UNIT TESTS OF CASA TASKS

v1.4

2010-06-02

Comments/questions: Sandra Castro

This document briefly describes how to implement the framework to run unit tests (UT) for CASA tasks. The unit tests use PyUnit[1], which is the standard Python unit tests framework. It is implemented by importing the module *unittest*. The tests are executed using nose[2], which is implemented by importing *nose*.

Existing unit tests:

A list of existing scripts already perform unit testing of tasks and have been selected to run with the UT framework. They have been converted to PyUnit and will be separated from the existing end-to-end (e2e) tests. See the list at the end of this document.

New unit tests:

They are located under *code/xmlcasa/scripts/tests*. They have been separated from the e2e tests to avoid confusion. The e2e tests remain located under *code/xmlcasa/scripts/regressions*. New unit tests should be named **test_taskname.py**.

Running unit tests:

The unit tests are meant to be called from outside a casapy session, but calling from inside casapy is also possible. The main script to run is **runUnitTest.py** located in *$CASAPATH.split()[0]/code/xmlcasa/scripts/regressions/admin.* The UT framework will search for tests located in *$CASAPATH.split()[0]/code/xmlcasa/scripts/tests*.

NOTE: If the tests are ready to run automatically using Hudson, they should be added to the list in *$CASAPATH.split()[0]/code/xmlcasa/scripts/tests/unittests_list.txt.*

1) Get the usage information.

```
casapy –c runUnitTest.py --help
```

2) Run the unit tests of task clean. It will run all test methods from all classes.

```
        casapy –c runUnitTest.py test_clean
```

3) Run only *test2* and *test23* from clean's unit tests. The names of specific tests are given as a list after the test name. There is no space between the test name and '['.

```
        casapy –c runUnitTest.py test_clean[test2,test23]
```

4) Run *test_r_r_1* from test_report, *test5* from test_clean and all tests of test_imfit and test_plotants.

```
casapy –c runUnitTest.py test_report[test_r_r_1] test_imfit
test_clean[test5] test_plotants
```

5) Run all the unit tests available. This option will run all the tests defined in the file *$CASAPATH.split()[0]/code/xmlcasa/scripts/tests/unittests_list.txt.*

```
        casapy –c runUnitTest.py
```

6) Run only a short list of unit tests.

```
        casapy –c runUnitTest.py --short
```

7) Run the tests defined in a text file. The list should contain one test per line.

```
        casapy –c runUnitTest.py --file ListofTests.txt
```

8) List all the tests available in *unittests_list.txt*.

```
        casapy –c runUnitTest.py --list
```

8) Run a test inside casapy.

```
CASA <1> : sys.path.append(os.environ["CASAPATH"].split()[0] +
'/code/xmlcasa/scripts/regressions/admin')
CASA <2> : import runUnitTest
CASA <3>: runUnitTest.main(['test_clean'])
CASA <4>: runUnitTest.main(['test_clean[test4,test22]'])
```

The framework will report at the end of the run how many tests were executed with the number of errors and failures, if any. In case of any error or failure, a traceback is printed on the screen. Any casapy command line option can be given before the *–c* option.

When executed, the script **runUnitTest.py** will create a directory called **nosedir** under the current directory. If **nosedir** already exists, it will be removed. It will save all the input and output of the tests inside **nosedir**. An XML file is created for each run under **nosedir/xml**. It contains the results of the tests, which can be displayed using Hudson.

Writing new unit tests:

Create a class which subclasses the **unittest.TestCase** class. At the beginning of the class, write variables with the names of the input and output files needed in the tests. No directory path should be given, only the filename. Write tests as methods of the main class. Each method should be an independent test. The framework identifies reserved methods, which are called before and after each test. Before each test method, a method called **setUp** will run with instructions to copy the input data to the current directory. After each test method, a **tearDown** method will run to clean up the input and optionally the output files created by the test. See example 1.

If the test does not need any input data, the **setUp** and **tearDown** methods can be omitted.

Every test method needs to have the prefix <u>test</u> so that the UT framework finds it. The order in which the tests are executed is not guaranteed, therefore the tests do need to be independent of each other and of the order they appear in the class.

Test methods do not get any arguments and do not return anything. It is possible to have other methods that get and return arguments, which can be called from inside any test method. In this case, the word **test** should not be part of the method's name. See example 2.

Outside the class, it is mandatory to have a function called **suite** that returns a list of the class(es) name(s).

New tests need only to be located in *$CASAPATH.split()[0]/code/xmlcasa/scripts/tests*, in order to be executed. If creating new data files for unit tests, place them in the svn repository under *$CASAPATH.split()[0]+'/data/regression/unittest/task_name.*

Example 1: unit tests of task clean, using two classes and two different data sets.

```
import os
import sys
import shutil
from __main__ import default
from tasks import *
from taskinit import *
import unittest

class clean_test1(unittest.TestCase):

    # Input and output names
    msfile = 'ngc7538_ut1.ms'
    res = None
    img = 'cleantest_im'
    def setUp(self):
        self.res = None
        default(clean)
        if (os.path.exists(self.msfile)):
            shutil.rmtree(self.msfile)

        datapath = os.environ.get('CASAPATH').split()[0] +
                   '/data/regression/unittest/clean/'
        shutil.copytree(datapath+self.msfile, self.msfile)

    def tearDown(self):
        if (os.path.exists(self.msfile)):
            shutil.rmtree(self.msfile)
        os.system('rm -rf ' + self.img+'*')

    def test0(self):
        '''Test 0: Default values'''
        self.res = clean()
        self.assertFalse(self.res)

    def test1(self):
        """Test 1: Wrong input should return False"""
        msfile = 'badfilename'
        self.res = clean(vis=msfile, imagename=self.img)
        self.assertFalse(self.res)

    def test2(self):
        """Test 2: Good input should return None"""
        self.res = clean(vis=self.msfile,imagename=self.img)
        self.assertEqual(self.res,None)

    def test3(self):
        """Test 3: Check if output exists"""
        self.res = clean(vis=self.msfile,imagename=self.img)
        self.assertTrue(os.path.exists(self.img+'.image'))

    def test4(self):
        '''Test 4: Non-default imagermode mode mosaic’’’
        self.res = clean(vis=self.msfile,imagename=self.img,
                         imagermode=’mosaic’)
        self.assertEqual(self.res, None)
        self.assertTrue(os.path.exists(self.img+'.image'))

    def test5(self):
        """Test 5: Non-default field value"""
        self.res = clean(vis=self.msfile,imagename=self.img,
                         field='3~8')
        self.assertEqual(self.res, None)
        self.assertTrue(os.path.exists(self.img+'.image'))


class clean_test2(unittest.TestCase):

    # Input and output names
    msfile = 'split1scan.ms'
    res = None
    img = 'cleantest2'

    def setUp(self):
        self.res = None
        default(clean)
        if (os.path.exists(self.msfile)):
            shutil.rmtree(self.msfile)
        datapath = os.environ.get('CASAPATH').split()[0] +
                    '/data/regression/unittest/clean/'
        shutil.copytree(datapath+self.msfile, self.msfile)

    def tearDown(self):
        if (os.path.exists(self.msfile)):
            shutil.rmtree(self.msfile)

    def test1a(self):
        """Clean 1a: Non-default mode velocity"""
        retValue = {'success': True, 'msgs': "", 'error_msgs': '' }
        res = clean(vis=self.msfile,imagename=self.img,
                    mode='velocity',restfreq='231901MHz')
        if(res != None):
            retValue['success']=False
            retValue['error_msgs']=retValue['error_msgs']\
                     +"\nError: Failed to run in velocity mode."
        if(not os.path.exists(self.img+'.image')):
            retValue['success']=False
            retValue['error_msgs']=retValue['error_msgs']\
                     +"\nError: Failed to create output image."

        # Verify if there are blank planes at the edges
        vals = imval(self.img+'.image')
        size = len(vals['data'])
        if (vals['data'][0]==0.0 or vals['data'][size-1]==0.0):
            retValue['success']=False
            retValue['error_msgs']=retValue['error_msgs']\
                     +"\nError: There are blank planes in the edges
                    of the image."


        self.assertTrue(retValue['success'],retValue['error_msgs'])



def suite():
    return [clean_test1,clean_test2]
```

Example 2: tests that do not need data and call a non-test method.

```
import report
import unittest

class version_test(unittest.TestCase):

    def shortDescription(self):
        return "Unit tests of comparing version strings"

    def test_fails(self):
        assert 2 + 2 == 5

    def test_execution_failure(self):
        raise Exception("die")

    def test_r_r_1(self):
        "test revision vs revision"
        a = "CASA Version 3.0.1 (r10006)"
        b = "CASA Version 3.0.1 (r9933)"
        self.order(b, a)
    def order(self, a, b):
        """Verify that the cmp_version function behaves
        as it should, given that a is earlier than b"""

        assert report.cmp_version(a, b) < 0
        assert report.cmp_version(b, a) > 0
        assert report.cmp_version(a, a) == 0
        assert report.cmp_version(b, b) == 0

def suite():
    return [version_test]
```

Note that there is another method called **shortDescription**, which is also a reserved method inside the framework. If present in the script, it will display the short description string for each test, followed by its results. If not present, the string written between 3-quotes in each test will be displayed instead.

References:

[1] PyUnit documentation, http://pyunit.sourceforge.net/pyunit.html
[2] Nose, http://somethingaboutorange.com/mrl/projects/nose/0.11.1

List of existing scripts that will be used as unit tests:

The following files are located in *scripts/regressions/tests*:
asdm_import_test
boxit_test
flagdata_test: needs to be finished
imcontsub_test: needs smaller data set
imfit_test
imhead_test
immath_test
immoment_test: needs smaller data set
imregrid_test:
imsmooth_test: needs smaller data set
imval_test
test_task_cvel: Not yet converted.
test_task_exportasdm:
vishead_test
visstat_test:

List of tests that have been converted to PyUnit.

Tests are located in code/xmlcasa/*scripts/tests:*
test_asdm-import
test_boxit
test_clean: new

test_clearstat
test_exportasdm
test_hanningsmooth
test_imcontsub
test_imfit
test_imhead
test_immath
test_immoment
test_imregrid
test_imsmooth
test_imstat: new
test_imval
test_listhistory
test_plotants
test_plotms: new
test_report
test_smoothcal
test_vishead
test_visstat