testplus: framework for structured test development

Release 1.0

June 29, 2002

Abstract

testplus is a Python module for building structured regression test suites.

Contents

List of Figures

List of Tables

1 Introduction

`testplus` is a minimal module for supporting the development and execution of generic structured regression test suites.

2 Paradigm

`testplus` is based on a very simple paradigm, which distinguishes atomic tests (test units) from test suites. This simple partition covers all testing situations.

test A test is a function with no parameter. A test passes if and only if it returns. A test fails if and only if its execution is interrupted by an exception.

test unit A test unit is an atomic test; that is, a test which cannot be decomposed as a series of sub-tests.

test suite A test suite is defined by a series of sub tests. A test suite will pass if an only if all its subtests pass.

A test structure can then be constructed by attaching prerequisites to tests:

prerequisite to a test The prerequisite to a test is a test which must pass before the test can be run. A test with a failed prerequisite is considered to have failed.

3 `tesplus` module

class `const sequence`(*TestHarness*)
(name: string, funs: seq(callable())[= []], dependents: seq(boolean)[= []], connect: callable()[= lambda: None], disconnect: callable()[= lambda : None])

This class is the work horse for defining and running test series.

Its instances are the constant sequences of their failed tests, and their string versions represent the test report.

nameis the name which will be attributed to the test series on reports. Typically, `name` will be set to `__name__`, representing the name of the Python module containing the test series.

funsis a list of objects callable without arguments; each representing a test.

dependentsis a list of boolean objects, each representing a test. If one or more is its elements has failed (`true`), nothing will be done, andthe test will be reported to have failed.

connectis an object callable without argument, which will be called before the first test is executed. Failure execute this function will make the test fail.

disconnectis an object callable without argument. It will always be called after competion of the test series, to the extent that the `connect` initialization function was called successfully.

`testcollect`(*globs: mapping,* matchfun: callable(x)[= *lambda x: re.match('test', x)*]) : `[callable()]`
Helper function to return the list of callable objects from a dictionary, sorted in source code order.

globsis the dictionary to search for the test functions. Typically, `globs` would be `globals()`.

matchfunis a filter on the callable function names of the dictionary.

`logfile`() : `file`
Return the current log file object.

```
def test_always_successful():
    '''this test will always pass
    '''
    pass

def test_doomed():
    assert 0, 'this test is doomed to fail in non-optimized mode'

def test_always_doomed():
    raise AssertError 'this test will always fail'

import testplus
harness = testplus.TestHarness( __name__,
funs = testplus.testcollect( globals()),
connect = lambda: __import__( 'apythonmodule'))

if __name__ == '__main__':
    import sys
    print harness
    sys.exit( len( harness))
```

Figure 1: simple.py: A simple test suite

prun( ]: [string[= *args*]*sys.argv[1:]*) :

Helper function to execute python code from a test subprocess.

argslist of python statement to execute in a sequentially in a same name space.

4 Example

Figure 1 describes a simple test suite, with the import of `apythonmodule` as prerequisite to running the suite.

By using testcollect(), tests will be executed in the order in which they appear in the source code.

Figure 2 describes a test suite which will execute only if simple test passed. Database connexion / disconnexion at the begining / end of the test suite are added.

Last, the `test_cube()` test consists in executing Python code in a subprocess which exits upon termination. Its non-zero exit status will be recognized as failure.

A Source Code and Informations

The source code can be obtained as libplus/testplus.py from the CVS repository associated to http://jpe.sourceforge.net.

Other informations can be obtained by contacting the author (Frédéric Giacometti) at mailto:giacometti@users.sourceforge.net.

```
import osplus, sys, testplus, simple

def test_cube():
    print osplus.pcommand( ' '.join( [sys.executable, '-c',
                                      '"import testplus; testplus.prun()"',
                                      '"import cube"',
                                      '"cube.run( timeout = 3)"']))

import somedb

harness = testplus.TestHarness( __name__,
                                connect = lambda : somedb.connect(),
                                disconnect = lambda: somedb.disconnect(),
                                funs = testplus.testcollect( globals()),
                                dependents = [ simple.harness])

if __name__ == '__main__':
    print harness
    sys.exit( len( harness))
```

Figure 2: A structured test suite

B Copyright and Licence Information

C Acknowledgements

This work was funded in part under NIH grant P41 RR08605 to the National Biomedical Computation Resource (NBCR), and carried out under the direction of Michel Sanner, in the laboratory of Arthur Olson, The Scripps Research Institute.

References

Index