PloneTestCase

The Plone 2 Test Environment Explained

Plone Conference 2004, Vienna

Stefan H. Holek
Plone Solutions
stefan@plonesolutions.com

Goals

After attending this tutorial you should:

- Know which packages you need to install and where to get them
- Know how to run Plone tests
- Know how to add a test suite to a Plone product
- Know what the default fixture is, what it provides, and how it can be used
- Know how to write (simple) tests
- Know where to find further information

Introduction

PloneTestCase is the test framework for Plone 2. It sits on top of ZopeTestCase by the same author (as the name may have already suggested to the attentive reader).

PloneTestCase makes it possible to write automated tests for Plone and Plone-based applications. Such tests include unit tests, integration tests, and even functional tests (which use the ZPublisher).

Plone has about 600 unit tests at the moment. I want it to have 1500. That's where you come in <wink>.

Required Software

First we'll assume that you have Zope 2.7 installed, preferably a fresh checkout of Zope-2_7-branch.

Then you'll need ZopeTestCase from http://zope.org/Members/shh/ZopeTestCase. Install it into $ZOPE_HOME/lib/python/Testing (not Products!).

If you don't have a fresh checkout of Zope-2_7-branch, download the test.py script from http://zope.org/Members/shh/Tutorial/test.py and install it into $ZOPE_HOME/bin (overwriting the one that is there).

I recommend making the test.py script executable. Also make sure the first line reads:

```
#!/usr/bin/env /path/to/python2.3/bin/python
```

Note that this MUST be the Python that is running your Zope!

And of course, you will need an instance with Plone 2.0.4 installed.

Running Tests

For running unit tests we typically use a test runner. There is a variety of test runners available, some work with Zope, and some even work with Zope 2. The fundamental issue is that for starting up Zope, a test runner needs to know about SOFTWARE_HOME and INSTANCE_HOME.

For this tutorial we'll stick with the test.py script, which we have more or less elaborately installed before.

Now we have everything we need to run the Plone tests. Let's do that then:

```
cd $INSTANCE_HOME
$ZOPE_HOME/bin/test.py -v  \
    --config-file etc/zope.conf  \
    --libdir Products/CMFPlone
```

What's Going On?

- First, the test runner tells us that it is going to run unit tests (not functional tests) from $INSTANCE_HOME/Products/CMFPlone.
- Second, it configures Zope from the specified config file.
- Third, it scans for and imports test modules.
  (Large parts of the PloneTestCase magic happen at import time, for example all required Zope products are installed, and a Plone site is created.)
- Finally, test.py runs the accumulated tests.

Now let's run just the membership tool tests:

```
cd $INSTANCE_HOME
$ZOPE_HOME/bin/test.py -v  \
    --config-file etc/zope.conf  \
    --libdir Products/CMFPlone   \
    testMembershipTool
```

There is a more detailed description of how test.py processes tests in the test runner comparison document at:
http://zope.org/Members/shh/TestRunnerComparison

How about runalltests.py, you ask? It is a small, featureless test runner. It exists because I think it is neat, YMMV.

Writing Tests

Let's digress for a quick overview of PyUnit concepts

- Test Case
  Tests a single scenario
- Test Fixture
  Preparations needed to run a test
- Test Suite
  Aggregation of multiple test cases
- Test Runner
  Runs a test suite and presents the results

In a TestCase

- The setUp() hook is used to create the fixture.
- The tearDown() hook may be used to destroy the fixture, if necessary.
- Names of test methods must start with a common prefix, typically 'test'.

A simple PyUnit test:

```
import unittest

class MyTest(unittest.TestCase):

    def setUp(self):
        self.answer = 42

    def testAnswer(self):
        self.assertEqual(self.answer, 42)

def test_suite():
    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(MyTest))
    return suite
```

For comparison, here is the PloneTestCase version:

```
from Products.CMFPlone.tests import PloneTestCase

class MyTest(PloneTestCase.PloneTestCase):

    def afterSetUp(self):
        self.answer = 42

    def testAnswer(self):
        self.assertEqual(self.answer, 42)

def test_suite():
    from unittest import TestSuite, makeSuite
    suite = TestSuite()
    suite.addTest(makeSuite(MyTest))
    return suite
```

What's Different?

- We don't derive from unittest.TestCase but from PloneTestCase.PloneTestCase (which of course is a subclass of unittest.TestCase).
- We are NOT allowed to use the setUp() and tearDown() hooks of PyUnit as they are reserved by PloneTestCase. PloneTestCase provides it's own hooks, notably afterSetUp(), beforeTearDown(), and afterClear(). (I could tell you why, but I won't.)

Hands On!

Create a dummy product:

```
cd $INSTANCE_HOME/Products
mkdir Tutorial
touch Tutorial/__init__.py
mkdir Tutorial/tests
touch Tutorial/tests/__init__.py
```

Go to Tutorial/tests and type in the first tests from above, name the file testAnswer.py

How would you run it?

```
cd $INSTANCE_HOME/Products/Tutorial
$ZOPE_HOME/bin/test.py -v --libdir . testAnswer
```

Now do the same for the second test, naming the file testPloneAnswer.py.

```
$ZOPE_HOME/bin/test.py -v  \
    --config-file ../../etc/zope.conf  \
    --libdir . testPloneAnswer
```

And to run all tests in Tutorial:

```
$ZOPE_HOME/bin/test.py -v  \
    --config-file ../../etc/zope.conf  \
    --libdir .
```

Default Fixture

To write more interesting Plone tests, we first need to know more about the test environment.
We have already seen that PloneTestCase creates a complete Plone site for us. It doesn't stop there though.

But what do we need?

- An Application object
- A REQUEST
- A Plone Site object
- A User Folder
- A default user with role 'Member'
- A member area for the default user
- And we need to log in

Don’t worry, all is catered for. In hooks and test methods of a PloneTestCase subclass you can access these objects as:

- self.app
- self.app.REQUEST
- self.portal
- self.portal.acl_users
- self.folder

Use the ‘PloneTestCase.default_user’ constant when you need the default user’s name, ‘PloneTestCase.portal_name’ should you need the name of the portal.

What Is this Man Telling Me?

Fortunately, you are now in a position where you no longer need to just trust me. You can write (and run) tests to verify my claims. Like so:

```
from Products.CMFPlone.tests import PloneTestCase
from AccessControl import getSecurityManager

portal_name = PloneTestCase.portal_name
default_user = PloneTestCase.default_user

class FixtureTest(PloneTestCase.PloneTestCase):

    def testApp(self):
        self.failUnless(
            'Control_Panel' in self.app.objectIds())

    def testPortal(self):
        self.failUnless(
            portal_name in self.app.objectIds())

    def testMembersFolder(self):
        self.failUnless(
            'Members' in self.portal.objectIds())

    def testUserFolder(self):
        self.failUnless(
            'acl_users' in self.portal.objectIds())

    def testUser(self):
        uf = self.portal.acl_users
        self.failIf(uf.getUserById(default_user) is None)
```

```
    def testMemberArea(self):
        self.assertEqual(
            self.portal.Members[default_user],
            self.folder)

    def testRequest(self):
        self.failUnless(
            self.app.REQUEST.has_key('SERVER_URL'))

    def testAcquiredRequest(self):
        self.failUnless(
            self.folder.REQUEST.has_key('SERVER_URL'))

    def testLoggedIn(self):
        auth_user = \
            getSecurityManager().getUser().getId()
        self.assertEqual(auth_user, default_user)

def test_suite():
    from unittest import TestSuite, makeSuite
    suite = TestSuite()
    suite.addTest(makeSuite(FixtureTest))
    return suite
```

You may want to type this in, at least partially. Name the file testFixture.py. You know how to run the tests by now (or I have failed <wink>).

Observations

- The import of PloneTestCase must be the first import statement in every test module.
- PyUnit provides methods that help with making assertions: failUnless(), assertEqual(), etc.
- The Zope API works.
- Acquisition works.
- Everything this man says is true <wink>.

Want more? My pleasure!
(This may be a good time to install DocFinderTab, if you haven't already.)

Testing Content

As Plone is about content management, let's see how to test a content object.

```
from Products.CMFPlone.tests import PloneTestCase
from Acquisition import aq_base

class DocumentTest(PloneTestCase.PloneTestCase):

    def afterSetUp(self):
        self.catalog = self.portal.portal_catalog
        self.workflow = self.portal.portal_workflow
        # Create a document in our home folder
        self.folder.invokeFactory('Document', id='doc')

    def testAddDocument(self):
        self.failUnless(
            hasattr(aq_base(self.folder), 'doc'))

    def testEditDocument(self):
        self.folder.doc.edit(
            text_format='plain', text='foo')
        self.assertEqual(
            self.folder.doc.EditableBody(), 'foo')

    def testFindDocument(self):
        self.failUnless(self.catalog(id='doc'))

    def testPublishDocument(self):
        self.setRoles(['Reviewer'])
        self.workflow.doActionFor(
            self.folder.doc, 'publish')
        state = self.workflow.getInfoFor(
            self.folder.doc, 'review_state')
        self.assertEqual(state, 'published')

def test_suite():
    from unittest import TestSuite, makeSuite
    suite = TestSuite()
    suite.addTest(makeSuite(DocumentTest))
    return suite
```

Observations

- The Plone site works. We can create documents, edit them, and find them in the catalog. We can even use workflow!
- We can use the setRoles() API to change the roles of the default user.
- We have to strip off undesired acquisition wrappers using aq_base().
- We don't need to clean up.

Testing Security

```
from Products.CMFPlone.tests import PloneTestCase
from AccessControl import Unauthorized

default_user = PloneTestCase.default_user

class SecurityTest(PloneTestCase.PloneTestCase):

    def afterSetUp(self):
        self.folder.invokeFactory('Document', id='doc')
        self.folder.doc.manage_permission(
            'View', ['Manager'], acquire=0)

    def testOwnerViewsDocument(self):
        self.assertRaises(Unauthorized,
            self.folder.restrictedTraverse, 'doc')

    def testManagerViewsDocument(self):
        self.setRoles(['Manager'])
        self.folder.restrictedTraverse('doc')

class MultiUserTest(PloneTestCase.PloneTestCase):

    def afterSetUp(self):
        self.membership = self.portal.portal_membership
        self.membership.addMember(
            'user2', 'secret', ['Member'], [])

        self.folder.invokeFactory('Document', id='doc')
        self.folder.doc.manage_permission(
            'View', ['Owner'], acquire=0)

    def testOwnerViewsDocument(self):
        self.folder.restrictedTraverse('doc')

    def testMemberViewsDocument(self):
        self.login('user2')
        self.assertRaises(Unauthorized,
            self.folder.restrictedTraverse, 'doc')

    def testAnonymousViewsDocument(self):
        self.logout()
        self.assertRaises(Unauthorized,
            self.folder.restrictedTraverse, 'doc')
```

```
def test_suite():
    from unittest import TestSuite, makeSuite
    suite = TestSuite()
    suite.addTest(makeSuite(SecurityTest))
    suite.addTest(makeSuite(MultiUserTest))
    return suite
```

Observations

- We need to trigger Zope security validation by explicitly calling restrictedTraverse().
- We can use the setRoles() API to change the roles of the default user.
- We can use the login() API to log in as another user.
- We can use the logout() API to log out and become Anonymous User.
- We can write more than one TestCase in a single module as long as we add all of them to the test suite.

Summary

We demonstrated how to run test suites and individual test modules, and we showed how to construct TestCases derived from the PloneTestCase base class. We described the default fixture and hinted at the security API. The tutorial at the conference will (time permitting) walk you through even more elaborate test scenarios, for example from the CMFPlone test suite.

Given the right tools, it can be straightforward to write automated tests for Zope and Plone. The PloneTestCase package provides you with a fully featured Plone environment, allowing you to concentrate on writing tests while leaving the plumbing to the framework.

Resources

Dive Into Python, Chapter on Unit Testing

http://diveintopython.org/unit_testing/index.html

PyUnit Documentation

http://www.python.org/doc/current/lib/module-unittest.html

ZopeTestCase

http://zope.org/Members/shh/ZopeTestCase
http://zope.org/Members/shh/ZopeTestCaseWiki
CVS access through Collective

CMFTestCase

http://zope.org/Members/shh/CMFTestCase
CVS access through Collective

PloneTestCase (standalone)

CVS access through Collective

DocFinderTab

http://zope.org/Members/shh/DocFinderTab

testrunner.py

http://zope.org/Members/shh/TestRunner

test.py

Ships with Zope >= 2.7.3

Test Runner Comparison

http://zope.org/Members/shh/TestRunnerComparison

Code Examples for the Tutorial

http://zope.org/Members/shh/Tutorial