Information about using NodeUnit for testing the CPSC301 Course Project

Charles DeBavelaere – cdebavel@ucalgary.ca

Building tests:

For a quick example of how to build tests, refer to ./tests/examples/examples.js.
Basic tests can be implemented as follows:

```
exports.<test name> = function(test) {
   test.expect(<number of expected Assertions>);
   test.ok(<value to test, should equal 'true'>, "<message to show if this fails>");
   test.equal(<actual value>, <expected value>, "<message to show if this fails>");
   test.done();    // All assertions/tests for this particular test are done
};
```

In addition to test.ok and test.equal, there are other possible assertions (list from https://github.com/caolan/nodeunit):

ok(value, [message]) - Tests if value is a true value.

equal(actual, expected, [message]) - Tests shallow, coercive equality with the equal comparison operator (==).

notEqual(actual, expected, [message]) - Tests shallow, coercive non-equality with the not equal comparison operator (!=).

deepEqual(actual, expected, [message]) - Tests for deep equality.

notDeepEqual(actual, expected, [message]) - Tests for any deep inequality.

strictEqual(actual, expected, [message]) - Tests strict equality, as determined by the strict equality operator (===)

notStrictEqual(actual, expected, [message]) - Tests strict non-equality, as determined by the strict not equal operator (!==)

throws(block, [error], [message]) - Expects block to throw an error.

doesNotThrow(block, [error], [message]) - Expects block not to throw an error.

ifError(value) - Tests if value is not a false value, throws if it is a true value. Useful when testing the first argument, error in callbacks.

expect(amount) - Specify how many assertions are expected to run within a test. Very useful for ensuring that all your callbacks and assertions are run. [However, this is OPTIONAL]

done() - Finish the current test function, and move on to the next. ALL tests should call this! **[This is REQUIRED]**

Tests may also be implemented as suites, which is demonstrated in the previously-mentioned examples.js.

Steps for adding new tests to the project:

1) Create an appropriately-named folder in the ./tests/ directory
   For this example, let's call it *tests*
2) Create an appropriately-named javascript file in that folder.
   E.g., *test.js*
3) Create an appropriately-named script file in that folder
   E.g., *test.sh*
4) Changed the script's permissions
   E.g., *chmod 755 test.sh*
5) Set up the script to execute your tests
   E.g.,
   ```
   #!/bin/sh
   node ../nodeunit/bin/nodeunit test.js
   ```
6) Set up your tests (use examples.js as a template)
7) Add your tests to the ./tests/alltests.sh script
   E.g., node ./nodeunit/bin/nodeunit tests
   OR node ./nodeunit/bin/nodeunit tests/*.js
   OR node ./nodeunit/bin/nodeunit tests/test.js

To run tests:

For ALL TESTS:

```
>$ cd ./tests
>$ ./alltests.sh
```

For SPECIFIC TESTS:

```
>$ cd ./tests/<mytests>
>$ ./<myscript>.sh
```

For a more in-depth example of writing tests for NodeUnit, refer to ./tests/dbtest/dbtest_nodeunit.js

If you have any questions about testing with NodeUnit, you can email me at cdebavel@ucalgary.ca