plonetraining.testing Documentation

Release 1.0

Timo Stollenwerk

December 03, 2016

Contents

CHAPTER 1

Talk: Test-Driven Development with Plone

CHAPTER 2

Contents

2.1 Part 1: Package Setup

2.1.1 Create Package

Create virtual Python environment:

```
$ virtualenv-2.7 .env
```

Activate virtual Python environment:

```
$ source .env/bin/activate
```

Install mr.bob with plone templates:

```
$ pip install mr.bob bobtemplates.plone
```

Create a new 'plone_addon' package:

```
$ mrbob -O plonetraining.testing bobtemplates:plone_addon
```

2.1.2 Buildout

Run buildout:

```
$ cd plonetraining.testing
$ python bootstrap-buildout.py --setuptools-version=8.3
$ bin/buildout
```

Run tests:

```
$ bin/test
```

Run all tests including robot tests:

```
$ bin/test --all
```

2.2 Part 2: Plone Testing Setup

2.2.1 testing.py

```
# -*- coding: utf-8 -*-
"""Base module for unittesting."""

from plone.app.robotframework.testing import REMOTE_LIBRARY_BUNDLE_FIXTURE
from plone.app.testing import applyProfile
from plone.app.testing import FunctionalTesting
from plone.app.testing import IntegrationTesting
from plone.app.testing import PLONE_FIXTURE
from plone.app.testing import PloneSandboxLayer
from plone.testing import z2
from zope.configuration import xmlconfig

import plonetraining.testing


class PlonetrainingTestingLayer(PloneSandboxLayer):

    defaultBases = (PLONE_FIXTURE,)

    def setUpZope(self, app, configurationContext):
        print('\n ---> setUpZope \n')
        xmlconfig.file(
            'configure.zcml',
            plonetraining.testing,
            context=configurationContext
        )

    def setUpPloneSite(self, portal):
        print('\n ---> setUpPloneSite \n')
        applyProfile(portal, 'plonetraining.testing:default')

    def tearDownZope(self, app):
        print('\n ---> tearDownZope \n')

    def tearDownPloneSite(self, portal):
        print('\n ---> tearDownPloneSite \n')


PLONETRAINING_TESTING_FIXTURE = PlonetrainingTestingLayer()

PLONETRAINING_TESTING_INTEGRATION_TESTING = IntegrationTesting(
    bases=(PLONETRAINING_TESTING_FIXTURE,),
    name='PlonetrainingTestingLayer:IntegrationTesting'
)

PLONETRAINING_TESTING_FUNCTIONAL_TESTING = FunctionalTesting(
    bases=(PLONETRAINING_TESTING_FIXTURE,),
    name='PlonetrainingTestingLayer:FunctionalTesting'
)

PLONETRAINING_TESTING_ACCEPTANCE_TESTING = FunctionalTesting(
    bases=(
        PLONETRAINING_TESTING_FIXTURE,
```

```
        REMOTE_LIBRARY_BUNDLE_FIXTURE,
        z2.ZSERVER_FIXTURE
    ),
    name='PlonetrainingTestingLayer:AcceptanceTesting'
)
```

2.2.2 Testing Layers

1. testing.py: setUpZope(self, app, configurationContext)
2. testing.py: setUpPloneSite(self, portal)
3. test_setup.py: setUp
4. test_setup.py: test_product_installed
5. test_setup.py: tearDown
6. tearDownPloneSite(self, portal)
7. tearDownZope(self, app)

2.2.3 tests/test_setup.py

```
# -*- coding: utf-8 -*-
"""Setup/installation tests for this package."""
from plonetraining.testing.testing import PLONETRAINING_TESTING_INTEGRATION_TESTING  # noqa
from plone import api

import unittest2 as unittest


class TestInstall(unittest.TestCase):
    """Test installation of plonetraining.testing into Plone."""

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        """Custom shared utility setup for tests."""
        self.portal = self.layer['portal']
        self.installer = api.portal.get_tool('portal_quickinstaller')

    def test_product_installed(self):
        """Test if plonetraining.testing is installed with portal_quickinstaller."""
        self.assertTrue(self.installer.isProductInstalled('plonetraining.testing'))

    def test_uninstall(self):
        """Test if plonetraining.testing is cleanly uninstalled."""
        self.installer.uninstallProducts(['plonetraining.testing'])
        self.assertFalse(self.installer.isProductInstalled('plonetraining.testing'))

    # browserlayer.xml
    def test_browserlayer(self):
        """Test that IPlonetrainingTestingLayer is registered."""
        from plonetraining.testing.interfaces import IPlonetrainingTestingLayer
        from plone.browserlayer import utils
        self.assertIn(IPlonetrainingTestingLayer, utils.registered_layers())
```

2.3 Part 3: Testing Dexterity

2.3.1 Create simple Dexterity Type

Add Dexterity to the package (setup.py):

```
    install_requires=[
        'plone.api',
        'setuptools',
        'z3c.jbot',
        'plone.app.dexterity',
        'plone.app.portlets',
```

Make sure dexterity is installed together with the package:

```
<?xml version="1.0"?>
<metadata>
  <version>1000</version>
  <dependencies>
    <dependency>profile-plone.app.dexterity:default</dependency>
  </dependencies>
</metadata>
```

configure.zcml:

```
<includeDependencies package="." />
```

Create profiles/default/types directory:

```
$ mkdir profiles/default/types
```

Create Factory Type Information (FTI) for Task Type (profiles/default/types/Task.xml):

```
<?xml version="1.0"?>
<object name="Task" meta_type="Dexterity FTI" i18n:domain="plonetraining.testing"
   xmlns:i18n="http://xml.zope.org/namespaces/i18n">
  <property name="title" i18n:translate="">Task</property>
  <property name="description"
    i18n:translate=""></property>
  <property name="icon_expr">string:${portal_url}/++theme++plonetraining.testing/Task.png</property>
  <property name="factory">Task</property>
  <property name="add_view_expr">string:${folder_url}/++add++Task</property>
  <property name="link_target"></property>
  <property name="immediate_view">view</property>
  <property name="global_allow">True</property>
  <property name="filter_content_types">True</property>
  <property name="allowed_content_types">
  </property>
  <property name="allow_discussion">False</property>
  <property name="default_view">view</property>
  <property name="view_methods">
    <element value="view"/>
  </property>
  <property name="default_view_fallback">False</property>
  <property name="add_permission">cmf.AddPortalContent</property>
  <property name="klass">plone.dexterity.content.Item</property>
  <property name="behaviors">
    <element value="plone.app.content.interfaces.INameFromTitle"/>
  </property>
```

```
  <property name="schema">plonetraining.testing.interfaces.ITask</property>
  <property name="model_source"></property>
  <property name="model_file"></property>
  <alias from="(Default)" to="(dynamic view)" />
  <alias from="edit" to="@@edit" />
  <alias from="sharing" to="@@sharing" />
  <alias from="view" to="(selected layout)" />
  <action title="View" action_id="view" category="object" condition_expr=""
    description="" icon_expr="" link_target="" url_expr="string:${object_url}"
    visible="True">
    <permission value="View" />
  </action>
  <action title="Edit" action_id="edit" category="object" condition_expr=""
    description="" icon_expr="" link_target=""
    url_expr="string:${object_url}/edit" visible="True">
    <permission value="Modify portal content" />
  </action>
</object>
```

Include Task FTI in Generic Setup Profile (profiles/default/types.xml):

```
<?xml version="1.0"?>
<object name="portal_types" meta_type="Plone Types Tool">
  <object name="Task" meta_type="Dexterity FTI" />
</object>
```

2.3.2 Interface

interfaces.py:

```
# -*- coding: utf-8 -*-
from plonetraining.testing import _
from zope import schema
from zope.interface import Interface
from zope.publisher.interfaces.browser import IDefaultBrowserLayer


class IPlonetrainingTestingLayer(IDefaultBrowserLayer):
    """Marker interface that defines a browser layer."""


class ITask(Interface):

    title = schema.TextLine(
        title=_(u"Title"),
        required=True,
    )

    description = schema.Text(
        title=_(u"Description"),
        required=False,
    )
```

2.3.3 Integration Test

tests/test_task.py:

```
# -*- coding: utf-8 -*-
from plone.app.testing import SITE_OWNER_NAME
from plone.app.testing import SITE_OWNER_PASSWORD
from plone.testing.z2 import Browser
from plone.app.testing import TEST_USER_ID
from zope.component import queryUtility
from zope.component import createObject
from plone.app.testing import setRoles
from plone.dexterity.interfaces import IDexterityFTI
from plonetraining.testing.testing import PLONETRAINING_TESTING_FUNCTIONAL_TESTING  # noqa
from plonetraining.testing.testing import PLONETRAINING_TESTING_INTEGRATION_TESTING  # noqa
from plonetraining.testing.interfaces import ITask

import unittest2 as unittest


class TaskIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])

    def test_schema(self):
        fti = queryUtility(IDexterityFTI, name='Task')
        schema = fti.lookupSchema()
        self.assertEqual(ITask, schema)

    def test_fti(self):
        fti = queryUtility(IDexterityFTI, name='Task')
        self.assertTrue(fti)

    def test_factory(self):
        fti = queryUtility(IDexterityFTI, name='Task')
        factory = fti.factory
        task = createObject(factory)
        self.assertTrue(ITask.providedBy(task))

    def test_adding(self):
        self.portal.invokeFactory('Task', 'task')
        self.assertTrue(ITask.providedBy(self.portal.task))
```

2.3.4 Functional Test

tests/test_task.py:

```
# -*- coding: utf-8 -*-
from plone.app.testing import SITE_OWNER_NAME
from plone.app.testing import SITE_OWNER_PASSWORD
from plone.testing.z2 import Browser
from plone.app.testing import TEST_USER_ID
from zope.component import queryUtility
from zope.component import createObject
from plone.app.testing import setRoles
from plone.dexterity.interfaces import IDexterityFTI
```

```
from plonetraining.testing.testing import PLONETRAINING_TESTING_FUNCTIONAL_TESTING  # noqa
from plonetraining.testing.testing import PLONETRAINING_TESTING_INTEGRATION_TESTING  # noqa
from plonetraining.testing.interfaces import ITask

import unittest2 as unittest


class TaskFunctionalTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_FUNCTIONAL_TESTING

    def setUp(self):
        app = self.layer['app']
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        self.portal_url = self.portal.absolute_url()

        # Set up browser
        self.browser = Browser(app)
        self.browser.handleErrors = False
        self.browser.addHeader(
            'Authorization',
            'Basic %s:%s' % (SITE_OWNER_NAME, SITE_OWNER_PASSWORD,)
        )

    def test_add_task(self):
        self.browser.open(self.portal_url + '/++add++Task')
        self.browser.getControl(name="form.widgets.title").value = \
            "My Task"
        self.browser.getControl(name="form.widgets.description")\
            .value = "This is my task"
        self.browser.getControl("Save").click()

        self.assertEqual(
            "My Task",
            self.portal['my-task'].title,
        )

    def test_view_task(self):
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory(
            "Task",
            id="my-task",
            title="My Task",
        )

        import transaction
        transaction.commit()

        self.browser.open(self.portal_url + '/my-task')

        self.assertTrue('My Task' in self.browser.contents)
```

2.3.5 Robot Test

tests/robot/test_task.robot:

```
# ============================================================================
# EXAMPLE ROBOT TESTS
# ============================================================================
#
# Run this robot test stand-alone:
#
#  $ bin/test -s plonetraining.testing -t test_task.robot --all
#
# Run this robot test with robot server (which is faster):
#
# 1) Start robot server:
#
# $ bin/robot-server --reload-path src plonetraining.testing.testing.PLONETRAINING_TESTING_ACCEPTANCE
#
# 2) Run robot tests:
#
# $ bin/robot src/plonetraining/testing/tests/robot/test_task.robot
#
# See the http://docs.plone.org for further details (search for robot
# framework).
#
# ============================================================================

*** Settings *****************************************************************

Resource  plone/app/robotframework/selenium.robot
Resource  plone/app/robotframework/keywords.robot

Library  Remote  ${PLONE_URL}/RobotRemote

Test Setup  Open test browser
Test Teardown  Close all browsers


*** Test Cases ***************************************************************

Scenario: As a site administrator I can add a Task
  Given a logged-in site administrator
    and an add task form
   When I type 'My Task' into the title field
    and I submit the form
   Then a task with the title 'My Task' has been created

Scenario: As a site administrator I can view a Task
  Given a logged-in site administrator
    and a task 'My Task'
   When I go to the task view
   Then I can see the task title 'My Task'


*** Keywords *****************************************************************

# --- Given ------------------------------------------------------------------

a logged-in site administrator
  Enable autologin as  Site Administrator

an add task form
```

```
  Go To  ${PLONE_URL}/++add++Task

a task 'My Task'
  Create content  type=Task  id=my-task  title=My Task


# --- WHEN -------------------------------------------------------------------

I type '${title}' into the title field
  Input Text  name=form.widgets.title  ${title}

I submit the form
  Click Button  Save

I go to the task view
  Go To  ${PLONE_URL}/my-task
  Wait until page contains  Site Map


# --- THEN -------------------------------------------------------------------

a task with the title '${title}' has been created
  Wait until page contains  Site Map
  Page should contain  ${title}
  Page should contain  Item created

I can see the task title '${title}'
  Wait until page contains  Site Map
  Page should contain  ${title}
```

2.4 Part 4: Testing Views

2.4.1 Test Simple View

```
class TaskViewIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_with_get_multi_adapter(self):
        # Get the view
        view = getMultiAdapter((self.task, self.request), name="view")
        # Put the view into the acquisition chain
        view = view.__of__(self.task)
        # Call the view
        self.assertTrue(view())

    def test_view_with_restricted_traverse(self):
        view = self.task.restrictedTraverse('view')
```

```
        self.assertTrue(view())

    def test_view_with_unrestricted_traverse(self):
        view = self.task.unrestrictedTraverse('view')
        self.assertTrue(view())

    def test_view_html_structure(self):
        import lxml
        view = getMultiAdapter((self.task, self.request), name="view")
        view = view.__of__(self.task)
        output = lxml.html.fromstring(view())
        self.assertEqual(1, len(output.xpath("/html/body/div")))
```

Implementation

browser/configure.zcml:

```
  <!-- Task View -->
  <browser:page
    name="view"
    for="plonetraining.testing.interfaces.ITask"
    class=".task.TaskView"
    permission="zope2.View"
    />
```

browser/task.py:

```
from zope.site.hooks import getSite
from Products.Five.browser import BrowserView
from Products.Five.browser.pagetemplatefile import ViewPageTemplateFile

import json


class TaskView(BrowserView):

    template = ViewPageTemplateFile('task.pt')
```

browser/task.pt:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      xmlns:i18n="http://xml.zope.org/namespaces/i18n"
      lang="en"
      metal:use-macro="here/prefs_main_template/macros/master"
      i18n:domain="plonetraining.testing">

<body>

<div metal:fill-slot="content-core">
  <h1 tal:content="context/title">Task</h1>
  <p tal:content="context/description|nothing">Task Description</p>
  <p>
    Request Term:
    <span tal:content="context/term|nothing">Request Term</span>
  </p>
</div>
```

```
</body>
</html>
```

2.4.2 Test View Browserlayer

```
class TaskViewWithBrowserlayerIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_with_browserlayer(self):
        # Make the request provide the browser layer so our view can be looked
        # up
        from zope.interface import directlyProvides
        from plonetraining.testing.interfaces import IPlonetrainingTestingLayer
        directlyProvides(self.request, IPlonetrainingTestingLayer)
        # Get the view
        view = getMultiAdapter(
            (self.task, self.request),
            name="view-with-browserlayer"
        )
        # Put the view into the acquisition chain
        view = view.__of__(self.task)
        # Call the view
        self.assertTrue(view())
```

2.4.3 Test View with Request Parameter

```
class TaskViewWithRequestParameterIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_with_request_parameter(self):
        self.request.set('term', 'foo')
        view = getMultiAdapter(
            (self.task, self.request),
            name="view-with-params"
        )
        view = view.__of__(self.task)
        self.failUnless(view())
```

2.4.4 Test Protected View

```
class TaskViewProtectedIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_protected(self):
        """Try to access a protected view and make sure we raise Unauthorized.
        """
        from AccessControl import Unauthorized
        logout()
        self.assertRaises(
            Unauthorized,
            self.task.restrictedTraverse,
            'view-protected'
        )
```

2.4.5 Test JSON View

```
class TaskViewJsonIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_json(self):
        view = getMultiAdapter(
            (self.task, self.request),
            name="view-json"
        )
        view = view.__of__(self.task)

        self.assertEqual(
            {
                u'title': u'Task',
                u'description': u''
            },
            json.loads(view())
        )
        self.assertEqual(
            'application/json; charset=utf-8',
            view.request.response.headers.get('content-type'),
        )
```

2.4.6 Test XML View

```
class TaskViewXmlIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_json(self):
        view = getMultiAdapter(
            (self.task, self.request),
            name="view-xml"
        )
        view = view.__of__(self.task)

        import lxml
        output = lxml.etree.fromstring(view())

        self.assertEqual(len(output.xpath("/task/title")), 1)
        self.assertEqual(output.xpath("/task/title")[0].text, u'Task')
        self.assertEqual(len(output.xpath("/task/description")), 1)
        self.assertEqual(output.xpath("/task/description")[0].text, None)
        self.assertEqual(
            'application/xml; charset=utf-8',
            view.request.response.headers.get('content-type')
        )
```

2.4.7 Test View Redirect

```
class TaskViewRedirectIntegrationTest(unittest.TestCase):

    layer = PLONETRAINING_TESTING_INTEGRATION_TESTING

    def setUp(self):
        self.portal = self.layer['portal']
        self.request = self.layer['request']
        setRoles(self.portal, TEST_USER_ID, ['Manager'])
        self.portal.invokeFactory('Task', id='task', title='Task')
        self.task = self.portal.task

    def test_view_redirect(self):
        view = getMultiAdapter(
            (self.task, self.request),
            name="view-redirect"
        )
        view = view.__of__(self.task)

        view()

        self.assertEqual(
            'http://nohost/plone',
```

```
            self.request.response.headers['location']
        )
```

2.4.8 Troubleshooting

KeyError: 'ACTUAL_URL'

Sometimes a view expect an 'ACTUAL_URL' param. If this is the case, make sure you provide the param in the test request:

```
def setUp(self):
    self.portal = self.layer['portal']
    self.request = self.layer['request']
    setRoles(self.portal, TEST_USER_ID, ['Manager'])
    self.portal.invokeFactory('Folder', 'test-folder')
    self.folder = self.portal['test-folder']
    self.request.set('URL', self.folder.absolute_url())
    self.request.set('ACTUAL_URL', self.folder.absolute_url())

def test_view(self):
    view = self.collection.restrictedTraverse('@@RSS')
    self.assertTrue(view())
    self.assertEquals(view.request.response.status, 200)
```

ComponentLookupError

If a view can not be looked up on a particular context, Plone will raise a ComponentLookupError (because views are multi-adapters), e.g.:

```
ComponentLookupError: ((<PloneSite at /plone>, <HTTPRequest, URL=http://nohost/plone>), <InterfaceCla
```

This can be solved for instance by providing a browser layer that has been missing:

```
def setUp(self):
    self.request = self.layer['request']
    from zope.interface import directlyProvides
    directlyProvides(self.request, IMyCompanyContenttypes)
    ...
```

2.5 Part 5: Testing Generic Setup

2.5.1 Template

test_setup.py:

```
from Products.CMFCore.utils import getToolByName
import unittest2 as unittest
from collective.mypackage.testing import \
    COLLECTIVE_MYPACKAGE_INTEGRATION_TESTING


class TestExample(unittest.TestCase):
```

```
    layer = COLLECTIVE_MYPACKAGE_INTEGRATION_TESTING

    def setUp(self):
        self.app = self.layer['app']
        self.portal = self.layer['portal']
        self.qi_tool = getToolByName(self.portal, 'portal_quickinstaller')

    def test_product_is_installed(self):
        """ Validate that our products GS profile has been run and the product
            installed
        """
        pid = 'puxam.policy'
        installed = [p['id'] for p in self.qi_tool.listInstalledProducts()]
        self.assertTrue(pid in installed,
                        'package appears not to have been installed')
```

2.5.2 Dependencies

Test if dependencies have been installed:

```
def test_product_is_installed(self):
    """ Validate that our products GS profile has been run and the product
        has been installed.
    """
    pid = 'collective.mailchimp'
    installed = [p['id'] for p in self.qi_tool.listInstalledProducts()]
    self.assertTrue(
        pid in installed,
        "The package '%s' appears not to have been installed." % pid')
```

setup.py:

```
install_requires=[
  'setuptools',
  'collective.mailchimp',
],
```

profiles/default/metadata.xml:

```
<?xml version="1.0"?>
<metadata>
  <version>1</version>
  <dependencies>
    <dependency>profile-collective.mailchimp:default</dependency>
  </dependencies>
</metadata>
```

2.5.3 Javascript Registration

Test if a Javascript file has been registered:

```
def test_js_available(self):
    jsreg = getToolByName(self.portal, 'portal_javascripts')
    script_ids = jsreg.getResourceIds()
    self.assertTrue('BarackSlideshow.js' in script_ids)
```

2.5.4 CSS Registration

Test if a CSS file has been registered:

```
def test_mailchimp_css_available(self):
    cssreg = getToolByName(self.portal, "portal_css")
    stylesheets_ids = cssreg.getResourceIds()
    self.assertTrue(
        '++resource++collective.mailchimp.stylesheets/mailchimp.css'
        in stylesheets_ids
    )
```

Test if a CSS has been enabled in the CSS registry:

```
def test_mailchimp_css_enabled(self):
   cssreg = getToolByName(self.portal, "portal_css")
   self.assertTrue(
       cssreg.getResource(
           '++resource++collective.mailchimp.stylesheets/mailchimp.css'
       ).getEnabled()
   )
```

2.5.5 Layer registered

interfaces.py:

```
from zope.interface import Interface

class IMyCompanyTheme(Interface):
    """"""
```

browserlayer.xml:

```
<layers>
  <layer
    name="mycompany.theme"
    interface="mycompany.theme.interfaces.IMyCompanyTheme"
    />
</layers>
```

test_setup.py:

```
def test_barackslideshow_layer_available(self):
    from plone.browserlayer import utils
    from collective.barackslideshow.tests.layer import IBarackSlideshowLayer
    self.failUnless(IBarackSlideshowLayer in utils.registered_layers())
```

2.5.6 Exclude From Search

Exclude a content type from search:

```
def makeTypeSearchable(portal, type_id, searchable):
    ptool = getToolByName(portal, 'portal_properties')
    blacklisted = list(ptool.site_properties.getProperty('types_not_searched'))
    if searchable and type_id in blacklisted:
        blacklisted.remove(type_id)
    elif not searchable and type_id not in blacklisted:
```

```
        blacklisted.append(type_id)
    ptool.site_properties.manage_changeProperties(
        types_not_searched=blacklisted)

makeTypeSearchable(portal, 'Image', searchable=False)
```

Test:

```
def test_exclude_images_from_search(self):
    self.assertTrue(
        'Image' in \
        self.ptool.site_properties.getProperty("types_not_searched"))
```

2.5.7 Resource Directories

test_setup.py:

```
def test_resources_directory(self):
    self.assertTrue(
        self.portal.restrictedTraverse(
            "++theme++dkg.contenttypes/medical-information.png"
        )
    )
```

configure.zcml:

```
<plone:static
  type="theme"
  directory="resources"
  />
```

2.5.8 Image

Test:

```
def test_method_render_grafik(self):
    self.portal.mi.eb.invokeFactory('grafik', 'text1')
    image_file = os.path.join(os.path.dirname(__file__), u'logo.jpg')
    self.portal.mi.eb.text1.grafik = NamedBlobImage(
    data=open(image_file, 'r').read(),
    contentType='image/jpg',
    filename=u'logo.jpg'
    )
    self.assertTrue(self.portal.mi.eb.text1.render())
```

Test if code is run as test

if self.request['URL'] == 'http://nohost': # test run

2.5.9 Catalog Index

Test if catalog index 'total_comments' has been installed:

```
def test_catalog_index_total_comments_installed(self):
    catalog = getToolByName(self.portal, "portal_catalog")
    self.assertTrue(
        'total_comments' in
        catalog.indexes()
    )
```

profiles/default/catalog.xml:

```
<?xml version="1.0"?>
<object name="portal_catalog">

  <index name="total_comments" meta_type="FieldIndex">
    <indexed_attr value="total_comments"/>
  </index>

</object>
```

2.5.10 Catalog Metadata

Test if catalog metadata has been installed:

```
def test_catalog_metadata_installed(self):
    self.portal.invokeFactory('Document',
                              'doc')
    self.portal.article.catchword = "Foo"
    self.portal.article.reindexObject()
    self.assertTrue('catchword' in self.catalog.schema())
    result = self.catalog.searchResults(
        path='/'.join(self.portal.article.getPhysicalPath()))
    self.assertTrue(len(result), 1)
    self.assertEquals(result[0].catchword, "Foo")
```

profiles/default/catalog.xml:

```
<?xml version="1.0"?>
<object name="portal_catalog" meta_type="Plone Catalog Tool">
  <index name="autor_in" meta_type="FieldIndex">
    <indexed_attr value="autor_in" />
  </index>
 <column value="autor_in" />
</object>
```

2.5.11 Searchable Index

Test if index is searchable:

```
def test_subjects_searchable(self):
    self.folder.invokeFactory("Document", "doc1")
    doc1 = self.folder.doc1
    doc1.setSubject([u"Python", u"Pyramid"])
    doc1.reindexObject()
    result = self.catalog.searchResults(dict(
        SearchableText = "Python"
        ))
```

```
    self.assertTrue(len(result), 1)
    self.assertTrue(result[0].title, "doc1")
```

2.5.12 Hide content type from navigation

Test if content type is hidden from navigation:

```
def test_hide_types_form_navigation(self):
    navtree_properties = self.portal.portal_properties.navtree_properties
    self.assertTrue(navtree_properties.hasProperty('metaTypesNotToList'))
    self.assertTrue('mycompany.membership.emailresetter' in
        navtree_properties.metaTypesNotToList)
    self.assertTrue('mycompany.membership.member' in
        navtree_properties.metaTypesNotToList)
    self.assertTrue('mycompany.membership.passwordresetter' in
        navtree_properties.metaTypesNotToList)
    self.assertTrue('mycompany.membership.registrator' in
        navtree_properties.metaTypesNotToList)
```

profiles/default/propertiestool.xml:

```
<?xml version="1.0"?>
<object name="portal_properties" meta_type="Plone Properties Tool">
 <object name="navtree_properties" meta_type="Plone Property Sheet">
  <property name="title">NavigationTree properties</property>
  <property name="metaTypesNotToList" type="lines">
   <element value="mycompany.membership.emailresetter"/>
   <element value="mycompany.membership.passwordresetter"/>
   <element value="mycompany.membership.registrator"/>
  </property>
 </object>
</object>
```

2.5.13 Do not search content type

Test if content type is excluded from search:

```
def test_types_not_searched(self):
    types_not_searched = self.portal.portal_properties\
        .site_properties.types_not_searched
    self.assertTrue('mycompany.membership.emailresetter'
                    in types_not_searched)
    self.assertTrue('mycompany.membership.passwordresetter'
                    in types_not_searched)
    self.assertTrue('mycompany.membership.registrator'
                    in types_not_searched)
```

profiles/default/propertiestool.xml:

```
<?xml version="1.0"?>
<object name="portal_properties">
  <object name="site_properties">
    <property name="types_not_searched" purge="false">
      <element value="mycompany.membership.emailresetter"/>
      <element value="mycompany.membership.passwordresetter"/>
      <element value="mycompany.membership.registrator"/>
```

```
    </property>
  </object>
</object>
```

2.5.14 Portal Actions

Test if portal actions have been added properly:

```
def test_actions(self):
    user_actions = self.portal.portal_actions.user
    self.assertTrue("preferences" in user_actions.objectIds())
    self.assertTrue('@@my-profile' in user_actions.preferences.url_expr)
    self.assertEquals(user_actions.preferences.visible, True)
```

profiles/default/actions.xml:

```
<?xml version="1.0"?>
<object name="portal_actions"
   xmlns:i18n="http://xml.zope.org/namespaces/i18n">
 <object name="user">
  <object name="preferences" meta_type="CMF Action" i18n:domain="mycompany.membership">
   <property name="title" i18n:translate="">Preferences</property>
   <property name="description" i18n:translate=""></property>
   <property
      name="url_expr">string:${globals_view/navigationRootUrl}/@@my-profile</property>
   <property name="icon_expr"></property>
   <property name="available_expr">python:member is not None</property>
   <property name="permissions">
    <element value="View"/>
   </property>
   <property name="visible">True</property>
  </object>
 </object>
</object>
```

2.5.15 Enable user folder

Test if user folder has been enabled:

```
self.mtool = self.portal.portal_membership
self.assertEquals(self.mtool.memberareaCreationFlag, 1)
self.assertEquals(self.mtool.memberarea_type, 'mycompany.membership.member')
self.assertEquals(self.mtool.getMembersFolder().absolute_url(),
                  'http://nohost/plone/autoren')
```

setuphandlers.py:

```
membership_tool.membersfolder_id = MEMBERS_FOLDER_ID
logger.info("Members folder set up: %s\n" % MEMBERS_FOLDER_ID)

# Configure member areas
membership_tool.setMemberAreaType(MEMBER_AREA_TYPE)
logger.info("Member area type: %s\n" % MEMBER_AREA_TYPE)

membership_tool.setMemberareaCreationFlag()
logger.info("Member area creation active\n")
```

2.5.16 Workflow

Test if workflow has been installed:

```
def test_workflows_installed(self):
    """Make sure both comment workflows have been installed properly.
    """
    self.assertTrue('one_state_workflow' in
                    self.portal.portal_workflow.objectIds())
    self.assertTrue('comment_review_workflow' in
                    self.portal.portal_workflow.objectIds())
```

Test default workflow for a certain content type:

```
def test_default_workflow(self):
    """Make sure one_state_workflow is the default workflow.
    """
    self.assertEqual(('one_state_workflow',),
                     self.portal.portal_workflow.getChainForPortalType(
                         'Discussion Item'))
```

2.5.17 Users and Groups

Test that a user has been added:

```
def test_users_installed(self):
    pas = getToolByName(self.portal, 'acl_users')
    user_ids = [x['login'] for x in pas.searchUsers()]
    self.assertTrue('john' in user_ids)
```

setuphandlers.py:

```
def setupGroups(portal):
    acl_users = getToolByName(portal, 'acl_users')
    if not acl_users.searchGroups(name='Editorial'):
        gtool = getToolByName(portal, 'portal_groups')
        gtool.addGroup('Editorial', roles=[])
```

Test that a group has been added:

```
def test_editorial_group_installed(self):
    self.assertTrue(
        'Editorial' in self.utool.source_groups.getGroupNames())
```

2.5.18 Roles

test_setup.py:

```
def test_mycompany_site_administrator_role_installed(self):
    self.assertTrue(
        "MyCompany Site Administrator" in self.portal.valid_roles())
```

profiles/default/roles.xml:

```
<?xml version="1.0"?>
<rolemap>
  <roles>
```

```
    <role name="Freitag Site Administrator" />
  </roles>
</rolemap>
```

2.5.19 Mock Mailhost

Mock Mailhost:

```
from zope.component import getSiteManager

from Products.MailHost.interfaces import IMailHost
from Products.CMFPlone.tests.utils import MockMailHost


class EasyNewsletterTests(unittest.TestCase):

    layer = EASYNEWSLETTER_INTEGRATION_TESTING

    def setUp(self):
        # Set up a mock mailhost
        self.portal._original_MailHost = self.portal.MailHost
        self.portal.MailHost = mailhost = MockMailHost('MailHost')
        sm = getSiteManager(context=self.portal)
        sm.unregisterUtility(provided=IMailHost)
        sm.registerUtility(mailhost, provided=IMailHost)
        # We need to fake a valid mail setup
        self.portal.email_from_address = "portal@plone.test"
        self.mailhost = self.portal.MailHost

    def test_send_email(self):
        self.assertEqual(len(self.mailhost.messages), 1)
        self.assertTrue(self.mailhost.messages[0])
        msg = str(self.mailhost.messages[0])
        self.assertTrue('To: john@plone.test' in msg)
        self.assertTrue('From: portal@plone.test' in msg)
```

2.5.20 Versioning

(Dexterity/plone.app.versioningbehavior only)

profiles/default/types/MyCustomType.xml:

```
<property name="behaviors">
  <element value="plone.app.versioningbehavior.behaviors.IVersionable" />
</property>
```

Test:

```
def test_versioning_behavior_enabled(self):
    self.portal.mi.sec.tc.invokeFactory('AudienceText', 'text1')
    from plone.app.versioningbehavior.behaviors import IVersioningSupport
    self.assertTrue(
        IVersioningSupport.providedBy(self.portal.mi.sec.tc.text1)
    )
```

profiles/default/repositorytool.xml:

```
<?xml version="1.0"?>
<repositorytool>
  <policymap>
    <type name="MyCustomType">
      <policy name="at_edit_autoversion"/>
      <policy name="version_on_revert"/>
    </type>
  </policymap>
</repositorytool>
```

Test:

```
def test_versioning_enabled(self):
    self.portal.mi.sec.tc.invokeFactory('AudienceText', 'text1')
    repository_tool = getToolByName(self.portal, "portal_repository")
    self.assertTrue(
        repository_tool.isVersionable(self.portal.mi.sec.tc.text1)
    )
    self.assertTrue(
        repository_tool.supportsPolicy(
            self.portal.mi.sec.tc.text1,
            'at_edit_autoversion'
        )
    )
```

2.6 Part 5: Testing Viewlets

Note: See https://github.com/plone/plone.app.discussion/blob/master/plone/app/discussion/tests/test_comments_viewlet.py

2.7 Part 6: Testing Portlets

Note: See https://github.com/collective/collective.mailchimp/blob/master/collective/mailchimp/tests/test_portlet.py

2.8 Part 7: Robot Framework

2.8.1 Robot Framework Setup

Add plone.app.robotframework to the package dependencies in setup.py:

```
    extras_require={
        'test': [
            'plone.app.testing',
            'plone.app.contenttypes',
            'plone.app.robotframework[debug,ride,reload]',
        ],
    },
```

Add plone.app.robotframework to your buildout configuration:

```
[robot]
recipe = zc.recipe.egg
eggs =
    Pillow
    ${test:eggs}
    plone.app.robotframework[debug,ride,reload]
```

Add a robot framework testing fixture to your test setup:

```
PLONETRAINING_TESTING_ACCEPTANCE_TESTING = FunctionalTesting(
    bases=(
        PLONETRAINING_TESTING_FIXTURE,
        REMOTE_LIBRARY_BUNDLE_FIXTURE,
        z2.ZSERVER_FIXTURE
    ),
    name='PlonetrainingTestingLayer:AcceptanceTesting'
)
```

Add a python file that automatically looks up all your robot tests in the ‘robots folder and runs them within your test suite:

```
# -*- coding: UTF-8 -*-
from plonetraining.testing.testing import PLONETRAINING_TESTING_ACCEPTANCE_TESTING  # noqa
from plone.app.testing import ROBOT_TEST_LEVEL
from plone.testing import layered

import os
import robotsuite
import unittest


def test_suite():
    suite = unittest.TestSuite()
    current_dir = os.path.abspath(os.path.dirname(__file__))
    robot_dir = os.path.join(current_dir, 'robot')
    robot_tests = [
        os.path.join('robot', doc) for doc in os.listdir(robot_dir)
        if doc.endswith('.robot') and doc.startswith('test_')
    ]
    for robot_test in robot_tests:
        robottestsuite = robotsuite.RobotTestSuite(robot_test)
        robottestsuite.level = ROBOT_TEST_LEVEL
        suite.addTests([
            layered(
                robottestsuite,
                layer=PLONETRAINING_TESTING_ACCEPTANCE_TESTING
            ),
        ])
    return suite
```

Note: robottestsuite.level assign all your robot test to a higher zope.testrunner test level. That means that your robot tests are not run by default (e.g. if you run ‘bin/test’. In order to run your robot tests you have to tell the zope.testrunner to run all test level (e.g. with ‘bin/test –all). This way you can exclude the long-running robot test when running your other tests.

Add a first robot test:

```
# ============================================================================
# EXAMPLE ROBOT TESTS
# ============================================================================
#
# Run this robot test stand-alone:
#
#  $ bin/test -s plonetraining.testing -t test_example.robot --all
#
# Run this robot test with robot server (which is faster):
#
# 1) Start robot server:
#
# $ bin/robot-server plonetraining.testing.testing.PLONETRAINING_TESTING_ACCEPTANCE_TESTING
#
# 2) Run robot tests:
#
# $ bin/robot src/plonetraining/testing/tests/robot/test_example.robot
#
# See the http://docs.plone.org for further details (search for robot
# framework).
#
# ============================================================================

*** Settings *****************************************************************

Resource  plone/app/robotframework/selenium.robot
Resource  plone/app/robotframework/keywords.robot

Library  Remote  ${PLONE_URL}/RobotRemote

Test Setup  Open test browser
Test Teardown  Close all browsers


*** Test Cases ***************************************************************

Scenario: As a member I want to be able to log into the website
  [Documentation]  Example of a BDD-style (Behavior-driven development) test.
  Given a login form
   When I enter valid credentials
   Then I am logged in

Scenario: Capture Screenshot of the Login Form
  [Tags]  screenshot
  Go To  ${PLONE_URL}/login_form
  Capture Page Screenshot  filename=login_form.png


*** Keywords *****************************************************************

# --- Given ------------------------------------------------------------------

a login form
  Go To  ${PLONE_URL}/login_form
  Wait until page contains  Login Name
  Wait until page contains  Password
```

```
# --- WHEN -------------------------------------------------------------------

I enter valid credentials
  Input Text  __ac_name  admin
  Input Text  __ac_password  secret
  Click Button  Log in


# --- THEN -------------------------------------------------------------------

I am logged in
  Wait until page contains  Site Map
  Page should contain  You are now logged in
  Page should contain  admin
```

2.8.2 Run Robot Tests

Start the robot-server:

```
$ bin/robot-server --reload-path src plonetraining.testing.testing.PLONETRAINING_TESTING_ACCEPTANCE_T
```

Note: –reload-path will automatically reload your changes to Python code, so you don't have to restart the robot server.

Run the robot tests:

```
$ bin/robot src/plonetraining/testing/tests/robot/test_example.robot
```

You can also run the robot test 'stand-alone' without robot-server. Though, this will take more time:

```
$ bin/test -t test_example --all
```

2.8.3 Robot Framework Report

Robot Framework creates HTML outputs that allows you to investiate test runs and failures. By default, plone.app.robotframework will write three files (log.html, output.xml and report.html into your buildout directory). Open log.html for a full report.

If tests fail, robot framework will automatically create screenshots, to make it easiert to find the problem.

2.8.4 Debugging Robot Framework Tests

Robot Framework allows you to debug tests in an interactive manner. Just add 'Debug' to any part of the test.

Say you want to debug the 'I am logged in' keyword in the test_example.robot test. Add 'Debug' to the end of the keyword:

```
I am logged in
  Wait until page contains  Site Map
  Page should contain  You are now logged in
  Page should contain  admin
  Debug
```

Then run the test with:

```
$ bin/robot src/plonetraining/testing/tests/robot/test_example.robot
```

The test will open the browser as usual and then stop with an interactive robot shell at the point where you added the 'Debug':

```
==============================================================================
Test Example
==============================================================================
Scenario: As a member I want to be able to log into the website ::... ...
>>>>> Enter interactive shell, only accepted plain text format keyword.
>
```

You can now type in robot keywords, e.g.:

```
> Page should contain  Home
>
```

You can exit the debugger by typing 'exit' and hitting return:

```
> exit
...
```

Note: The robot debugger currently can not handle special characters and you can not assign and use robot variables. This will hopefully change in the future.

2.8.5 Tagging Robot Framework Tests

Robot Framework allows us to tag tests with one or more tags:

```
Scenario: As a site administrator I can add a Task
  [Tags]  current
  Given a logged-in site administrator
    and an add task form
   When I type 'My Task' into the title field
    and I submit the form
   Then a task with the title 'My Task' has been created
```

You can then just run robot tests with that tag by providing '-i <tagname>':

```
$ bin/robot -i current src/plonetraining/testing/tests/robot/test_example.robot
```

2.8.6 Autologin

plone.app.robotframework comes with keywords that allows you to log as different users:

```
a logged-in site administrator
  Enable autologin as  Site Administrator
```

This will login the current user with the 'Site Administrator' role.

It is also possible to login with multiple roles:

```
Enable autologin as  Site Administrator  Reviewer
```

You can logout with:

```
Disable Autologin
```

Or set the username with:

```
Set Autologin Username  john
```

2.8.7 Create Content

plone.app.robotframework comes with keywords to create content:

```
Create content  type=Task  id=my-task  title=My Task
```

This creates a 'Task' content object with the id 'my-task' and the title 'My Task'.

2.8.8 Screenshots

You can capture screenshots during your robot framework tests that you can ,for instance, use in your docs:

Scenario: Capture Screenshot of the Login Form [Tags] screenshot Go To ${PLONE_URL}/login_form Capture Page Screenshot filename=login_form.png

---

Note: See http://datakurre.pandala.org/2013/04/generate-annotated-screenshots-with.html for how to generate annotated screenshots with Robot Framework.

2.8.9 Further Reading

- todo: how to write good robot framework tests.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search