TUnit Test Flow

From TinyOS Wiki

The Java side of TUnit is responsible for finding tests to execute, compiling them, installing them to the nodes, and collecting results.

The only mandatory interface to implement is the TestCase interface. Multiple Test Cases may exist as part of your Test Suite. A Test Suite forms a single, compilable test application in TinyOS.

On the embedded side, tests follow the standard xUnit (http://en.wikipedia.org/wiki/XUnit) flow.

Contents

SetUpOneTime.run()

SetUpOneTime is an optional TestControl interface, which run()'s one time for every node under test. This can be thought of like a Boot.booted() event.

SetUpOneTime is run immediately for each Supporting Node once the application has been installed. This could, for example, allow the Supporting Node to turn on its radio, begin transmitting a barrage of packets, etc. SetUpOneTime is run at the Driving Node only when given the command by the computer to begin testing, after all other nodes under test are running.

If you implement the SetUpOneTime interface in your test, you **MUST** call SetUpOneTime.done() when your Test Suite is done setting up.

```
command void SetUpOneTime.run() {
  call SetUpOneTime.done();
}
```

SetUp.run()

SetUp is an optional TestControl interface, only run() by the Driving Node. It is run() once before each TestCase. This function may be used to reset states, clear out variables, etc.

If you implement the SetUp interface in your test, you **MUST** call SetUp.done() when your TestCase is done setting up.

```
command void SetUp.run() {
  call SetUp.done();
}
```

TestCase.run()

Each TestCase interface is only executed at the Driving Node. Although the Driving Node is responsible for starting each test, Supporting Nodes may make assertions for the test and also end the test by calling TestCase.done(). Supporting Nodes are never notified (except explicitly by the Driving Node) when a test has started.

You **MUST** call TestCase.done() when your test is complete.

```
command void TestCase.run() {
  assertTrue("False!", TRUE);
  call TestCase.done();
}
```

TearDown.run()

TearDown is an optional TestControl interface is executed each time after every test. This allows you to clean up a test before moving on to the next, but is somewhat redundant to SetUp.

If you implement the TearDown interface in your test, you **MUST** call TearDown.done() when your TestCase is done tearing down.

```
command void TearDown.run() {
  call TearDown.done();
}
```

TearDownOneTime.run()

TearDownOneTime is an optional TestControl interface, run at every node when all tests are complete. If you turned on a radio for your test, the polite thing to do is to turn the radio off so future tests are not affected by a active radio lingering nearby.

If you implement the TearDownOneTime interface in your test, you **MUST** call TearDown.done() when your TestCase is done tearing down.

```
command void TearDownOneTime.run() {
  call TearDownOneTime.done();
}
```

System Sequence Diagrams

This is the general system flow for executing a single test. Note that the TUnit Java side knows how many tests to expect, because it parses through the app.c file. When your test reports some results, the TUnit Java application knows the name of the test that generated those results - directly from your source code!

See Also

- TUnit
- Setting up TUnit
- Flow of TUnit Java Execution

Next

- TUnit Assertions