


FarCry Plugins : testMXUnit

This page last changed on Feb 18, 2010 by [rob](#).

 If you are reading this from within the plugin as the README.pdf, and any of this fails for you, please see <http://docs.farcrycms.org/display/FCPLUG/testMXUnit> for the latest information about testMXUnit.

Prerequisite

FarCry unit tests uses <http://mxunit.org/> with a wrapper that adds in FarCry hooks and webskin compatibility. You can get the latest copy of the plugin by checking out the code from the following SVN repository URL:

```
http://modius.svn.cvsdude.com/farcry/plugins/testMXUnit/trunk
```

Once you have the code checked out into your plugins folder, you will need to add an Alias (in Apache) or a Virtual Directory in IIS. The path should be similar to the following:

```
Alias /mxunit /Users/rob-rohan/Sites/blarg/Yadda/plugins/testMXUnit/www/mxunit
```

 Note the path goes in one level deeper than most FarCry alias'.

You will then need to add the plugin to your plugin list in your *www/farcryConstructor.cfm* file. For example:

```
<cfset THIS.plugins =  
"farcrypoll,farcryradiostar,farcrycms,farcryverity,farcrycfximage,testMXUnit" />
```

You will then need to deploy the *mxTest* ContentType. You can do this in the webtop in the *Admin* tab. Select *Developer Utilities* from the pull down, and then click on the *Types* link under the *COAPI Tools* heading. Scroll down until you see *mxTest* and click the *deploy* link.

Finally, you will need to run *updateapp* on your application.

You are ready to write and run tests.

Running Tests

To run unit tests, log into the webtop and choose the *Admin* tab. From the pull down select *Testing*. This will display two menu options *Run Tests* and *Configure Tests*.

Configure Tests

You select which tests you would like to run on the *Configure Tests* screen. After you have ticked the boxes next to the CFCs you'd like to run (and click *Save Configuration*), you can then proceed to *Run Tests*.

Run Tests

Run Tests Does what it says on the tin - it runs the tests selected from the *Configure Tests* screen.

All tests have been completed (failures: 1 passes: 1)			
ExampleTest	failTest	15ms	Failed (more detail)
ExampleTest	passTest	0ms	Passed

Writing Tests

Create a directory named *tests* in the root level of your plugin or project. In other words, the directory should be at the same level as *packages*, *tags*, *webskin*, etc. Within the test directory you will create CFCs that hold all your tests. By creating your test CFCs in this directory, they will automatically be selectable in the *Configure Tests* screen described above.

Simple tests

The Test CFCs must extend *mxunit.framework.TestCase*. There is a file called *ExampleTest.cfc* in the *testMXUnit* plugin that you can use as a template. The contents of that file at the time of this writing are the following:

```
<cfcomponent extends="mxunit.framework.TestCase">
<!--- setup and teardown --->
<cffunction name="setUp" returnType="void" access="public">
<!--- Any code needed to return your environment to normal goes here --->
</cffunction>

<cffunction name="tearDown" returnType="void" access="public">
<!--- Any code needed to return your environment to normal goes here --->
</cffunction>

<!--- //////////////////////////////////////// --->

<cffunction name="passTest" returnType="void" access="public">
<!--- Any code needed to return your environment to normal goes here --->
<cfset assertEquals(true,true) />
</cffunction>

<cffunction name="failTest" returnType="void" access="public">
<!--- Any code needed to return your environment to normal goes here --->
<cfset assertEquals("yes", "no") />
</cffunction>

</cfcomponent>
```

Once your tests are written, you can add them your *testSuite.addAll()* in your projects *testMXUnit displayPageStandard.cfm* file.

Since the tests are running from within the context of FarCry, all of the FarCry tags and functions are available from within your test. While it is not pedantically correct to write tests that use items outside of the unit test and method you are testing, it is often quite helpful (and sometimes required).

For documentation and tutorials on MXUnit see <http://mxunit.org/doc/index.cfm>

FarCry tests

Some tests require integration to a higher degree with the FarCry framework. For example, creating a bunch of ContentTypes and then rolling those back when the test is done. For tests that require this kind of integration, have your test CFCs extend *farcry.plugins.testmxunit.tests.FarcryTestCase*. This component provides functionality for creating and destroying test data. These tests work the same way as basic tests, but the following functions are available.

Temporary data

These functions create data in the database. The *FarcryTestCase* automatically removes this data once the test is complete. Generally these are used in *setUp* and *tearDown*. *super.setUp()* must be called at the start of the overriding *setUp* function, and *super.tearDown()* must be called at the end of the overriding *tearDown* function.

Function	Description
----------	-------------

createTemporaryObject	Creates a temporary object in the database. One argument for each property. Include a <i>categories</i> to attach categories in refCategories. Object is automatically removed at the end of the test.
createTemporaryCategory	Requires <i>alias</i> , <i>parentid</i> , <i>categoryid</i> , and <i>categoryLabel</i> . Category (and references) is automatically removed at the end of the test.
pinCategories	Takes a list <i>category</i> and remembers them. At the end of each test all changes to these categories (and the object references) are reverted.
pinObjects	Takes <i>typename</i> , and optionally <i>timeframe</i> (seconds) which restricts match to objects created recently. Objects are reverted, re-added, or deleted as necessary to restore the snapshot.
pinScope	Takes <i>variable</i> defining a valid isdefined argument. If the variable exists it is duplicated and reset at the end of the test. If it does not, structdelete is used to remove the final part. e.g. request.a.b => structdelete("request.a","b"). Only useful for basic variable structures.

Assertions

These functions are FarCry specific assertions.

Function	Description
assertContentTypeExists	Asserts that <i>typename</i> can be instantiated. Optional <i>message</i> for failure..
assertObjectExists	Asserts that an object matching <i>typename</i> , optional <i>timeframe</i> (i.e. created in the last X seconds), and miscellaneous properties (passed as corresponding arguments), exists. Supports array properties (exact set match, but not sequence), and any property that the database can handle as a varchar. Optional <i>message</i> for failure..
assertNotObjectExists	Asserts that an object matching <i>typename</i> , optional <i>timeframe</i> (i.e. created in the last X seconds), and miscellaneous properties (passed as corresponding arguments), exists. Does NOT support array properties, and properties must be able to handled as varchar. Optional <i>message</i> for failure..
assertVariable	Asserts that <i>variable</i> exists and is equal to <i>value</i> . Only supports simple values. Optional <i>message</i> for failure.