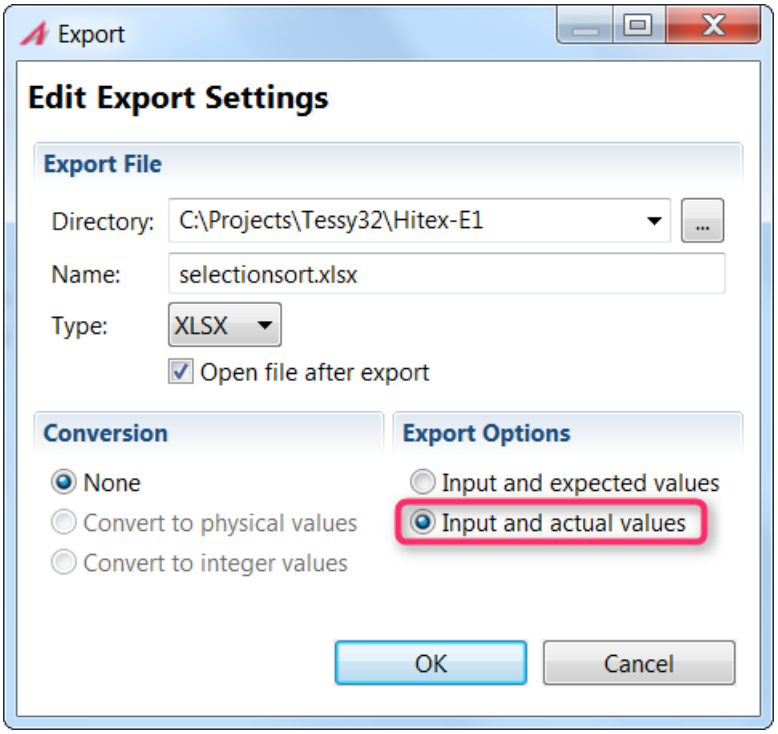


Fig. 21 Exporting the actual values to Microsoft Excel

	A	B	C	D	E	F	G	H	I	J	K	L
1		&target_array[0]	&target_array[1]	&target_array[2]	&target_array[3]	array	size	&target_array[0]	&target_array[1]	&target_array[2]	&target_array[3]	selectionsort()
2		i	i	i	i	i	i	o	o	o	o	o
3	tc1.1	34	84	29	4	target_array	4	4	29	34	84	3

Fig. 22 The actual values in Microsoft Excel (tab "Values")

The first row indicates the names of the interface variables. The second row indicates, if a variable is input or output. The input elements of the array are displayed in column B to E. The output elements are displayed in column H to K.

The data values of the test case are displayed in the third row. It is easily confirmed that 4, 29, 34, 84 is the sorted representation of 34, 84, 29, 4. You can also easily confirm that 3 swaps are necessary to get the sorted result

1.2.7 Turning Actual Results into Expected Results

We have now verified that the actual results of this test case are correct. We can now direct TESSY to turn the actual results of the current test case into expected results for the next test case execution. This is done by using the context menu of the Test Data view. If you use the context menu for the “Outputs” line, all actual result become expected results. It is also possible to do this only for single variables.

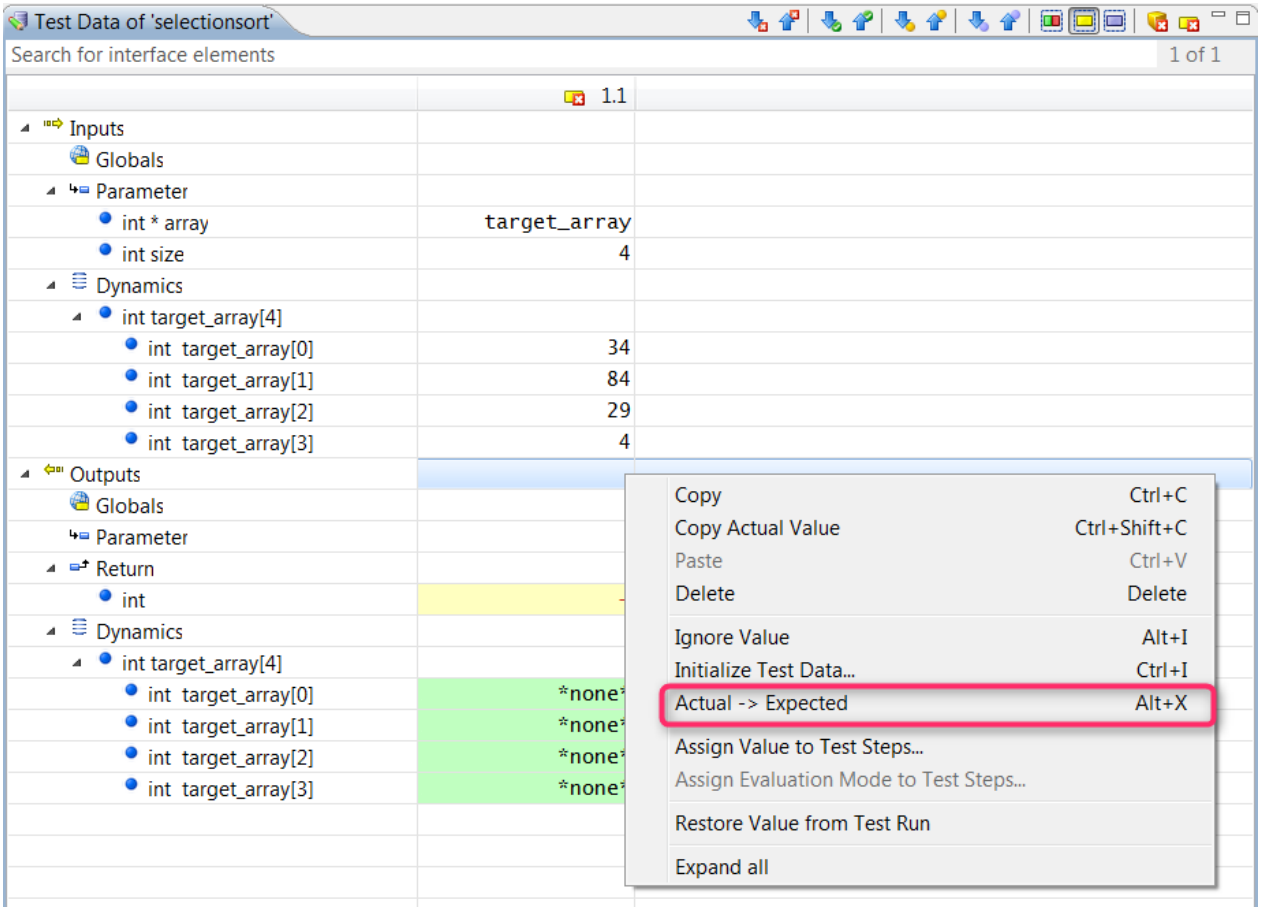


Fig. 23 Turning actual results into expected results

This last step is very important. It yields a correct expected result for the next test case execution.

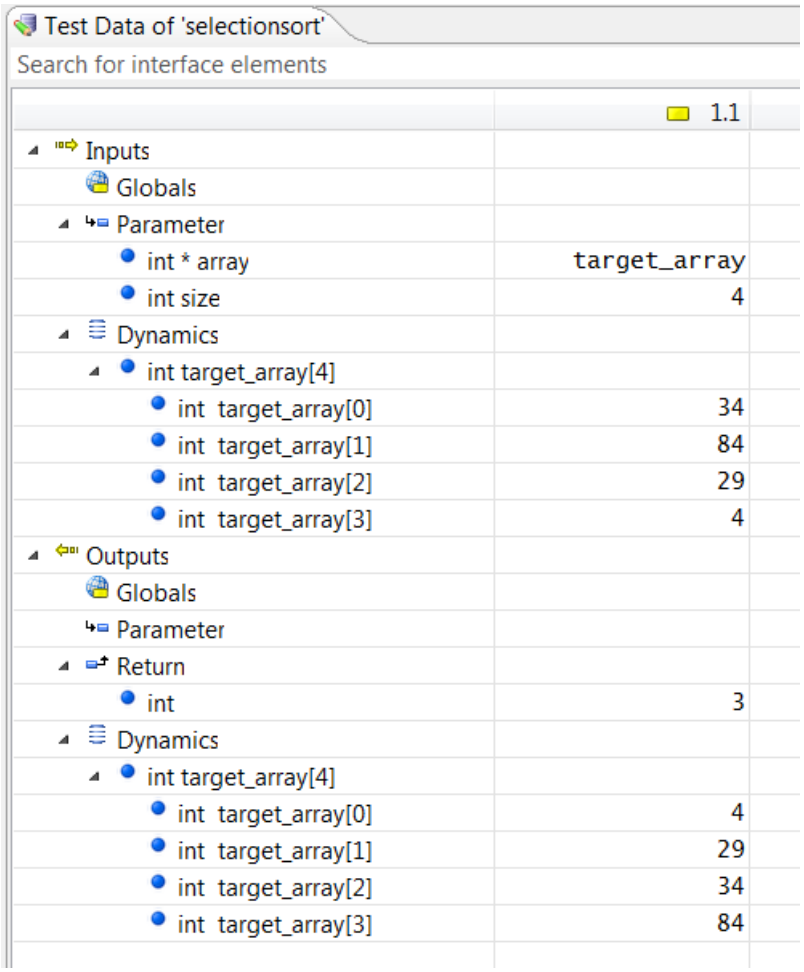


Fig. 24 The Test Data view after the command **Actual > Expected** was executed

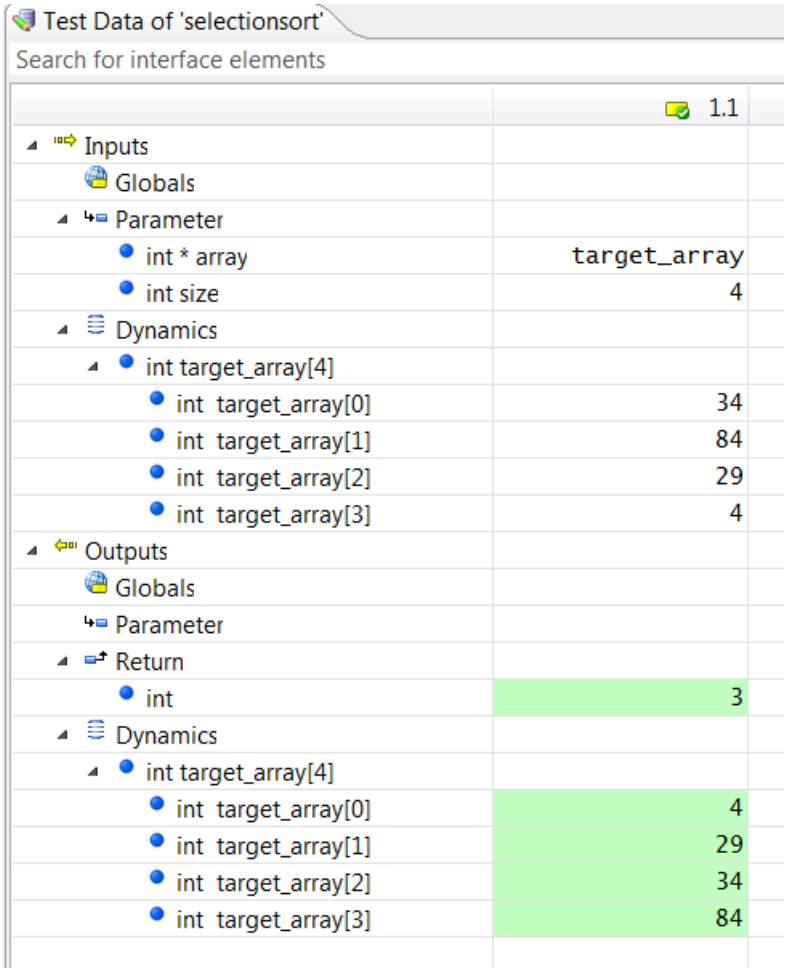


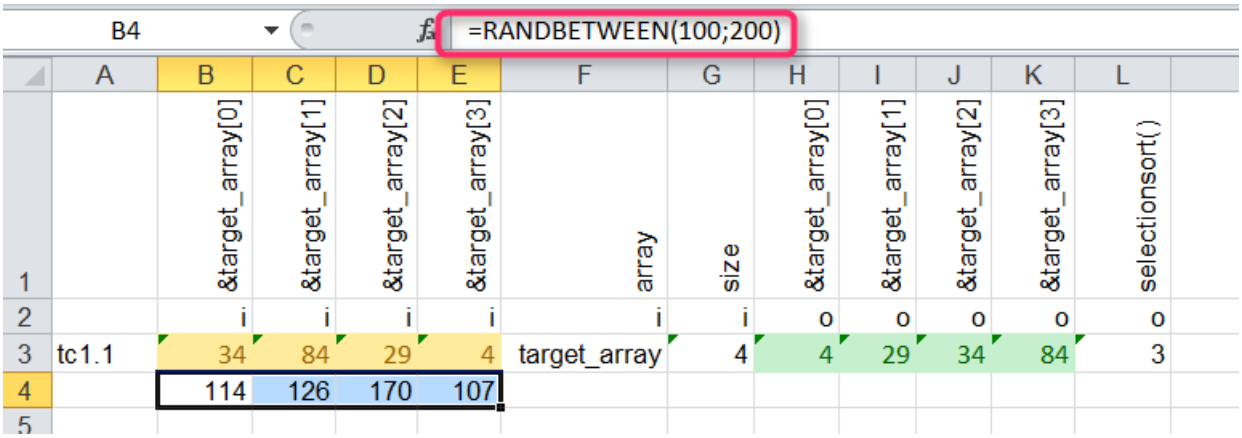
Fig. 25 The Test Data view after the test was executed with expected data specified

If we repeat the test, e.g. because the sorting algorithm has changed slightly, the repeated test will only pass if the test results are exactly the same test results as for the current test. We do not have to verify the test outcome manually.

1.3 Another Way of Specifying Random Test Data

TESSY can import test data from Excel files and from plain text files (ASCII, csv format).

This allows to easily importing random test data created by other sources than TESSY, e.g. created by Excel.



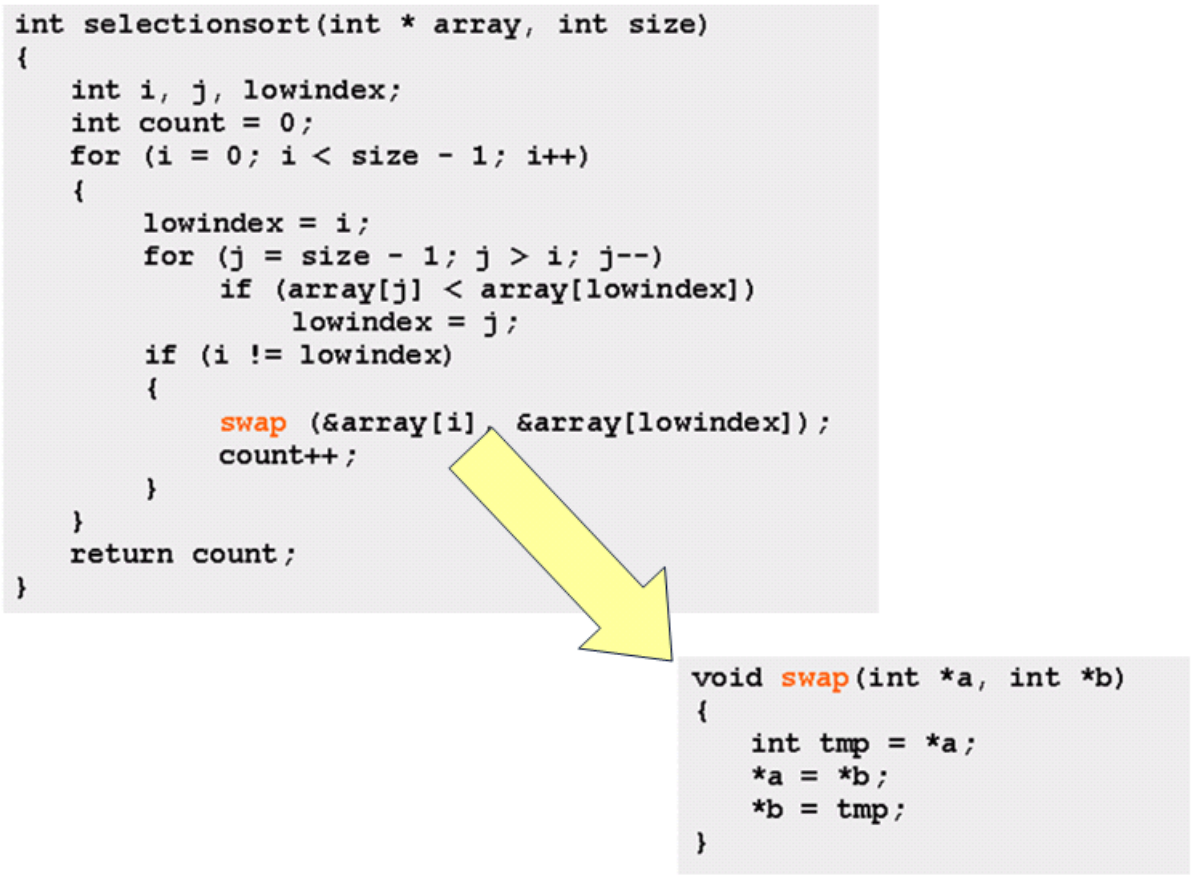
	A	B	C	D	E	F	G	H	I	J	K	L
1		&target_array[0]	&target_array[1]	&target_array[2]	&target_array[3]	array	size	&target_array[0]	&target_array[1]	&target_array[2]	&target_array[3]	selectionsort()
2		i	i	i	i	i	i	o	o	o	o	o
3	tc1.1	34	84	29	4	target_array	4	4	29	34	84	3
4		114	126	170	107							
5												

Fig. 26 Random test input data can also be created by Excel

In German versions of Excel, the corresponding Excel function is called "ZUFALLSBEREICH".

1.4 A Slightly Different Test Object

Here, the test object is implemented slightly different. The swap operation is implemented in a separate function, and not "inline" the test object. We assume, that swap() is implemented in a different source file than selectionsort(), and that this additional file is not accessible to TESSY. If TESSY analyzes only the source file, in which selectionsort() is implemented in, TESSY is not able to determine the passing direction of the memory area the pointer "array" points to. This is due to the fact that the actual assignment operation no longer takes place inside the source file that TESSY analyzes.



```
int selectionsort(int * array, int size)
{
    int i, j, lowindex;
    int count = 0;
    for (i = 0; i < size - 1; i++)
    {
        lowindex = i;
        for (j = size - 1; j > i; j--)
            if (array[j] < array[lowindex])
                lowindex = j;
        if (i != lowindex)
        {
            swap (&array[i], &array[lowindex]);
            count++;
        }
    }
    return count;
}
```

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Fig. 27 Selectionsort() and swap() are implemented in two different source files

Without access to the source file, in which swap() is implemented, TESSY cannot determine the passing directions completely.

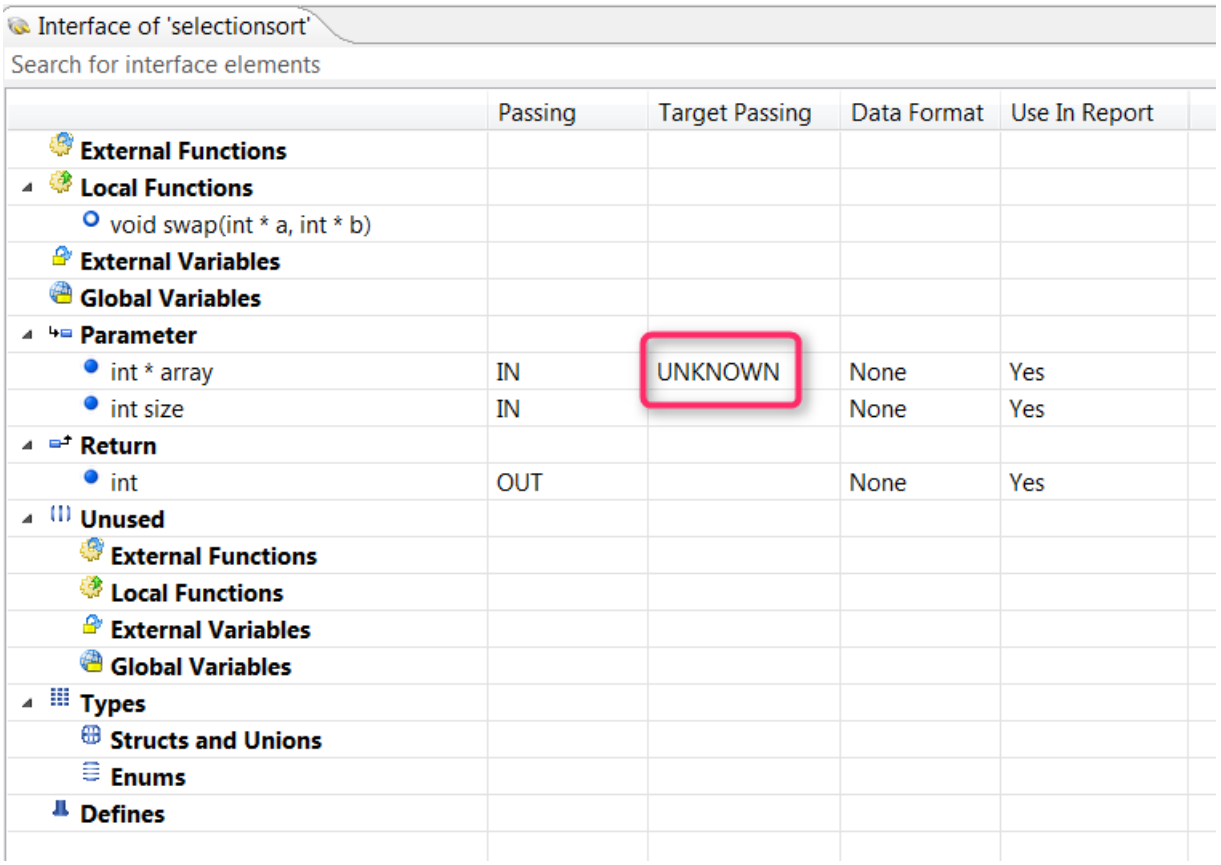


Fig. 28 TESSY has not the information to determine the complete interface

In this case, the test interface has to be adjusted manually by the user.

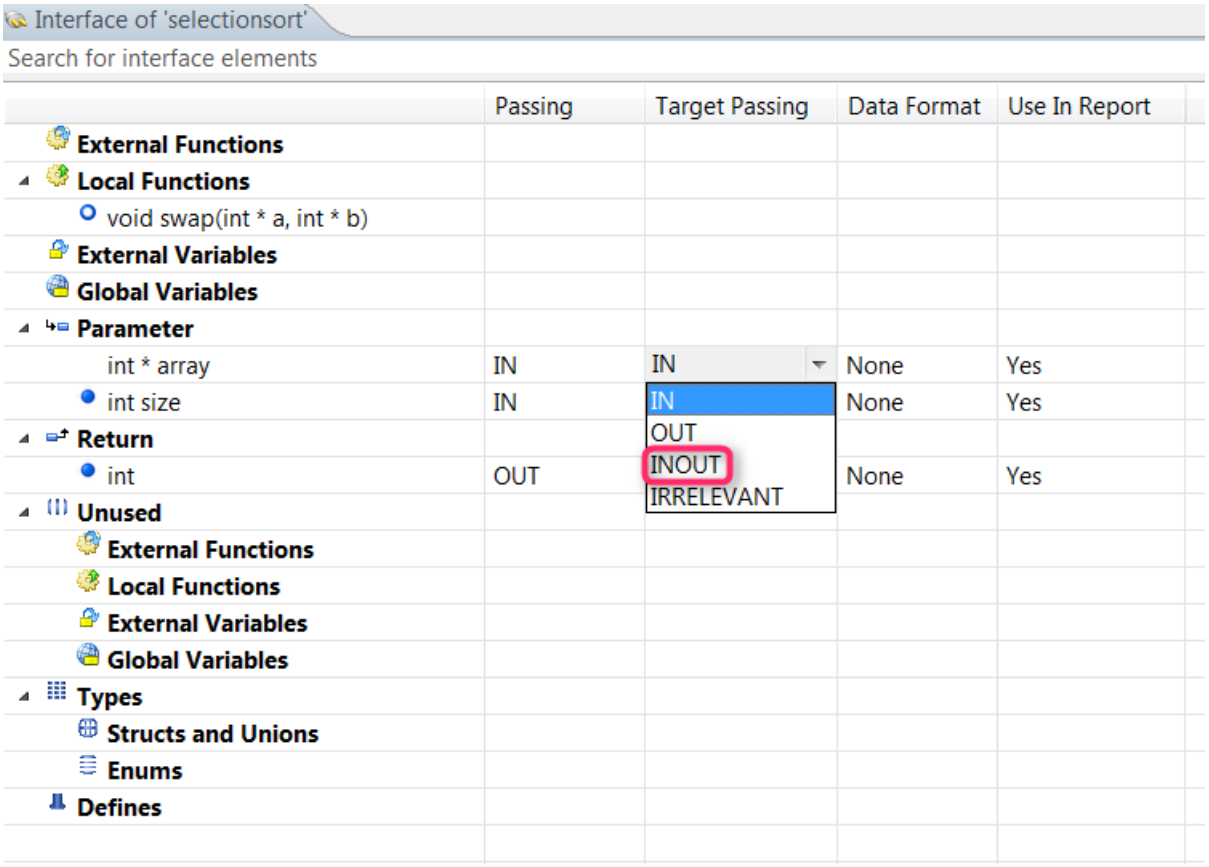


Fig. 29 The interface has to be adjusted manually

Without adjusting the interface, no test cases can be created.

2 Author

Frank Büchner
Dipl.-Inform.
Principal Engineer Software Quality
Hitex GmbH
Greschbachstr. 12
D-76229 Karlsruhe
Tel.: +49-721-9628-125
frank.buechner@hitex.de

Any comments are appreciated.

■