UTEST Documentation

Portable Unit Testing Platform

# General Test Construction

* Make sure to include the Module Under Test directly in the file that will define the test cases. This allows the test cases to have access to all static variables and functions within the module
* Go through the call trees of all functions in the Module Under Test and create dependency hooks for any function called that is not implemented in that module. Depending on the needs of particular test cases, these hooks m ay need to have functionality added so that the test cases can manipulate the circumstances around the particular Function Under Test in that test case.
* Create an individual function for each test case. This function will be included in the test case definition. Also create any prequel (before-all and/or before-each) functions that any particular set of test cases may need. These will be included in the test set definitions.
* The general sequence of testing is Initialization (UTEST\_init), Test Definition (UTEST\_add\_testset and UTEST\_add\_testcase), Test Performance (UTEST\_perform\_all), and Test Summary (UTEST\_summary).

# Function Reference

## Test Building

#### void UTEST\_add\_testcase(testfn)

This function is used to add a test case to the last test set created. The function containing the activities of the test case may be specified. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *testfn* | This is a pointer to the function containing the test case activities |

#### void UTEST\_add\_testset(fut, beforeall, beforeeach)

This function is used to build a set of test cases. Two accompanying functions may be specified: one to run at the start of the set and another to run before each test case in the set. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *fut* | This is a pointer to the function under test |
| in | *beforeall* | This is a pointer to the function to run at the start of the test set |
| in | *beforeeach* | This is a pointer to the function to run at the start of each test case |

#### void UTEST\_no\_action()

This function is used as a placeholder for unneeded before-all and before-each testset parameters. It takes no parameters and has no return value.

## Dependency Building

#### void UTEST\_count\_calls(x)

This function causes the specified subroutine to be flagged as having been called. This function is used in stubbed out dependencies to log control flow to be used for subsequent assertions.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *x* | This is a pointer to the subroutine to be counted |

## Test Actions

#### void UTEST\_let(var, val)

This function assigns a variable value and documents the action. This function is used in building test cases or prequel functions. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *var* | The variable to set |
| in | *val* | The value to which the variable is set |

#### void UTEST\_call(fn, ...)

This function calls the actual function under test and documents the action. The return value from the function is maintained for assertions. The function execution is also timed and this time is maintained for assertions. This function is used in building test cases or prequel functions. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *fn* | This is a pointer to the function under test |
| in | *...* | A variable-length list of any parameters that the function under test takes |

#### void UTEST\_void\_call(fn, ...)

This function calls the actual function under test and documents the action. It is used for functions of type void—i.e. that have no return value. The function execution is also timed and this time is maintained for assertions. This function is used in building test cases or prequel functions. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *fn* | This is a pointer to the function under test |
| in | *...* | A variable-length list of any parameters that the function under test takes |

#### void UTEST\_assert(var, exp)

This function tests and documents an equal assertion. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *var* | The variable to test |
| in | *exp* | The expected value to test against |

#### void UTEST\_assert\_between(var, min, max)

This function tests and documents the assertion that the variable is between the two values. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *var* | The variable to test |
| in | *min* | The lower-range value to test against |
| in | *max* | The upper-range value to test against |

#### void UTEST\_assert\_called(x)

This function tests and documents the assertion that the specified subroutine was called by the FUT. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *x* | This is a pointer to the subroutine to be checked for execution |

#### void UTEST\_assert\_less(var, exp)

This function tests and documents the assertion that the variable < expected value. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *var* | The variable to test |
| in | *exp* | The expected value to test against |

#### void UTEST\_assert\_more(var, exp)

This function tests and documents the assertion that the variable > expected value. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *var* | The variable to test |
| in | *exp* | The expected value to test against |

#### void UTEST\_assert\_not\_equal(var, exp)

This function tests and documents a not-equal assertion. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *var* | The variable to test |
| in | *exp* | The expected value to test against |

#### void UTEST\_assert\_return(exp)

This function tests and documents an equal assertion against the return value of the function under test. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *exp* | The expected value to test against |

#### void UTEST\_assert\_time(x, tol)

This function tests and documents the assertion that the execution time of the function under test is a certain expected time within a certain tolerance. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *x* | The expected execution time of the FUT in microseconds (us) |
| in | *tol* | The +/- tolerance of the expected value in microseconds (us) |

#### void UTEST\_assert\_time\_beats(x)

This function tests and documents the assertion that the execution time of the function under test takes less time than a certain expected time. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *x* | The maximum expected execution time of the FUT in microseconds (us) |

#### void UTEST\_assert\_time\_exceeds(x)

This function tests and documents the assertion that the execution time of the function under test takes longer than a certain expected time. It has no return value. Falseness of the assertion will cause the present test case to fail.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *x* | The minimum expected execution time of the FUT in microseconds (us) |

## Test Performance

#### void UTEST\_init()

This function must be called before any tests are configured or run. It takes care of platform and test variable initialization. It takes no parameters and has no return value.

#### void UTEST\_perform\_all()

This function executes all the test sets and their consituent test cases. It has no parameter or return value.

#### void UTEST\_perform\_case(testcase)

This function may be used to run a single test case. Only the before-each function of the parent test set will be run. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *testcase* | The overall index of the test case to perform |

#### void UTEST\_perform\_set( testset)

This function may be used to run a single test set. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *testset* | The index of the test set to perform |

## Reporting

#### void UTEST\_puts(str)

This function is the put string function that outputs the given null-terminated string to the standard test output. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *str* | A pointer to a null-terminated character array |

#### void UTEST\_ref(x)

This function outputs a reference number that may be associated with the subsequent test assertion. This function is used in building test cases or prequel functions. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *x* | The numeric requirement reference to output |

#### void UTEST\_benchmark()

This function document the execution time of the preceeding FUT. It has no parameters or return value.

#### void UTEST\_summary()

This function documents the accumulated results of all test set and their constituent test cases. It has no parameters or return value.

# Platform Porting

## File: utest\_config.h

#### #define UTEST\_CONFIG\_CLOCKS\_PER\_USEC

This is the platform-specific number of clock ticks returned by the time.h clock() function.

#### #define UTEST\_CONFIG\_MAX\_NUM\_SUB\_CALLS

This is the maximum allowed number of subroutines that may be called by the unit under test within any single test case.

#### #define UTEST\_CONFIG\_MAX\_NUM\_TEST\_CASES

This is the maximum allowed total number of test cases (they may be distributed through the individual test sets in any numbers).

#### #define UTEST\_CONFIG\_MAX\_NUM\_TEST\_SETS

This is the maximum allowed number of test sets.

#### #define UTEST\_CONFIG\_STRING\_BUFFER\_SIZE

This is the maximum allowed size for any single string sent via the **utest\_puts()** function.

## File: utest\_port.c

#### void utest\_init (void )

This function is platform-specific and should handle any necessary system hardware and/or peripheral initialization. It takes no parameters and has no return value.

#### void utest\_puts (char \* buf)

This function is platform-specific and should handle outputting the specified string on the desired test output peripheral. Any other occasional activities (e.g. watchdog timer) may also be handled here. It has no return value.

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *buf* | A pointer to a null-terminated character array |

# Example

## File: module.c

/\* module file \*/

extern int bar(int param);

/\* Req 101 – Foo shall return parameter2 if half of parameter2 is more than parameter1. \*/

/\* Req 102 – Foo shall return parameter1 if parameter1 is greater or equal to half of parameter2. \*/

/\* Req 103 – Foo shall return zero (0) if either parameter is negative. \*/

int foo (int param1, int param2)

{

int err, retval;

if ((param1 < 0) || (param2 < 0))

{

retval = 0;

}

else

{

/\* Call the divide by 2 function \*/

err = bar(param2);

if (err > param1)

{

retval = param2;

}

else

{

retval = param1;

}

}

return(retval);

}

## File: main.c

/\* main entry point and test definitions for low-level testing \*/

#include "utest\_core.c"

#include “utest\_port.c”

/\* INCLUDE THE MODULE UNDER TEST HERE \*/

#include "module.c"

/\* INCLUDE THE MODULE UNDER TEST HERE \*/

/\* PLACE THE HOOKS FOR DEPENDENCIES HERE \*/

int barParam, barReturn;

int bar(int param)

{

barParam = param;

return(barReturn);

}

/\* PLACE THE HOOKS FOR DEPENDENCIES HERE \*/

/\* TEST\_CODE HERE \*/

#define MAX\_PARAM 0x7FFE

int p1, p2;

void test\_foo\_before\_each()

{

UTEST\_let(barParam, UTEST\_INVALID\_DATA);

}

void test\_foo\_max\_param2\_param1\_equal()

{

UTEST\_let(p1,MAX\_PARAM/2);

UTEST\_let(p2,MAX\_PARAM);

UTEST\_let(barReturn, MAX\_PARAM/2);

UTEST\_call(foo, p1, p2);

UTEST\_assert(barParam, MAX\_PARAM);

UTEST\_ref(102);

UTEST\_assert\_return(p1);

}

void test\_foo\_max\_param2\_param1\_greater()

{

UTEST\_let(p1,MAX\_PARAM/2+1);

UTEST\_let(p2,MAX\_PARAM);

UTEST\_let(barReturn, p2/2);

UTEST\_call(foo, p1, p2);

UTEST\_assert(barParam, MAX\_PARAM);

UTEST\_ref(102);

UTEST\_assert\_return(p1);

}

void test\_foo\_max\_param2\_param2\_out()

{

UTEST\_let(p1,MAX\_PARAM/2-1);

UTEST\_let(p2,MAX\_PARAM);

UTEST\_let(barReturn, p2/2);

UTEST\_call(foo, p1, p2);

UTEST\_assert(barParam, MAX\_PARAM);

UTEST\_ref(101);

UTEST\_assert\_return(p2);

}

void test\_foo\_min\_param1\_param1\_equal()

{

UTEST\_let(p1,1);

UTEST\_let(p2,2);

UTEST\_let(barReturn, p2/2);

UTEST\_call(foo, p1, p2);

UTEST\_assert(barParam, 2);

UTEST\_ref(102);

UTEST\_assert\_return(p1);

}

void test\_foo\_min\_param1\_param1\_greater()

{

UTEST\_let(p1,1);

UTEST\_let(p2,1);

UTEST\_let(barReturn, p2/2);

UTEST\_call(foo, p1, p2);

UTEST\_assert(barParam, 1);

UTEST\_ref(102);

UTEST\_assert\_return(p1);

}

void test\_foo\_min\_param1\_param2\_out()

{

UTEST\_let(p1,1);

UTEST\_let(p2,4);

UTEST\_let(barReturn, p2/2);

UTEST\_call(foo, p1, p2);

UTEST\_assert(barParam, 4);

UTEST\_ref(101);

UTEST\_assert\_return(p2);

}

void test\_foo\_neg\_param1()

{

UTEST\_let(p1,-1);

UTEST\_let(p2,100);

UTEST\_let(barReturn, UTEST\_INVALID\_DATA);

UTEST\_call(foo, p1, p2);

UTEST\_ref(103);

UTEST\_assert\_return(0);

}

void test\_foo\_neg\_param2()

{

UTEST\_let(p1,100);

UTEST\_let(p2,-1);

UTEST\_let(barReturn, UTEST\_INVALID\_DATA);

UTEST\_call(foo, p1, p2);

UTEST\_ref(103);

UTEST\_assert\_return(0);

}

void test\_foo\_neg\_params()

{

UTEST\_let(p1,-1);

UTEST\_let(p2,-1);

UTEST\_let(barReturn, UTEST\_INVALID\_DATA);

UTEST\_call(foo, p1, p2);

UTEST\_ref(103);

UTEST\_assert\_return(0);

}

/\* TEST\_CODE HERE \*/

int main (void)

{

UTEST\_init();

UTEST\_add\_testset(foo,UTEST\_no\_action,test\_foo\_before\_each);

UTEST\_add\_testcase(test\_foo\_max\_param2\_param1\_equal);

UTEST\_add\_testcase(test\_foo\_max\_param2\_param1\_greater);

UTEST\_add\_testcase(test\_foo\_max\_param2\_param2\_out);

UTEST\_add\_testcase(test\_foo\_min\_param1\_param1\_equal);

UTEST\_add\_testcase(test\_foo\_min\_param1\_param1\_greater);

UTEST\_add\_testcase(test\_foo\_min\_param1\_param2\_out);

UTEST\_add\_testcase(test\_foo\_neg\_param1);

UTEST\_add\_testcase(test\_foo\_neg\_param2);

UTEST\_add\_testcase(test\_foo\_neg\_params);

UTEST\_perform\_all();

UTEST\_summary();

while(1);

return(0);

}

## Output

Test Set 1: foo

Test Case 1-1: test\_foo\_max\_param2\_param1\_equal

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = MAX\_PARAM/2 (16383)

> Let p2 = MAX\_PARAM (32766)

> Let barReturn = MAX\_PARAM/2 (16383)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be MAX\_PARAM (32766), is 32766: PASS

Reference: 102

< Returned value should be p1 (16383), is 16383: PASS

Result: \_PASS\_

Test Case 1-2: test\_foo\_max\_param2\_param1\_greater

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = MAX\_PARAM/2+1 (16384)

> Let p2 = MAX\_PARAM (32766)

> Let barReturn = p2/2 (16383)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be MAX\_PARAM (32766), is 32766: PASS

Reference: 102

< Returned value should be p1 (16384), is 16384: PASS

Result: \_PASS\_

Test Case 1-3: test\_foo\_max\_param2\_param2\_out

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = MAX\_PARAM/2-1 (16382)

> Let p2 = MAX\_PARAM (32766)

> Let barReturn = p2/2 (16383)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be MAX\_PARAM (32766), is 32766: PASS

Reference: 101

< Returned value should be p2 (32766), is 32766: PASS

Result: \_PASS\_

Test Case 1-4: test\_foo\_min\_param1\_param1\_equal

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 1 (1)

> Let p2 = 2 (2)

> Let barReturn = p2/2 (1)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be 2 (2), is 2: PASS

Reference: 102

< Returned value should be p1 (1), is 1: PASS

Result: \_PASS\_

Test Case 1-5: test\_foo\_min\_param1\_param1\_greater

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 1 (1)

> Let p2 = 1 (1)

> Let barReturn = p2/2 (0)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be 1 (1), is 1: PASS

Reference: 102

< Returned value should be p1 (1), is 1: PASS

Result: \_PASS\_

Test Case 1-6: test\_foo\_min\_param1\_param2\_out

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 1 (1)

> Let p2 = 4 (4)

> Let barReturn = p2/2 (2)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be 4 (4), is 4: PASS

Reference: 101

< Returned value should be p2 (4), is 4: PASS

Result: \_PASS\_

Test Case 1-7: test\_foo\_neg\_param1

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = -1 (-1)

> Let p2 = 100 (100)

> Let barReturn = UTEST\_INVALID\_DATA (57005)

+ Calling UUT foo...

- Returned from UUT.

Reference: 103

< Returned value should be 0 (0), is 0: PASS

Result: \_PASS\_

Test Case 1-8: test\_foo\_neg\_param2

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 100 (100)

> Let p2 = -1 (-1)

> Let barReturn = UTEST\_INVALID\_DATA (57005)

+ Calling UUT foo...

- Returned from UUT.

Reference: 103

< Returned value should be 0 (0), is 0: PASS

Result: \_PASS\_

Test Case 1-9: test\_foo\_neg\_params

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = -1 (-1)

> Let p2 = -1 (-1)

> Let barReturn = UTEST\_INVALID\_DATA (57005)

+ Calling UUT foo...

- Returned from UUT.

Reference: 103

< Returned value should be 0 (0), is 0: PASS

Result: \_PASS\_

Test Set Results: 9 of 9 test cases passed: 100%

Set Pass Fail

1 9 0 100%

------------------------

Total 9 0 100%

## Output with failures (negative value check not implemented)

Test Set 1: foo

Test Case 1-1: test\_foo\_max\_param2\_param1\_equal

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = MAX\_PARAM/2 (16383)

> Let p2 = MAX\_PARAM (32766)

> Let barReturn = MAX\_PARAM/2 (16383)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be MAX\_PARAM (32766), is 32766: PASS

Reference: 102

< Returned value should be p1 (16383), is 16383: PASS

Result: \_PASS\_

Test Case 1-2: test\_foo\_max\_param2\_param1\_greater

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = MAX\_PARAM/2+1 (16384)

> Let p2 = MAX\_PARAM (32766)

> Let barReturn = p2/2 (16383)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be MAX\_PARAM (32766), is 32766: PASS

Reference: 102

< Returned value should be p1 (16384), is 16384: PASS

Result: \_PASS\_

Test Case 1-3: test\_foo\_max\_param2\_param2\_out

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = MAX\_PARAM/2-1 (16382)

> Let p2 = MAX\_PARAM (32766)

> Let barReturn = p2/2 (16383)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be MAX\_PARAM (32766), is 32766: PASS

Reference: 101

< Returned value should be p2 (32766), is 32766: PASS

Result: \_PASS\_

Test Case 1-4: test\_foo\_min\_param1\_param1\_equal

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 1 (1)

> Let p2 = 2 (2)

> Let barReturn = p2/2 (1)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be 2 (2), is 2: PASS

Reference: 102

< Returned value should be p1 (1), is 1: PASS

Result: \_PASS\_

Test Case 1-5: test\_foo\_min\_param1\_param1\_greater

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 1 (1)

> Let p2 = 1 (1)

> Let barReturn = p2/2 (0)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be 1 (1), is 1: PASS

Reference: 102

< Returned value should be p1 (1), is 1: PASS

Result: \_PASS\_

Test Case 1-6: test\_foo\_min\_param1\_param2\_out

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 1 (1)

> Let p2 = 4 (4)

> Let barReturn = p2/2 (2)

+ Calling UUT foo...

- Returned from UUT.

< barParam should be 4 (4), is 4: PASS

Reference: 101

< Returned value should be p2 (4), is 4: PASS

Result: \_PASS\_

Test Case 1-7: test\_foo\_neg\_param1

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = -1 (-1)

> Let p2 = 100 (100)

> Let barReturn = UTEST\_INVALID\_DATA (57005)

+ Calling UUT foo...

- Returned from UUT.

Reference: 103

< Returned value should be 0 (0), is -1: FAIL

Result: \_FAIL\_

Test Case 1-8: test\_foo\_neg\_param2

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = 100 (100)

> Let p2 = -1 (-1)

> Let barReturn = UTEST\_INVALID\_DATA (57005)

+ Calling UUT foo...

- Returned from UUT.

Reference: 103

< Returned value should be 0 (0), is 100: FAIL

Result: \_FAIL\_

Test Case 1-9: test\_foo\_neg\_params

> Let barParam = UTEST\_INVALID\_DATA (57005)

> Let p1 = -1 (-1)

> Let p2 = -1 (-1)

> Let barReturn = UTEST\_INVALID\_DATA (57005)

+ Calling UUT foo...

- Returned from UUT.

Reference: 103

< Returned value should be 0 (0), is -1: FAIL

Result: \_FAIL\_

Test Set Results: 6 of 9 test cases passed: 66%

Set Pass Fail

1 6 3 66%

------------------------

Total 6 3 66%