Your First Test

From TinyOS Wiki

Contents

Test Directory

Let's create a Test Suite (http://en.wikipedia.org/wiki/Test_suite) with a few assertions (http://docs.tinyos.net/index.php/TUnit_Assertions) in it to see what TUnit can do. This test won't really be testing anything, it's just going to make some assertions and exit.

First, create a directory to store this test. This directory can go anywhere, but I'll put it in *tinyos-2.x-contrib/tunit/tests/MyFirstTest* for now:

```
tinyos-2.x-contrib
|-- tunit
|   |-- tests
|   |   |-- MyFirstTest
```

MyFirstTestC Configuration File

Create a configuration file, called *MyFirstTestC.nc*, and start writing it like you would any other TinyOS application. This will be the main entry point for your program.

tinyos-2.x-contrib/tunit/tests/MyFirstTest/MyFirstTestC.nc

```
  0|  configuration MyFirstTestC {
  1|  }
  2|
  3|  implementation {
  4|
  5|    components new TestCaseC() as BasicAssertionTestC;
  6|
  7|    components MyFirstTestP;
  8|
  9|    MyFirstTestP.BasicAssertionTest -> BasicAssertionTestC;
 10|
 11|  }
```

On line 5, we create a new instance of **TestCaseC**. This is saying "Add a new test to our suite". Notice we immediately rename all instances of TestCaseC's as *SomeReallyLongAndDescriptiveTestNameC*. In this case, we called it *BasicAssertionTestC*.

```
TUnit Tip
It's important to rename your TestCaseC instances with descriptive names.
When that test fails, TUnit will magically tell you the name of the offending
test so you can go look it up.
```

Notice that we don't include a MainC component anywhere in our test configuration. Your test should not use Boot.booted(), Init.init(), etc. because your test is actually wrapped inside of the TUnit framework. Look ahead to the TestControl interfaces (*SetUpOneTime*, *SetUp*, *TearDown*, *TearDownOneTime*) for more information on how to do those things properly.

If you're worried about what the TestCaseC (http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x-contrib/tunit/tos/lib/tunit/TestCaseC.nc?content-type=text%2Fplain) component actually refers to, it comes from the TUnit embedded libraries. The library is found in *tinyos-2.x-contrib/tunit/tos*. TUnit will take care of referencing this entire library for you at compile time so you don't have to think about it.

The signature of the TestCaseC component looks like this:

```
generic configuration TestCaseC() {
  provides {
    interface TestCase;
    interface TestControl as SetUpOneTime @atmostonce();
    interface TestControl as SetUp @atmostonce();
    interface TestControl as TearDown @atmostonce();
    interface TestControl as TearDownOneTime @atmostonce();
  }
}
```

The interface we're interested in to run our basic assertion test is the TestCase interface. The TestCase interface is defined in tinyos-2.x-contrib/tunit/tos/interfaces/TestCase.nc (http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x-contrib/tunit/tos/interfaces/TestCase.nc?content-type=text%2Fplain) :

```
interface TestCase {

  event void run();

  async command void done();

}
```

Pretty simple interface, right? Your TestCase signals you to run(), and you **must** call back when your test is done(). With that in mind, we have a module to build...

MyFirstTestP Module File

As the configuration file above dictated, we need to have a module called MyFirstTestP that *uses* the interface *TestCase as BasicAssertionTest*. Let's create the MyFirstTestP module now:

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

```
#include "TestCase.h"

module MyFirstTestP {
  uses {
    interface TestCase as BasicAssertionTest;
  }
}

implementation {

  event void BasicAssertionTest.run() {
    assertSuccess();
12|      call BasicAssertionTest.done();
13|    }
14|
15|  }
```

There are a couple things to point out here.

Line 0 is the all-important *TestCase.h* import. This header file allows TUnit to make assertions. The TestCase.h (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tos/lib/tunit/TestCase.h?view=markup) file is mixed in with the TUnit embedded library. Don't think about it, just import it and refer back to the list of assertions (http://docs.tinyos.net/index.php/TUnit_Assertions) when necessary.

Line 10 is the entry point for our first test. TUnit will tell each test when to run(), and your responsibility is to tell TUnit when that test is done(). TinyOS is *split-phase*, so there's no way for TUnit to know when your test is actually done without you explicitly saying so.

Inside the test, line 11 makes an assertion. In this case, we *assertSuccess()*, which is the equivalent of saying "If the test made it this far in the code, the test was a success."

Each Test Case must make at least one assertion. If a test doesn't make any assertions, TUnit will give you a warning. It's possible your test failed to execute correctly and test what it was supposed to test.

Line 12 is very important. I cannot stress this enough: when your test is done running, you **must** tell TUnit that the test is done(). This deserves a huge tip comment on it's own:

```
TUnit Tip
TUnit's tells your tests when to run().
After receiving the signal to run() and your test is completes,
you MUST call back to TUnit telling it your test is done()!
```

If we didn't *call BasicAssertionTest.done()* on line 12, TUnit would have timed out after a default of 1 minute (expandable using suite.properties rules, described later). This timeout would normally occur if the test locked up your microcontroller, allowing the TUnit framework on the PC side to continue safely executing other tests.

```
TUnit Tip
If TUnit times out for apparently no reason,
you probably forgot issue a done() command somewhere.
```

Makefile

Just like every other TinyOS application, this one needs a Makefile so it can be compiled:

```
tinyos-2.x-contrib/tunit/tests/MyFirstTest/Makefile

COMPONENT=MyFirstTestC
include $(MAKERULES)
```

Notice we don't try to include anything in the Makefile except the name of the entry test configuration, and the makerules reference. Don't try to include compile flags or anything else in the Makefile except these two lines. The suite.properties file is the place to put compiler options, and is discussed later.

Also, TUnit is responsible for pulling in the TUnit embedded libraries, so don't worry about telling the compiler where they are located.

At this point, your directory structure should look like this:

```
tinyos-2.x-contrib
|-- tunit
|   |-- tests
|   |   |-- MyFirstTest
|   |   |   |-- Makefile
|   |   |   |-- MyFirstTestC.nc
|   |   |   |-- MyFirstTestP.nc
```

Run TUnit

Go to your command line and drill down to the MyFirstTest directory. Make sure you have your motes plugged in and the tunit.xml file configured correctly (see Setting up TUnit), and run the TUnit Java application (which I have aliased "tunit"). Here is a truncated version of what I see when I run the test:

```
$ cd /opt/tinyos-2.x-contrib/tunit/tests/MyFirstTest
$ tunit
0 [main] INFO com.rincon.tunit.TUnit  - Base package directory located: C:\
15 [main] INFO com.rincon.tunit.TUnit  - Found a TOSCONTRIB environment variable

140 [main] INFO com.rincon.tunit.TUnit  - STARTING TEST RUN TELOSB

Now we're actually running our first Test Run, check if the motes exist.
If the motes in this test run were found, find a test to compile.
When a test is found in your current directory or a sub-directory, compile it
and install it to the node attached to your computer.

    compiling MyFirstTestC to a telosb binary
ncc -o build/telosb/main.exe -Os -O -mdisable-hwmul -Wall -Wshadow -DDEF_TOS_AM_
GROUP=0x7d -Wnesc-all -target=telosb -fnesc-cfile=build/telosb/app.c -board= -Ic
:/tinyos/cygwin/opt_svn/tinyos-2.x-contrib/tunit/tos/lib/tunit -Ic:/tinyos/cygwi
n/opt_svn/tinyos-2.x-contrib/tunit/tos/lib/tunitstats -Ic:/tinyos/cygwin/opt_svn
/tinyos-2.x-contrib/tunit/tos/system -Ic:/tinyos/cygwin/opt_svn/tinyos-2.x-contr
ib/tunit/tos/interfaces -Ic:/tinyos/cygwin/opt_svn/tinyos-2.x-contrib/tunit/tos/
lib/directserial -Ic:/tinyos/cygwin/opt_svn/tinyos-2.x-contrib/tunit/tos/lib/fif
oqueue -DTUNIT_TOTAL_NODES=1  MyFirstTestC.nc -lm
    compiled MyFirstTestC to build/telosb/main.exe
            7200 bytes in ROM
             471 bytes in RAM

After installing the application to the node, make sure we can communicate
with the mote over the serial forwarder by sending a ping.
If a pong is received (hidden), kick off the test.

38167 [Thread-19] INFO com.rincon.tunit.run.ResultCollector  -
Test 0 (MyFirstTestC.BasicAssertionTestC)
        at MyFirstTestP.nc, line 11:
  PASSED

The test is complete, clean everything up and find the next test to run.
In this case, there are no more tests to run, so we exit and print all the results
at the end.


T-Unit Results
-----------------------------------------------
Total runtime: 44.974 [s]
Total tests recorded: 2
Total errors: 0
Total failures: 0
```

Notice that there was only a single Test Case making a single assertion, but 2 tests were recorded. This is because the compilation process is a test itself. Because we compiled successfully, that test passed and increases our total tests recorded.

See Also

- TUnit
- Setting up TUnit
- Single-Node Unit Testing

Next

- Assertion Playground