Unit Testing Framework

# Goals

Provide a framework for unit testing to:

* Standardize how unit tests are developed (Look and Feel). Makes unit tests more consistent, reusable, maintainable, and understandable.
* Provide the necessary tools and guidance so that developers do not waste time creating their own tools and instead focuses on writing unit tests.
* Make unit testing easier.
* Support automated regression testing. This will become more important as the FSSB enhances its reusable code base. In a layered architecture it is important to determine whether an API change has impacted existing code.

# Reference Documents

*Flight Software Branch Unit Test Standard - 582-2000-002*

The importance of this document is that it defines a software unit and a unit test.

# Introduction or Concepts or Definition Section

Here are the definitions from the unit test standard:

A unit is defined as single interface file, together with the body files that implement the unit. In C, the interface file is a header file. In Ada, it is a package specification.

A unit test is defined to be a test that executes all of the code in a unit, with the unit not embedded in a larger system. The unit test proves that the unit executes correctly. Unit tests are always white-box; they take advantage of knowledge of how the unit is constructed. Unit tests may consist of more than one executable, if that makes them more understandable. The unit source code, the unit test input, the unit test source code, and unit test results must together show the unit source code is properly implemented and free of implementation errors.

(Maybe this belongs in a users guide????)

All unit tests should be kept simple and if possible and test at most one behavior (this is a requirement on the unit test writer and I think it should go in the introduction section or maybe as notes in the UT standard if not already there).

* Keeping the unit tests small and simple make them easier to understand and maintain.
* Each TestCase should consist of multiple test methods (functions) with an easy way to execute all test methods. (i.e. RunTest method)
* As a matter of style sufficient comments should be added to the test to make its purpose obvious.

# Requirements

All unit tests shall be able to be run by a single command.

* This could be accomplished by a using single executable or automated by using a makefile.

All unit tests shall be self verifying.

* A standard set of asserts will be used to determine the pass/fail status.
* All asserts shall be derived from a single Boolean base assert (AssertTrue).
* All asserts shall return a Boolean value that indicates the pass/fail status of the assert. This is intended to be used to conditionally include or skip additional asserts in the unit test. For example if AssertPacketSent returns FALSE (i.e. the expected packet was not sent) then the unit test code could skip additional asserts that verify the contents of the packet.
* Custom user defined asserts could also be used to handle cases where the standard set of asserts is not sufficient. Custom asserts save test coding effort and make the code more readable. All custom asserts should follow the rules stated above.

All unit tests shall output a pass/fail status.

* A pass/fail status should be output for each assert (Maybe consider an option to only output messages on failures).
* A pass/fail status summary should be output at the completion of the unit test that details the total number of tests executed (maybe) and test failures.
* The unit test runner executable should return a return code that indicates the pass/fail status, i.e. the test passed if all test failures == 0. This is intended to be used if the unit test is run from a makefile.

It shall be easy to determine where unit test failures have occurred.

* Each assert should output enough information to determine where test failures occurred. This could be the line number and file and or the name of the test function and which assert failed.
* Additional information could be output such as expected and received values using something like sprintf.
* Additionally the name of the unit test could be output at the start of each unit test.

All unit tests should be independent of other unit tests.

* Well written unit tests exhibit isolation.
* The entire test environment needs to be reset to a default state prior to running each unit test.
* Support setup and teardown functions to initialize the test environment before and after each unit test.

All unit tests shall be ANSI C compliant

Unit tests shall be organized in a hierarchical manner where it is possible to combine unit tests from different sources together into higher level test runners.

* TestSuites could be used to group multiple Test Cases together.

All unit tests shall send their output to either standard output or a log file (selectable by a switch).

* Maybe consider an additional optional switch to select the level of output, i.e. output everything or just output assert failures and a summary at the end.
* The test summary should always be output to standard output. So if the switch is set to send all output to a log file the pass fail status of the test could be determined immediately without the need to open and examine the log file.

All unit tests shall follow a standard naming convention and a standard format (i.e. Include a TestRunner, TestSuites, and TestCases).

* We need to provide documentation and templates.
* I guess code reviews should be responsible for enforcing the standard.

There should be a clean separation between the test and flight code (i.e. all of the code should not be lumped into a single directory).

* Could implement separate identical directory trees, one for src and one for test.
* Could add a test directory directly below each tasks src directory.

The platform (or environment) dependencies shall be hierarchical so only the modules (outside of basic UT driver) required by the FSW component being tested are also required by the unit test.

* Additional modules from a library can be optionally added based on the developers needs.

All unit tests shall be capable of running on the target hardware.

* This one may be more difficult than it sounds.

Consider using Doxygen to generate user api documentation.