Unity Test API

Running Tests

`RUN_TEST(func)`	Each Test is run within the macro `RUN_TEST`. This macro performs necessary setup before the test is called and handles cleanup and result tabulation afterwards.
`TEST_WRAP(function)`	If the test functions call helper functions and those helper functions have the ability to make assertions, calls to those helpers should be wrapped in a TEST_WRAP macro. This macro aborts the test if the helper triggered a failure.

Ignoring Tests

There are times when a test is incomplete or not valid for some reason. At these times, `TEST_IGNORE` can be called. Control will immediately be returned to the caller of the test, and no failures will be returned.

`TEST_IGNORE()`	Ignore this test and return immediately
`TEST_IGNORE_MESSAGE (message)`	Ignore this test and return immediately. Output a message stating why the test was ignored.

Aborting Tests

There are times when a test will contain an infinite loop on error conditions, or there may be reason to escape from the test early without executing the rest of the test. A pair of macros support this functionality in Unity. The first (`TEST_PROTECT`) sets up the feature, and handles emergency abort cases. `TEST_THROW` can then be used at any time within the tests to return to the last `TEST_PROTECT` call.

`TEST_PROTECT()`	Setup and Catch macro
`TEST_THROW (message)`	Abort Test macro

Example:

```
main()
{
    if (TEST_PROTECT() == 0)
    {
        MyTest();
    }
}
```

If MyTest calls TEST_THROW, a failure with the message provided will be inserted, and program control will immediately return to TEST_PROTECT with a non-zero return value.

Unity Assertion Summary

Basic Validity Tests

TEST_ASSERT_TRUE (condition)	Evaluates whatever code is in condition and fails if it evaluates to false
TEST_ASSERT_FALSE (condition)	Evaluates whatever code is in condition and fails if it evaluates to true
TEST_ASSERT (condition)	Another way of calling TEST_ASSERT_TRUE
TEST_ASSERT_UNLESS (condition)	Another way of calling TEST_ASSERT_FALSE
TEST_FAIL (message)	This test is automatically marked as a failure. The message is output stating why.

Numerical Assertions: Integers

TEST_ASSERT_EQUAL (expected, actual)	Another way of calling TEST_ASSERT_EQUAL_INT
TEST_ASSERT_EQUAL_INT (expected, actual)	Compare two integers for equality and display errors as signed integers.
TEST_ASSERT_EQUAL_UINT (expected, actual)	Compare two integers for equality and display errors as unsigned integers.
TEST_ASSERT_EQUAL_HEX8 (expected, actual)	Compare two integers for equality and display errors as an 8-bit hex value
TEST_ASSERT_EQUAL_HEX16 (expected, actual)	Compare two integers for equality and display errors as an 16-bit hex value
TEST_ASSERT_EQUAL_HEX32 (expected, actual)	Compare two integers for equality and display errors as an 32-bit hex value
TEST_ASSERT_EQUAL_HEX (expected, actual)	Another way of calling TEST_ASSERT_EQUAL_HEX32
TEST_ASSERT_INT_WITHIN (delta, expected, actual)	Asserts that the actual value is within plus or minus delta of the expected value.
TEST_ASSERT_EQUAL_MESSAGE (expected, actual, message)	Another way of calling TEST_ASSERT_EQUAL_INT_MESSAGE
TEST_ASSERT_EQUAL_INT_MESSAGE (expected, actual, message)	Compare two integers for equality and display errors as signed integers. Outputs a custom message on failure.

`TEST_ASSERT_EQUAL_UINT_MESSAGE (expected, actual, message)`	Compare two integers for equality and display errors as unsigned integers. Outputs a custom message on failure.
`TEST_ASSERT_EQUAL_HEX8_MESSAGE (expected, actual, message)`	Compare two integers for equality and display errors as an 8-bit hex value. Outputs a custom message on failure.
`TEST_ASSERT_EQUAL_HEX16_MESSAGE (expected, actual, message)`	Compare two integers for equality and display errors as an 16-bit hex value. Outputs a custom message on failure.
`TEST_ASSERT_EQUAL_HEX32_MESSAGE (expected, actual, message)`	Compare two integers for equality and display errors as an 32-bit hex value. Outputs a custom message on failure.
`TEST_ASSERT_EQUAL_HEX_MESSAGE (expected, actual, message)`	Another way of calling `TEST_ASSERT_EQUAL_HEX32_MESSAGE`

Numerical Assertions: Bitwise

`TEST_ASSERT_BITS (mask, expected, actual)`	Use an integer mask to specify which bits should be compared between two other integers. High bits in the mask are compared, low bits ignored.
`TEST_ASSERT_BITS_HIGH (mask, actual)`	Use an integer mask to specify which bits should be inspected to determine if they are all set high. High bits in the mask are compared, low bits ignored.
`TEST_ASSERT_BITS_LOW (mask, actual)`	Use an integer mask to specify which bits should be inspected to determine if they are all set low. High bits in the mask are compared, low bits ignored.
`TEST_ASSERT_BIT_HIGH (bit, actual)`	Test a single bit and verify that it is high. The bit is specified 0-31 for a 32-bit integer.
`TEST_ASSERT_BIT_LOW (bit, actual)`	Test a single bit and verify that it is low. The bit is specified 0-31 for a 32-bit integer.

Numerical Assertions: Floats

`TEST_ASSERT_FLOAT_WITHIN (delta, expected, actual)`	Asserts that the actual value is within plus or minus delta of the expected value.

String Assertions

`TEST_ASSERT_EQUAL_STRING (expected, actual)`	Compare two null-terminate strings. Fail if any

	character is different or if the lengths are different.
`TEST_ASSERT_EQUAL_STRING_MESSAGE (expected, actual, message)`	Compare two null-terminate strings. Fail if any character is different or if the lengths are different. Output a custom message on failure.

Pointer Assertions

Most pointer operations can be performed by simply using the integer comparisons above. However, a couple of special cases are added for clarity.

`TEST_ASSERT_NULL (pointer)`	Fails if the pointer is not equal to NULL
`TEST_ASSERT_NOT_NULL (pointer)`	Fails if the pointer is equal to NULL