Nightscout Testing Framework

Team 2-2: Jack Fraser, Jesse Deacon, Swapnil Srivastava

Introduction

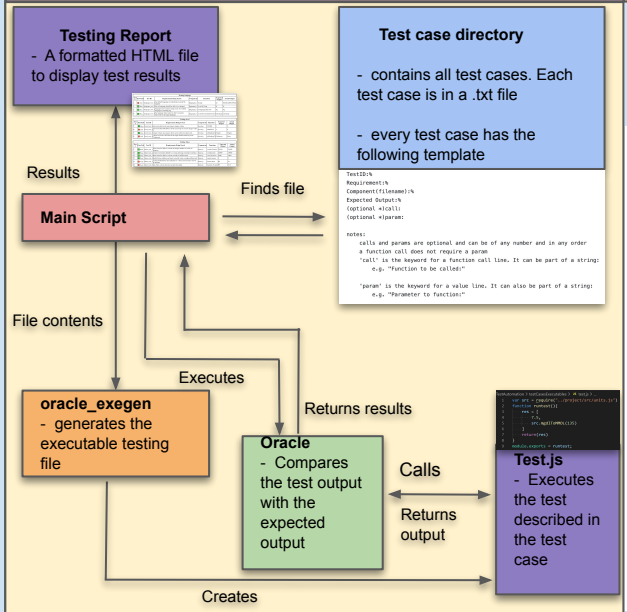
Introduction

Unit testing is the process of testing the functions, methods, and objects within a given project by comparing the accuracy of individual function calls against their target values. It is exceptionally useful because, while unit tests alone do not prove that a software is bug-free, they can be used to ensure that correctness of core functions has been achieved for a number of predictable scenarios, and can be run after any modification, ensuring that a change to the code does not accidentally introduce new errors without being noticed. In some software development schedules, unit tests might be designed before the functions are, so that tests can be run on even partially completed code to see how that code is progressing towards completion. This has the advantage of emphasizing efficient code that achieves clear project goals & requirements; it also has the potential disadvantage of making programmers write code to pass the tests, rather than write holistically good code. Nonetheless, when used correctly, unit testing is a tool to improve a software's reliability and robustness, and as such is an excellent addition to any acceptance testing process.

High-level Description

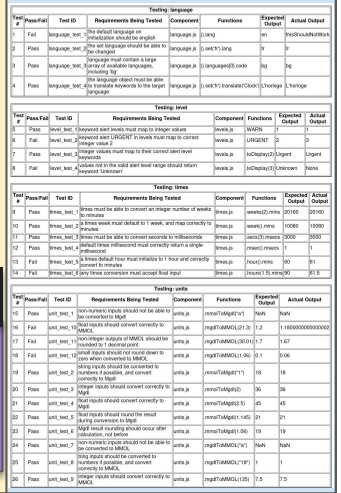
The 2-2 Testing Framework is designed utilizing `python` as the main scripting language, with `javascript` as the actual execution language. Node and subprocess calls provide the method for the two to communicate.

Testing



Report

Results are Tabulated into an automatically-opened HTML file. They are formatted and grouped by source, so that each test can be viewed in the context of its peers.



Testing: language

Test #	Pass/Fail	Test ID	Requirements Being Tested	Component	Functions	Expected Output	Actual Output
1	Fail	language_test_1	the default language on initialization should be english	language.js	().lang	en	thisShouldNotWork
2	Pass	language_test_2	the set language should be able to be changed	language.js	().set('fr').lang	fr	fr
3	Pass	language_test_3	language must contain a large variety of available languages, including 'bg'	language.js	().languages()['zxxkx	bg	bg
4	Pass	language_test_4	the language object must be able to translate keywords to the target language	language.js	().set('fr').translate('Clock')	l'horloge	l'horloge

Testing: level

Test #	Pass/Fail	Test ID	Requirements Being Tested	Component	Functions	Expected Output	Actual Output
5	Pass	level_test_1	keyword alert levels must map to integer values	levels.js	.WARN	1	1
6	Fail	level_test_2	keyword alert URGENT in levels must map to correct integer value 2	levels.js	.URGENT	2	3
7	Pass	level_test_3	integer values must map to their correct alert level keywords	levels.js	.toDisplay(2)	Urgent	Urgent
8	Fail	level_test_4	values not in the valid alert level range should return keyword 'Unknown'	levels.js	.toDisplay(3)	Unknown	None

Testing: times

Test #	Pass/Fail	Test ID	Requirements Being Tested	Component	Functions	Expected Output	Actual Output
9	Pass	times_test_1	times must be able to convert an integer number of weeks to minutes	times.js	.weeks(2).mins	20160	20160
10	Pass	times_test_2	a times week must default to 1 week, and map correctly to minutes	times.js	.week().mins	10080	10080
11	Pass	times_test_3	times must be able to convert seconds to milliseconds	times.js	.secs(3).msecs	3000	3000
12	Pass	times_test_4	default times milliseconds must correctly return a single millisecond	times.js	.msec().msecs	1	1
13	Fail	times_test_5	a times default hour must initialize to 1 hour and correctly convert to minutes	times.js	.hour().mins	60	61
14	Fail	times_test_6	any times conversion must accept float input	times.js	.hours(1.5).mins	90	91.5

Testing: units

Test #	Pass/Fail	Test ID	Requirements Being Tested	Component	Functions	Expected Output	Actual Output
15	Pass	unit_test_1	non-numeric inputs should not be able to be converted to Mgdl	units.js	.mmolToMgdl("s")	NaN	NaN
16	Fail	unit_test_10	float inputs should convert correctly to MMOL	units.js	.mgdlToMMOL(21.2)	1.2	1.1666000000000002
17	Fail	unit_test_11	non-integer outputs of MMOL should be rounded to 1 decimal point	units.js	.mgdlToMMOL(58.01)	1.7	1.67
18	Fail	unit_test_12	small inputs should not round down to zero when converted to MMOL	units.js	.mgdlToMMOL(1.06)	0.1	0.06
19	Pass	unit_test_2	string inputs should be converted to numbers if possible, and convert correctly to Mgdl	units.js	.mmolToMgdl("1")	18	18
20	Pass	unit_test_3	integer inputs should convert correctly to Mgdl	units.js	.mmolToMgdl(2)	36	36
21	Pass	unit_test_4	float inputs should convert correctly to Mgdl	units.js	.mmolToMgdl(2.5)	45	45
22	Pass	unit_test_5	float inputs should round the result during conversion to Mgdl	units.js	.mmolToMgdl(1.140)	21	21
23	Pass	unit_test_6	Mgdl result rounding should occur after calculation, not before	units.js	.mmolToMgdl(1.06)	19	19
24	Pass	unit_test_7	non-numeric inputs should not be able to be converted to MMOL	units.js	.mgdlToMMOL("s")	NaN	NaN
25	Pass	unit_test_8	string inputs should be converted to numbers if possible, and convert correctly to MMOL	units.js	.mgdlToMMOL("18")	1	1
26	Pass	unit_test_9	integer inputs should convert correctly to MMOL	units.js	.mgdlToMMOL(135)	7.5	7.5

Details

Test Plan

Testing process
1. Find Testable Method
2. Import method to the TestAutomation/project/src/ directory
3. Modify method as needed to make it able to interact directly with the test environment.
4. Determine valid / invalid inputs for method
5. Write tests for method
6. Write test cases for tests in test environment
7. execute ``python3 scripts/runAllTests.py``

Requirements traceability

If any tests fail, requirements are not being met

Tested items:

languages.js
units.js
times.js
levels.js

Test Layout:

TestID:%
Requirement:%
Component(filename):%
Expected Output:%
(optional *)call:%
(optional *)param:%

The `%` must be substituted with the related detail. An important feature of this template design is that function calls and parameters can be combined in any permutation necessary, including simply being left out entirely.

Example function calls modelable:

-> crazyobjects.objmethod.function().resultmethd
-> language().set('fr').translate('Clock')
-> units.mmolToMgdl("3")
-> language().lang
-> times.week().mins
-> levels.WARN
-> functionmadness.f1(2).f2.f3()

Tested Files

Each file has multiple functions or objects tested. They each represent slightly different styles of implementation, requiring a dynamic approach to test-case interpretation.