Ian Dilling Cameron Beacham Brandon Wagner

CamelCased Testing Framework

Introduction

This document outlines the CamelCased testing framework and gives instructions on how to use it. The framework is built in python and is intended to be run on Ubuntu Linux. Upon execution of *./scripts/runAllTests.py* the framework will read through all test case files to get inputs and will then use specified test case executable files to get an output. The results will be documented in an html file that is immediately opened in a web browser.

System Requirements

Ubuntu Linux (preferably the latest version)
Python v2.7.5

Directory Structure

There are several important directories in the framework.

Project Directory: The first directory you will need is a directory containing the code or project to be tested. Ideally you should name this directory the same name as your project. It will contain all the code to be tested.

Documentation Directory: The *docs* directory contains all the documentation required for testing (including this document). Here you can add new documentation to suit your needs.

Reports Directory: The *reports* directory contains the test reports for each test. These will be html files that can be viewed in a web browser.

Scripts Directory: The *scripts* directory contains the script you will be executing to run the tests. The *runAllTests.py* file is the main file in this directory and it runs the entire framework.

Executables Directory: The *testCaseExecutables* directory contains all driver files for your tests. This is where you will create files that executing your testing logic. (see **Test Case Executables**)

Test Cases Directory: The *testCases* directory contains a test case text file for each test case. (see **Test Case Files**)

Test Case Files

A test case file represents one test case. Each test case file is a text file, and should have the .txt extension.

When creating a test case file, you must follow this template:

Template	Example
[TestCase #]	1
[requirement]	*Given a paginator object with at least one page, paginatorObject.validate_number(1) should return 1.*
[Executable file name]	testValidate_Number.py
[Class being tested with predefined inputs]	Paginator([1,2,3,4],2)
[method being tested]	validate_number(n)
[method inputs each delimited by comma]	1
[expected output]	1

Note that the syntax of the *tested class* and the *tested method* is language agnostic. It's recommended you create these matching the relative syntax of the language you are testing.

Test Case Executables

In the CamelCased testing framework you can have as many executables as you desire. However, you are encouraged to write 1 executable per method if possible. The file outputs will be used by the script to compare the expected outcome and the actual outcome.

When each executable is run, it is passed in 3 string arguments:
0. The class being tested with inputs ex. "*Paginator([1,2,3,4],2)"*
1. The method being tested ex. "*validate_number(n)"*
2. The inputs delimited by a comma ex. "1,2"

How these inputs are handled is up to the user. It is recommended you parse the inputs and execute the method being tested with those inputs. Another method would be to use built in string evaluation to execute a string as code. This is helpful when inputs are objects, arrays, or functions.

If testing for errors, you should use some form a error handling in your executable. This is mandatory because the testing script does not handle stderr. The recommended way to handle errors is to output the name of the error caught and have that error name be the expected outcome in the test case.

Adding and Running Tests

To add a test, create the test case file and place it in the testCases directory. You may chose what to name the testCase files, however, it is required that they all be the same name and they be numbered with leading zeros depending on how many test cases there are. For instance if there are 25 tests, test cases 1-9 should have a leading zero. This allows the files to be in proper order and lets the html report keep the test cases in order.

Next, if you haven't already, create the executable that will run that test. Make sure the name of the executable file matches the one in the test case file.

To run the tests, open terminal and make sure you are executing from the root directory of the framework. The script to run is *./scripts/runAllTests.py.* It will take a few moments for all the tests to run. You should be able to see its progress. Once complete, a web browser will open with the results of each test. You can look in the results directory to find the results of each test.

The report will contain 8 columns. The first column will show the test case number and the last will show the result of either pass or fail. Pass will be colored green while fail will be colored red.