Assertion Playground

From TinyOS Wiki

This lesson assumes you have already created your MyFirstTest Test Suite by following the Your First Test lesson.

Let's try out some different assertions by modifying the MyFirstTestP.nc file. Remember to refer back to TUnit Assertions to see what types of assertions can be made.

Contents

- 1 assertFail()
- 2 assertEquals(), assertNotEquals(), assertTrue(), assertFalse()
  - 2.1 Failure in assertEquals()
  - 2.2 Failure in assertTrue()
  - 2.3 Failure in assertNotEquals()
- 3 assertResultIsAbove(), assertResultIsBelow()
  - 3.1 Failure in assertResultIsBelow()
  - 3.2 Failure in assertResultIsAbove()
- 4 assertNull(), assertNotNull()
  - 4.1 assertNull() Failure
  - 4.2 assertNotNull() Failure
- 5 assertCompares()
  - 5.1 assertCompares() Failure
- 6 Too many assertions!
  - 6.1 suite.properties File
- 7 See Also
- 8 Next

assertFail()

We'll simply change our *assertSuccess()* into *assertFail(<fail msg>)*:

/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestP.nc

```
 0|  #include "TestCase.h"
 1|
 2|  module MyFirstTestP {
 3|    uses {
 4|      interface TestCase as BasicAssertionTest;
 5|    }
 6|  }
 7|
 8|  implementation {
 9|
10|    event void BasicAssertionTest.run() {
11|      assertFail("Failure!");
12|      call BasicAssertionTest.done();
13|    }
14|
15|  }
```

Every test failure requires you to give a textual reason why the test failed. In this case, we'll print out the string "Failure!" because our test is now failing at line 11. The *assertFail(<fail msg>)* is the equivalent of saying "If we reach this line of code, our test failed."

```
TUnit Tip
These microcontrollers don't have a lot of memory.
Although longer messages are ok (requiring several packets to be delivered to
the computer to report the whole message), keep your failure descriptions short.
Succinct messages are less likely to be truncated.
```

Now let's run TUnit...

```
$ tunit

...

T-Unit Results
----------------------------------------------
Total runtime: 41.922 [s]
Total tests recorded: 2
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC
        at MyFirstTestP.nc, line 11:
Failure!
```

assertEquals(), assertNotEquals(), assertTrue(), assertFalse()

We can verify two values are the same. Notice in this test, we'll make multiple assertions. TUnit, by default, has a queue for 5 assertion messages to get across to the computer in a single Test Case. This queue is flushed completely between tests or between task posts. The suite.properties file allows you to increase the number of assertions each Test Case is allowed to make, which we'll show in a minute.

```
/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestP.nc

  0|  #include "TestCase.h"
  1|
  2|  module MyFirstTestP {
  3|    uses {
  4|      interface TestCase as BasicAssertionTest;
  5|    }
  6|  }
  7|
  8|  implementation {
  9|
 10|    event void BasicAssertionTest.run() {
 11|      assertEquals("uint16_t isn't 2 bytes", 2, sizeof(uint32_t));
 12|      assertTrue("False!", TRUE);
 13|      assertFalse("True!", FALSE);
 14|      assertNotEquals("Values rounded off!", (float) 4.59, (int) 4.59);
 15|      call BasicAssertionTest.done();
 16|    }
 17|
 18|  }
```

Notice that on line 11 and 14, the value you *expect* is the first argument, and the *result* you actually obtained is the second. This format applies to every other unit testing framework you'll use. Running TUnit, all our tests pass.

```
T-Unit Results
----------------------------------------------
Total runtime: 42.441 [s]
Total tests recorded: 5
```

```
Total errors: 0
Total failures: 0
```

Failure in assertEquals()

Changing line 11 to *assertEquals("uint16_t isn't 2 bytes", 2, sizeof(uint32_t));*, we see some failure output that helps us locate the problem:

```
T-Unit Results
----------------------------------------------
Total runtime: 43.127 [s]
Total tests recorded: 5
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest): uint16_t isn't 2 bytes; Expecte
d [2] but got [4] (unsigned 32-bit form)
```

The comment about the unsigned 32-bit form is there to help you remember that any numbers displayed are all converted to uint32_t's. If we were to change line 11 to *assertEquals("Wrong number", -2, -3);*, then we'd see some ridiculous output due to the interpretation of the sign:

```
T-Unit Results
----------------------------------------------
Total runtime: 42.361 [s]
Total tests recorded: 5
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
     at MyFirstTestP.nc, line 11:
Wrong number; Expected [4294967294] but got [4294967293] (unsigned 32-bit form)
```

If you were to manually convert those numbers back to a signed number, you'd see they're still -2 and -3.

Failure in assertTrue()

If we were to change line 12 to *assertTrue("False!", FALSE);*, then we'd get...

```
T-Unit Results
----------------------------------------------
Total runtime: 42.066 [s]
Total tests recorded: 5
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 11:
False!
```

Failure in assertNotEquals()

If we changed line 14 to *assertNotEquals("Values rounded off!", (float) **4.0**, (int) 4.59);*, then the values both round off to 4 and become equal, which fails our assertion:

```
T-Unit Results
----------------------------------------------
Total runtime: 42.018 [s]
Total tests recorded: 5
Total errors: 0
Total failures: 1
```

```
MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
        at MyFirstTestP.nc, line 11:
Values rounded off!; Shouldn't have gotten [4] (unsigned 32-bit form)
```

Notice, though, that the types of arguments being compared doesn't matter. Here we're comparing a float and an int, and it's possible to compare any other type combination. For arrays or structures, we have to use assertCompares().

assertResultIsAbove(), assertResultIsBelow()

The assertResultIsAbove() and assertResultIsBelow() are only > and < tests; they are not >= or <=. These are typically used to set a threshold of minimum performance before the test is considered a failure.

This test will make sure the "*myResult*" variable is above 5000 and below 7000.

/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestP.nc

```
 0|  #include "TestCase.h"
 1|
 2|  module MyFirstTestP {
 3|    uses {
 4|      interface TestCase as BasicAssertionTest;
 5|    }
 6|  }
 7|
 8|  implementation {
 9|
10|    event void BasicAssertionTest.run() {
11|      uint16_t myResult = 6000;
12|      assertResultIsAbove("Too low", 5000, myResult);
13|      assertResultIsBelow("Too high", 7000, myResult);
14|      call BasicAssertionTest.done();
15|    }
16|
17|  }
```

Running the test like this, everything passes:

```
T-Unit Results
----------------------------------------------
Total runtime: 42.002 [s]
Total tests recorded: 3
Total errors: 0
Total failures: 0
```

Failure in assertResultIsBelow()

If we change "*myResult*" to be 7000 or above, then we get a failure (7000, 7001, etc. is not less than 7000):

```
T-Unit Results
----------------------------------------------
Total runtime: 41.94 [s]
Total tests recorded: 3
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 12:
Too high; Actual result [7000] was not below [7000] (unsigned 32-bit form)
```

Failure in assertResultIsAbove()

Or change "*myResult*" to 5000 or below, and we get:

```
T-Unit Results
----------------------------------------------
Total runtime: 41.846 [s]
Total tests recorded: 3
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 13:
Too low; Actual result [4999] was not above [5000] (unsigned 32-bit form)
```

assertNull(), assertNotNull()

/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestP.nc

```
 0|  #include "TestCase.h"
 1|
 2|  module MyFirstTestP {
 3|    uses {
 4|      interface TestCase as BasicAssertionTest;
 5|    }
 6|  }
 7|
 8|  implementation {
 9|
10|    event void BasicAssertionTest.run() {
11|      uint16_t value;
12|      uint16_t *ptr;
13|      assertNull(ptr);
14|
15|      ptr = &value;
16|      assertNotNull(ptr);
17|      call BasicAssertionTest.done();
18|    }
19|
20|  }
```

This test passes. Notice that the assertNull() and assertNotNull() assertions do not take failure messages as arguments.

assertNull() Failure

Now let's switch things up to make the pointer point to something throughout the entire test. This will cause assertNull() to fail, because the pointer is no longer null.

```
 ...

10|    event void BasicAssertionTest.run() {
11|      uint16_t value;
12|      uint16_t *ptr = &value;
13|      assertNull(ptr);
14|
15|      assertNotNull(ptr);
16|      call BasicAssertionTest.done();
17|    }
```

This results in a failure, telling you which pointer was *not* null when it should have been null:

```
T-Unit Results
----------------------------------------------
Total runtime: 43.031 [s]
Total tests recorded: 3
Total errors: 0
Total failures: 1
```

```
MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 13:
ptr was not null.
```

assertNotNull() Failure

When the pointer is left null throughout the entire test, then the assertNotNull() fails:

```
 ...

 10|    event void BasicAssertionTest.run() {
 11|      uint16_t value;
 12|      uint16_t *ptr;
 13|      assertNull(ptr);
 14|
 15|      assertNotNull(ptr);
 16|      call BasicAssertionTest.done();
 17|    }
```

```
T-Unit Results
----------------------------------------------
Total runtime: 42.125 [s]
Total tests recorded: 3
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 15:
ptr was null
```

assertCompares()

The assertCompares() assertion is used to compare arrays, structures, etc. Below is a test that will pass assertCompares(), comparing two buffers.

```
/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestP.nc

  0|  #include "TestCase.h"
  1|
  2|  module MyFirstTestP {
  3|    uses {
  4|      interface TestCase as BasicAssertionTest;
  5|    }
  6|  }
  7|
  8|  implementation {
  9|
 10|    enum {
 11|      BUFFER_SIZE = 5,
 12|    };
 13|
 14|    event void BasicAssertionTest.run() {
 15|      uint8_t buffer1[BUFFER_SIZE];
 16|      uint8_t buffer2[BUFFER_SIZE];
 17|
 18|      memset(buffer1, 0xAA, sizeof(buffer1));
 19|      memset(buffer2, 0xAA, sizeof(buffer2));
 20|
 21|      assertCompares("Buffers aren't identical", buffer1, buffer2, BUFFER_SIZE);
 22|
 23|      call BasicAssertionTest.done();
 24|    }
 25|
 26|  }
```

```
T-Unit Results
----------------------------------------------
```

```
Total runtime: 43.143 [s]
Total tests recorded: 2
Total errors: 0
Total failures: 0
```

assertCompares() Failure

If we change line 18 to *memset(buffer1, **0xBB**, sizeof(buffer1));*, then our buffers won't compare and we get a failure:

```
T-Unit Results
----------------------------------------------
Total runtime: 42.111 [s]
Total tests recorded: 2
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 16:
Buffers aren't identical; Expected [0] but got [17] (unsigned 32-bit form)
```

The expectation here is that memcmp() run by assertCompares() returns 0, indicating both buffers are identical. The 17 is the result of the memcmp() on the buffers, which shows the buffers are not equal.

Too many assertions!

By default there is a queue of 5 assertion messages that can get through to the computer. Each time an assertion is made, one of the messages is eaten up. If an assertion fails and requires a multi-packet failure message, then multiple packets from the queue get eaten up. Until a task is posted or the test is complete, messages will not be sent from the queue.

Let's see what happens when our test makes too many assertions...

/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestP.nc

```
 0|  #include "TestCase.h"
 1|
 2|  module MyFirstTestP {
 3|    uses {
 4|      interface TestCase as BasicAssertionTest;
 5|    }
 6|  }
 7|
 8|  implementation {
 9|
10|    event void BasicAssertionTest.run() {
11|      assertFail("This is a really long failure message that spans multiple messages and eats up the messag
12|      assertFail("Short msg");
13|      assertSuccess();
14|      call BasicAssertionTest.done();
15|    }
16|
17|  }
```

And the result is...

```
T-Unit Results
----------------------------------------------
Total runtime: 42.285 [s]
Total tests recorded: 2
Total errors: 0
Total failures: 1

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
```

```
      at MyFirstTestP.nc, line 11:
This is a really long failure message that spans multiple messages and eats up t
```

Notice only 2 tests were recorded. Of those, one of them was the test to see if we could compile. Our test made three assertions, but only the first one got through. Rude, but repairable. We'll need to access the suite.properties file to increase the number of elements in the TUnit queue.

suite.properties File

Create (or copy from another directory) a file called "suite.properties". This file contains all the rules TUnit should use when running the test in the current directory or sub-directories. Read the suite.properties file documentation for all the rules you can include.

```
/opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest/suite.properties

0|  @assertions 20
1|
```

That's it!

```
TUnit Tip
Each extra assertion in TUnit's embedded side queue takes up memory.
Keep the assertion queue low, and your test won't crash on smaller platforms.
```

Now let's run the test above with the suite.properties addition...

```
T-Unit Results
----------------------------------------------
Total runtime: 41.92 [s]
Total tests recorded: 4
Total errors: 0
Total failures: 2

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 11:
This is a really long failure message that spans multiple messages and eats up the message queue.

MyFirstTestC.BasicAssertionTestC (EmbeddedTest):
      at MyFirstTestP.nc, line 12:
Short msg
```

The total of 4 tests came from the 1 compile test, plus the 3 assertions made on the embedded side. We also see the full long message printed out and the short failure message afterward.

See Also

- TUnit
- Single-Node Unit Testing
- Your First Test

Next

- State Interface Test

Retrieved from "http://tinyos.stanford.edu/tinyos-wiki/index.php?title=Assertion_Playground&oldid=582"

- This page was last modified on 16 January 2008, at 07:59.
- This page has been accessed 26,440 times.