PloneTestCase

The Plone 2 Test Environment Explained

Stefan H. Holek
stefan@plonesolutions.com

Goals

- Know which packages we 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

At the and of this tutorial, we will...

Goals II

- 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 additional information

Quote

"Program testing can be used very efficiently to prove the presence of bugs, but never to show their absence."

--E. W. Dykstra

So what am I doing here?

Yes, But

- This won't keep us from trying <wink>
- Dykstra was after a "mathematical proof" kind of correctness in software programs
- We can do with a "pretty darn good" kind of correctness, thank you

Try to think of an alternative! Without automated tests we are in random country anyway. Any amount of order we can introduce into the process can only be a good thing. Even if it is not 100%.

And

"Software Engineering is Programming when you can't."

--E. W. Dykstra

There is also no alternative to the fact that software programs must be created by programmers. We cannot engineer away real life constraints like complexity and uncertainty. What we *can* do, is use tools and adopt practices that allow us to cope with those things. Writing automated tests is one of these practices; PloneTestCase is one of these tools.

Intro

- PloneTestCase is the test framework for Plone 2
- It sits on top of ZopeTestCase
- It allows to easily write automated tests for Plone and Plone-based applications
- Plone 2 has about 600 unit and integration tests (at the time of this writing)

Required Software

- Zope 2.7.2
- Even better: Zope-2_7-branch
- ZopeTestCase 0.9.2
  `http://zope.org/Members/shh/ZopeTestCase`
- Plone 2.0.4

test.py

- If you don’t have Zope-2_7-branch:
  `http://zope.org/Members/shh/Tutorial/test.py`
- Make it executable:
  `chmod a+x test.py`
- Make sure the first line reads:
  `#!/usr/bin/env /path/to/python2.3/bin/python`

  This MUST be the Python that is running your Zope!

Using the wrong Python interpreter probably is the #1 mistake when running tests

Try It!

```
$ZOPE_HOME/bin/test.py --help
```

Running Tests

Test Runners

- For running automated tests we typically use a test runner.
- There are a few out there, some of them even work with Zope 2.
- Main issue is that a test runner needs to be able to configure and startup Zope.

Running All Tests

```
cd $INSTANCE_HOME

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

Running All Tests (alt)

```
cd $INSTANCE_HOME

/path/to/python2.3/bin/python \
$ZOPE_HOME/bin/test.py -v     \
  --config-file etc/zope.conf \
  --libdir Products/CMFPlone
```

What's Going On?

- test.py tells us it is about to run unit tests from `$INSTANCE_HOME/Products/CMFPlone`
- test.py configures Zope from the config file
- test.py scans for and imports test modules
- test.py runs the accumulated tests

Observation

- Large parts of the PloneTestCase magic happen at import time, for example all required Zope products are installed, and a Plone site is created.

Running a Single Module

```
cd $INSTANCE_HOME

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

Writing Tests

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

TestCase Concepts

- 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”.

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
```

The Test Case abstraction comes in the form of a base class.
The setUp() hook is used to set up the fixture.
The name of the test method starts with “test”.
The test_suite() function is called by test runners.

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.
- We are NOT allowed to use the PyUnit hooks; they are reserved by PloneTestCase!
- PloneTestCase provides its own hooks, notably afterSetUp(), beforeTearDown(), and afterClear().

PloneTestCase is of course ultimately derived from unittest.TestCase!
afterSetUp() is the most useful hook, by a wide margin.

Dummy Product

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

cd Tutorial/tests
```

This being a tutorial, we have to start actually doing things

testAnswer

- Type in the PyUnit test from before, name the file testAnswer.py.
- How would you run it?

Correct!

```
cd $INSTANCE_HOME/Products/Tutorial

$ZOPE_HOME/bin/test.py -v \
  --libdir . testAnswer
```

testPloneAnswer

- Now type in the second test and name the file testPloneAnswer.py. Then run it:

```
cd $INSTANCE_HOME/Products/Tutorial

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

We need to pass a config file because this is a Zope test.

Running All Tests

```
cd $INSTANCE_HOME/Products/Tutorial

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

Without a module filter, all tests will be run.

Writing Interesting Tests

Default Fixture

- To write less boring tests, we need to know more about the test environment.
- We have already seen that PloneTestCase creates a Plone site for us, and it doesn't stop there...

What Do We Want?

- 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 want the default user to be logged in

Fixture Attributes

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

This is how PloneTestCase provides access to the individual fixture components.

Err... ?

- You feel a little uneasy about proceeding?
- You don't think you have fully grasped this "default fixture" thing?
- Excellent!
- That's a perfect time to write some tests...

Download

- At this point you will want to download the Tutorial product from:

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

Because this is a testing tutorial and not a typing tutorial, we will download the rest of the example tests.

testFixture

```
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())
```

testFixture II

```
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)
```

You can surely think of more test you want to right. Like, does the default user really have the Member role?

testFixture III

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

Observations

- PloneTestCase must be imported first thing
- There are methods that help with making assertions: failUnless(), assertEqual(), etc.
- The Zope API works
- Acquisition works

-> So we *do* have a fully featured Zope/Plone environment after all

testDocument

```
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
        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'))
```

testDocument II

```
    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 add documents, edit them, and find them in the catalog. We can even use workflow!
- We create new objects in our member area: self.folder
- We can use the setRoles() API to change our roles

Observations II

- We have to strip off undesired acquisition wrappers using aq_base()
- We don't need to clean up!

testSecurity

```
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')
```

testSecurity II

```
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')
```

testSecurity III

```
    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 login() API to log in as another user
- We can use the logout() API to log out and become Anonymous User

Observations II

- We can write more than one test case in a single module, as long as we add all of them to the test suite.

Summary

- We downloaded and installed required software
- We successfully ran various kinds of tests
- We created a product including a test suite

Summary II

- We learned about the default fixture and wrote tests to make sure we got that correctly.
- We wrote our first PloneTestCase tests

Shameless Plug

Should you be interested in an intense 2 or 3-day testing workshop for yourself and your development team contact:

info@plonesolutions.com

Stay With Us!

Until after the break

Future

Standalone Version

- PloneTestCase will we moved out of CMFPlone into its own product
- Test authors will get control over the default Plone site

Standalone Example

```
from Products.PloneTestCase import PloneTestCase

PloneTestCase.installProduct('Foo')
PloneTestCase.setupPloneSite(products=('Foo',))

class FooTest(PloneTestCase.PloneTestCase):

    ...
```

CMFTestCase

“Little Brother”

- CMFTestCase provides a CMFDefault environment and portal.
- CMF is more lightweight than Plone which makes the tests import and run significantly faster.

CTC Example

```
from Products.CMFTestCase import CMFTestCase

CMFTestCase.installProduct('Foo')
CMFTestCase.setupCMFSite(products=('Foo',))

class FooTest(CMFTestCase.CMFTestCase):

    ...
```

testrunner.py

Pocket Chain Saw

- testrunner.py knows about instance homes
- testrunner.py typically can do with less command line real estate than test.py

Note that test.py does *not* know about instance homes; it only knows about config files.

Examples

```
cd $INSTANCE_HOME/Products
$ZOPE_HOME/bin/testrunner.py -qid CMFPlone/tests

$ZOPE_HOME/bin/testrunner.py -q \
    -I $INSTANCE_HOME \
    -d CMFPlone/tests


cd $INSTANCE_HOME/Products/CMFPlone/tests
$ZOPE_HOME/bin/testrunner.py -qia

$ZOPE_HOME/bin/testrunner.py -qif testCheckId.py
```

testrunner.py can detect instance homes. testrunner.py can run tests from inside the “tests” package. testrunner.py can NOT run Zope’s test suite.

When to Write Tests?

When to Write Tests?

- First!
- To raise confidence in existing code
- To expose a bug and to prove it is fixed
- Whenever we feel stupid

1. TDD by Kent Beck!
2. That’s what we did with Plone, BTW (I **guarantee** you a very sobering experience).
3. “A bug is a test not written.”

Thanks!