Test Case Validation Macros

These macros can be used in test case and stub sources to validate test case execution.

Macro	Definition
`CPPTEST_FAIL(message)`	Fails unconditionally with a custom message.
`CPPTEST_ASSERT(condition)`	Asserts a condition that evaluates to true.
`CPPTEST_ASSERT_BITS(mask, expected, actual)`	Equivalent to `CPPTEST_ASSERT_BITS_MESSAGE`, but uses a general assertion message.
`CPPTEST_ASSERT_BITS_MESSAGE (message, mask, expected, actual)`	Asserts that masked bits (usingmask) of expected and actual are equal and prints a custom message on failure (mask, expected and actual are integer types). Only high bits from the mask are comparedinexpected and actual.
`CPPTEST_ASSERT_BITS_HIGH (mask, actual)`	Equivalent to `CPPTEST_ASSERT_BITS_HIGH` but uses a general assertion message.
`CPPTEST_ASSERT_BITS_HIGH_MESSAGE(message, mask, actual)`	Asserts that the masked bits (usingmask) of actual are high and prints a custom message on failure (mask and actual are integer types). Only high bits from the mask are inspected in actual.
`CPPTEST_ASSERT_BITS_LOW(mask, actual)`	Equivalent to `CPPTEST_ASSERT_BITS_LOW` but uses a general assertion message.
`CPPTEST_ASSERT_BITS_LOW_MESSAGE(message, mask, actual)`	Asserts that the masked bits (usingmask) of actual are low and prints a custom message on failure (mask and actual are integer types). Only high bits from the mask are inspected in actual.
`CPPTEST_ASSERT_BIT_HIGH_MESSAGE(message, n, actual)`	Asserts that the nth bit of actual is high and prints a custom message on failure (n and actual are integer types).
`CPPTEST_ASSERT_BIT_HIGH(n, actual)`	Equivalent to `CPPTEST_ASSERT_BIT_HIGH` but uses a general assertion message.
`CPPTEST_ASSERT_BIT_LOW_MESSAGE (message, n, actual)`	Asserts that the nth bit of actual is low and prints a custom message on failure (n and actual are integer types).
`CPPTEST_ASSERT_BIT_LOW(n, actual)`	Equivalent to `CPPTEST_ASSERT_BIT_LOW` but uses a general assertion message.
`CPPTEST_ASSERT_MESSAGE (message, condition)`	Asserts a condition that evaluates to true. Provides a custom message on failure.
`CPPTEST_ASSERT_EQUAL (expected, actual)`	Asserts that two values are equal.
`CPPTEST_ASSERT_EQUAL_MESSAGE (message, expected, actual)`	Asserts that two values are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_BOOL_EQUAL (expected, actual)`	Asserts that two boolean values are equal. If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_BOOL_EQUAL_MESSAGE(message, expected, actual)`	Asserts that two boolean values are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_INTEGER_EQUAL (expected, actual)`	Asserts that two integer values are equal. If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_INTEGER_EQUAL_MESSAGE(message, expected, actual)`	Asserts that two integer values are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_INTEGER_EQUAL_DELTA(expected, actual, delta)`	Asserts that two integer values are equal (up to the 'delta' accuracy).
`CPPTEST_ASSERT_INTEGER_EQUAL_DELTA_MESSAGE(message, expected, actual, delta)`	Asserts that two integer values are equal (up to the 'delta' accuracy). Provides a custom message on failure.
`CPPTEST_ASSERT_INTEGER_ARRAY_EQUAL(expected, actual, size)`	Asserts that two arrays of integer values are equal. First 'size' array elements are compared,
`CPPTEST_ASSERT_INTEGER_ARRAY_EQUAL_DELTA(expected, actual, size, delta)`	Asserts that two arrays of integer values are equal (up to the 'delta' accuracy). First 'size' array elements are compared.

`CPPTEST_ASSERT_UINTEGER_EQUAL (expect ed, actual)`	Asserts that two unsigned integer values are equal. If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_UINTEGER_EQUAL_ MESSAGE(message, expected, actual)`	Asserts that two unsigned integer values are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_UINTEGER_EQUAL_ DELTA(expected, actual, delta)`	Asserts that two unsigned integer values are equal (up to the 'delta' accuracy).
`CPPTEST_ASSERT_UINTEGER_EQUAL_ DELTA_MESSAGE(message, expected, actual, delta)`	Asserts that two unsigned integer values are equal (up to the 'delta' accuracy). Provides a custom message on failure.
`CPPTEST_ASSERT_UINTEGER_ARRAY_ EQUAL(expected, actual, size)`	Asserts that two arrays of unsigned integer values are equal. First 'size' array elements are compared.
`CPPTEST_ASSERT_UINTEGER_ARRAY_ EQUAL_DELTA(expected, actual, size, delta)`	Asserts that two arrays of unsigned integer values are equal (up to the 'delta' accuracy). First 'size' array elements are compared.
`CPPTEST_ASSERT_FLOAT_EQUAL_DEL TA(expected, actual, delta)`	Asserts that two floating point values are equal (up to the delta accuracy). First 'size' array elements are compared.
`CPPTEST_ASSERT_FLOAT_EQUAL_DEL TA_MESSAGE(message, expected, actual, delta)`	Asserts that two floating point values are equal (up to the delta accuracy). Provides a custom message on failure.
`CPPTEST_ASSERT_FLOAT_ARRAY_EQU AL_DELTA(expected, actual, size, delta)`	Asserts that two arrays of floating point values are equal (up to the 'delta' accuracy).
`CPPTEST_ASSERT_CSTR_EQUAL (expected, actual)`	Asserts that two C-style strings are equal. If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_CSTR_EQUAL_MESS AGE (message, expected, actual)`	Asserts that two C-style strings are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_CSTR_ARRAY_EQUA L(expected, actual, size)`	Asserts that two arrays of C-style strings are equal. First 'size' array elements are compared.
`CPPTEST_ASSERT_CSTR_N_EQUAL (expected, actual, max_size)`	Asserts that two C-style strings are equal (only first 'max_size' characters are compared). If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_CSTR_N_EQUAL_ME SSAGE(message, expected, actual, max_size)`	Asserts that two C-style strings are equal (only first 'max_size' characters are compared). Provides a custom message on failure.
`CPPTEST_ASSERT_MEM_BUFFER_EQUA L(expected, actual, size)`	Asserts that two data buffers are equal (compares 'size' number of bytes). If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_MEM_BUFFER_EQUA L_MESSAGE(message, expected, actual, size)`	Asserts that two data buffers are equal (compares 'size' number of bytes). Provides a custom message on failure.
`CPPTEST_ASSERT_PTR_EQUAL (expected, actual)`	Asserts that two pointer values are equal. If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_PTR_EQUAL_MESSA GE(message, expected, actual)`	Asserts that two pointer values are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_PTR_ARRAY_EQUAL (expected, actual, size)`	Asserts that two arrays of pointers are equal. First 'size' array elements are compared.
`CPPTEST_ASSERT_WCSTR_EQUAL (expected, actual)`	Asserts that two C-style wide character strings are equal. If it fails, both the actual and expected values are reported. Failed assertions can be automatically validated by C++test from the Quality Tasks view.
`CPPTEST_ASSERT_WCSTR_EQUAL_MES SAGE(message, expected, actual)`	Asserts that two C-style wide character strings are equal. Provides a custom message on failure.
`CPPTEST_ASSERT_ENUM_EQUAL(scop ed_enum_name, expected, actual)`	Asserts that two integral values are equal. If it fails both the actual and expected values are reported as enumerators. If the integral value does not map to enumerator identifier from specified enum then test case error is reported. See Handling Enum Values for details.

`CPPTEST_ASSERT_ENUM_EQUAL_MESSAGE(scoped_enum_name, msg, expected, actual)`	Asserts that two integral values are equal, like `CPPTEST_ASSERT_ENUM_EQUAL` described above, and in addition, provides a custom message on failure. See Handling Enum Values for details.
`CPPTEST_ASSERT_THROW (expression, ExceptionType)`	Asserts that the given expression throws an exception of the specified type.
`CPPTEST_ASSERT_THROW_MESSAGE (message, expression, ExceptionType)`	Asserts that the given expression throws an exception ofspecifiedtype. Provides a custom message on failure.
`CPPTEST_ASSERT_NO_THROW (expression)`	Asserts that the given expression does not throw any exceptions.
`CPPTEST_ASSERT_NO_THROW_MESSAGE(message, expression)`	Asserts that the given expression does not throw any exceptions. Provides a custom message on failure.
`CPPTEST_ASSERT_EXTERNAL (program, param1, param2, param3)`	Uses an external application to validate values. The first parameter specifies the executable to run. The program receives three parameters (param1, param2, param3). If it returns a non-zero value, the assert fails and the program stdout is used as the failure message.
`CPPTEST_ASSERT_EXTERNAL_MESSAGE(message, program, param1, param2, param3)`	Uses an external application to validate values. The first parameter specifies the executable to run. The program receives three parameters (param1, param2, param3). If it returns a non-zero value, the assert fails and the custom message is used as the failure message.
`CPPTEST_ASSERT_MEMORY_LEAKS()`	Asserts that dynamic memory blocks allocated during the current test case execution were deallocated before this macro call. Note: This macro requires the Memory Monitoring feature to be enabled in the Test Configuration (see Runtime Error Detection).

`CPPTEST_ASSERT_EQUAL_FMT (expected, actual, format)`
 	
`CPPTEST_ASSERT_EQUAL_DELTA_FMT (expected, actual, delta, format`
	Asserts that two values are equal. Reports assertion with expected / actual values in a custom format.
	
	These assertions use the C-standard snprintf() function to convert expected / actual values to their string representation using the custom pattern (<format> string).
	
	Additional notes:
	
	• The C-standard snprintf() function needs to be available for C++test's Test Harness.
	• The format string (<format>) should be compliant with the C-standard printf() function.
	• <expected>, <actual> and <delta> values need to be specified according to their real types, in a format accepted / expected by the printf() function - e.g. when the format for floating point values string is "%f", valid float values should be specified (for example, 0.0, not 0).
	• There is a buffer size defined for the resulting value string. If the actual value string size exceeds the buffer size, an execution error will be reported.
	• The default size of the output buffer is set to 24 bytes. The size of the buffer can be customized by adding the following option to Project Properties> C++test> Build Settings> Compiler Options: -DCPPTEST_FMT_BUFFER_SIZE=<new buffer size in bytes>.