How To Create Unit Tests

▼ Details

This tutorial explains how to create unit tests and by so how to make sure your business template will keep working from one upgrade to another. If you don't test your business template, it will sooner or later stop working and maybe even without you noticing when it happens.

Activate Portal Components

▼ Details

To test our template, we need to activate the **Portal Components** in our ERP5 instance which you use will use to run live test.

Before you activate the portal components, please check that following Business Templates are installed in your system : **erp5_jquery, erp5_jquery_ui, erp5_dhtml_style** .

Read first Dynamic Classes >> ZODB Components to get a basic idea. For security reasons, a Developer Role has been introduced which is not available through the UI. Only users with Developer Role can modify ZODB Components.

Now we will active the portal components. Similarly to Class Tool which requires creating a file in ERP5Type Product, you must edit **zope.conf** on the filesystem to add users to Developer Role. For instance, to add zope users to Developer Role:
%import Products.ERP5Type

developers zope

as shown in the screenshot.

After you edit zope.conf, **restart the modified partition and process**, as shown in the screenshot.

Now you can **access to portal components** in your ERP5 instance: **erp5/portal_components**. Have a look at this list which includes different types of portal components. We will than add a new Test Component in order to run the test for our template.

Add Test Component

▼ Details

In portal_components, create a new test named "testDiscussionThread" (in ERP5, Unit test names always start by an underscore test prefix) from **Action menu >> Add Test Component**. The tool will then generate a new Test from a template. Have a quick look at the generated code to understand how it works.

ERP5TypeTestCase is extending the zope class ZopeTestCase.PortalTestCase, which is itself extending the common **unittest.TestCase**.

See the documentation: http://docs.python.org/library/unittest.html#testcase-objects.

After you are in the new Test Component page, edit the following information:
ID: test.erp5.testDiscussionThread
Reference: testDiscussionThread (please follow the naming convention inhttps://www.erp5.com/_user/support/developer-Dynamic.Class.Summary/#id-naming-convention that you should already read).
Description: Unit test for forum template.

We will first save this Test as it is, then see how we can run Tests from the ERP5 interface, and then we will come back to this Test to improve it.

Validate the this test component before you go to next step.

Launch tests from portal_components

▼ Details

In the page of **/erp5/portal_components**, click **Action menu>>Run Live Tests**.

In the next dialog, be sure to write the name of the test you want to run, **testDiscussionThread**.

Click **"Live Tests"** button to launch the test.

What happens when a Test fails?

▼ Details

The screen you get shows you the progress of the Tests, and the output of the test runner.

If you remember well, the simple Test we created had a failing assertion. You are then not surprised to see the traceback on the screen showing you which tests of the Test Suite fail.

Now that you know how to launch Tests from the ERP5 web interface, we will improve our testDiscussionThread to test the behaviour of our application.

Edit Test header

▼ Details

Go back to the new test component **testDiscussionThread** you created where you can edit the test for forum template. (Full url is **/erp5/portal_components/test.erp5.testDiscussionThread**).

We are going to edit and expand this Test. (For a better editing experience you can install the business templates "erp5_ace_editor" or "code_mirror". Then in My Favourites menue go to Preferences and then in the User Interface tab you can activate either Ace Editor or Code Mirror as Source Code Editor.)

First, you should **rename** the Test class to**TestDiscussionThread**.

Then we are going to change the Parent class. ERP5TestCase is very useful, but SecurityTestCase extends this class with very helpful security-related assertions. Change the import too, SecurityTestCase can also be found in Products.ERP5Type.tests

Set the test title to TestDiscussionThread.

Specify the dependencies for this test. We need the erp5_forum_tutorial dependencies and the erp5_forum_tutorial itself ('erp5_base', 'erp5_web', erp5_ingestion_mysql_innodb_catalog', 'erp5_ingestion', 'erp5_dms', 'erp5_forum_tutorial')

Double-check that the result code is similar to the code provided in the slide.

Test setup

▼ Details

We will now define the methods needed for Test set up. The first method is **afterSetUp**, which is ran before each test. In this method, we will create all the users we will need for the test. The main helper for this task is **ERP5TestCase.createSimpleUser**.

Since afterSetUp is launched before every test, if you add any data, you have to check if the data is not already there.

Create helper functions if necessary

▼ Details

It is important to keep your Test code clean and readable.

As a general guideline, it's a good thing to extract a recurrent step in a method properly named to shorten code and simplify it.

Here, we create a simple function allowing us to create a Thread by calling the **DiscussionThreadModule_addThread** script we created earlier.

Note the **batch_mode** parameter. This parameter was not added in **DiscussionThreadModule_addThread** we created. We will come back to it later.

Write the first Test method (1)

▼ Details

We will now replace the dummy **test_01_sampleTest** with something more useful.

Copy the provided code instead of the dummy test.

Please spend a few minutes to read the code we're adding, to understand the test "Magic".

In particular, note the **assertUserCan** methods. Those security-related assertions are very useful for permission checks, and are defined in **SecurityTestCase**.

Write the first Test method (2)

▼ Details

Extend the test method with the provided code.

Code should be straightforward.

Note that when comparing objects, we are comparing their URLs, and not the objects themselves: in fact, we have no guarantee that thread.getParentValue() == self.forum_module; they represent the same URL, but they might be different instances of the same object.

Write the first Test method (3)

▼ Details

Extend the test method with the provided code.

Note, once again, the **batch_mode** parameter that we will need to add later to**DiscussionThreadModule_addReply** script.

"transaction" is a module, that is why we added an "import transaction" at the top of the Test file.

"transaction.commit" saves the created objects in the Database.

As a guideline, it's a bad habit to commit after each operation: in a production environment, we try to group operations to reduce as much as possible database contention.

self.tic is a clock “tick”. It forces background activities to run.

As a guideline, it's a bad habit to force your test to tic unless needed. Activities run when the system load is low, asynchronously. Inserting systematic **tics** after each operation in a Test eliminates the asynchronous character of activities.

Here, for example we use the commit+tic stance only because the next lines will perform a Catalog search (A SQL search, in fact). And we need to ensure that data has been committed before doing this search, otherwise new objects will not show up.

Write the first Test method (4)

▼ Details

Complete the test method with the provided code.

The Catalog access we were mentioning in the previous slide is the searchFolder call.

Edit portal type

▼ Details

We need to add text_content in Searchable Text Property IDs to search text_content of the post that we created in the database.

Access for visitors

▼ Details

Complete the test method with the provided code.

The method checks for permissions for visitors and users having no access on the forum module.

HTTP_OK code needs to be imported and that is why we need to put**from httplib import OK as HTTP_OK** at the beginning of the

test code.

"self.publish" is a handler to access the HTTP response returned by a specific object when published.

Visitors cannot write

▼ Details

Add this new test method.

It checks that visitors can't write to an existing thread or create a new thread.

Moderation

▼ Details

Add this new test method.

It checks that admin have the rights to close/unclose a thread.

Again, a commit is done after each transition only because it is necessary:
It does not make sense to check for permissions on an object if the object state has not been properly saved yet.

Test Hidden state

▼ Details

Add this new test method.

It checks that permissions are set properly for the “Hidden” state.

Test new reply if not Thread owner

▼ Details

Add this new test method.

It checks that permissions are set properly when an user is not the Owner of a Thread: here, since "forum_user" creates the thread, he has two roles, Author and Owner.

"another_forum_user", in the other hand has only the Author role.

We are done editing testDiscussionThread, you can now save it.

Users cannot moderate

▼ Details

Add this new test method.

It checks that users can't close a thread if they're not admin.

We are done editing testDiscussionThread, you can now save it.

batch_mode variable

▼ Details

Remember the batch_mode argument we used? We need to edit the Python scripts, in the skins folder, to add it.

Its use is simple: we add it as a keyword argument defaulting to False, meaning that default behaviour of or User Interface is unchanged. However, when set to True because the script is run from unit tests instead of from the web, the scripts will return the objects they created instead of redirecting to the view of those objects.

Edit the scripts as shown.

The **DiscussionThreadModule_addReply** must returns the post **DiscussionThreadModule_addThread** must returns the

thread it created.

Run the correct test.