Artículo
Timothy Leavitt · 24 mar, 2020 Lectura de 5 min

Open Exchange

Unit Tests and Test Coverage in the InterSystems Package Manager

This article will describe processes for running unit tests via the InterSystems Package Manager (aka IPM - see https://openexchange.intersystems.com/package/InterSystems-Package-Manager-1), including test coverage measurement (via https://openexchange.intersystems.com/package/Test-Coverage-Tool).

Unit testing in ObjectScript

There's already great documentation about writing unit tests in ObjectScript, so I won't repeat any of that. You can find the Unit Test tutorial
here: https://docs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=TUNTpreface

It's best practice to include your unit tests somewhere separate in your source tree, whether it's just "/tests" or something fancier. Within InterSystems, we end up using /internal/testing/unittests/ as our de facto standard, which makes sense because tests are internal/non-distributed and there are types of tests other than unit tests, but this might be a bit complex for simple open source projects. You may see this structure in some of our GitHub repos.

From a workflow perspective, this is super easy in VSCode - you just create the directory and put the classes there. With older server-centric approaches to source control (those used in Studio) you'll need to map this package appropriately, and the approach for that varies by source control extension.

From a unit test class naming perspective, my personal preference (and the best practice for my group) is:

UnitTest.<package/class being tested>[.<method/feature being tested>]

For example, if unit tests for method Foo in class MyApplication.SomeClass, the unit test class would be named UnitTest.MyApplication.SomeClass.Foo; if the tests were for the class as a whole, it'd just be UnitTest.MyApplication.SomeClass.

Unit tests in IPM

Making the InterSystems Package Manager aware of your unit tests is easy! Just add a line to module.xml like the following (taken from https://github.com/timleavitt/ObjectScript-Math/blob/master/module.xml - a fork of @Peter Steiwer 's excellent math package from the Open Exchange, which I'm using as a simple motivating example):

```
<Module>
  ...
  <UnitTest Name="tests" Package="UnitTest.Math" Phase="test"/>
</Module>
```

What this all means:

- The unit tests are in the "tests" directory underneath the module's root.
- The unit tests are in the "UnitTest.Math" package. This makes sense, because the classes being tested are in the "Math" package.
- The unit tests run in the "test" phase in the package lifecycle. (There's also a "verify" phase in which they

could run, but that's a story for another day.)

Running Unit Tests

With unit tests defined as explained above, the package manager provides some really helpful tools for running them. You can still set ^UnitTestRoot, etc. as you usually would with %UnitTest.Manager, but you'll probably find the following options much easier - especially if you're working on several projects in the same environment.

You can try out all of these by cloning the objectscript-math repo listed above and then loading it with zpm "load /path/to/cloned/repo/", or on your own package by replacing "objectscript-math" with your package names (and test names).

To reload the module and then run all the unit tests:

zpm "objectscript-math test"

To just run the unit tests (without reloading):

zpm "objectscript-math test -only"

To just run the unit tests (without reloading) and provide verbose output:

zpm "objectscript-math test -only -verbose"

To just run a particular test suite (meaning a directory of tests - in this case, all the tests in UnitTest/Math/Utils) without reloading, and provide verbose output:

zpm "objectscript-math test -only -verbose -DUnitTest.Suite=UnitTest.Math.Utils"

To just run a particular test case (in this case, UnitTest.Math.Utils.TestValidateRange) without reloading, and provide verbose output:

zpm "objectscript-math test -only -verbose -DUnitTest.Case=UnitTest.Math.Utils.TestValidateRange"

Or, if you're just working out the kinks in a single test method:

zpm "objectscript-math test -only -verbose -DUnitTest.Case=UnitTest.Math.Utils.TestValidateRange -DUnitTest.Method=TestpValueNull"

Test coverage measurement via IPM

So you have some unit tests - but are they any good? Measuring test coverage won't fully answer that question, but it at least helps. I presented on this at Global Summit back in 2018 - see https://youtu.be/nUSeGHwN5pc .

The first thing you'll need to do is install the test coverage package:

zpm "install testcoverage"

Note that this doesn't require IPM to install/run; you can find more information on the Open Exchange: https://openexchange.intersystems.com/package/Test-Coverage-Tool

That said, you can get the most out of the test coverage tool if you're also using IPM.

Before running tests, you need to specify which classes/routines you expect your tests to cover. This is important because, in very large codebases (for example, HealthShare), measuring and collecting test coverage for all of the files in the project may require more memory than your system has. (Specifically, gmheap for the line-by-line monitor, if you're curious.)

The list of files goes in a file named coverage.list within your unit test root; different subdirectories (suites) of unit tests can have their own copy of this to override which classes/routines will be tracked while the test suite is running.

For a simple example with objectscript-math,
see: https://github.com/timleavitt/ObjectScript-Math/blob/master/tests/UnitTest/coverage.list ; the user guide for the test coverage tool goes into further details.

To run the unit tests with test coverage measurement enabled, there's just one more argument to add to the command, specifying that TestCoverage.Manager should be used instead of %UnitTest.Manager to run the tests:

zpm "objectscript-math test -only -DUnitTest.ManagerClass=TestCoverage.Manager"

The output (even in non-verbose mode) will include a URL where you can view which lines of your classes/routines were covered by unit tests, as well as some aggregate statistics.

Next Steps

What about automating all of this in CI? What about reporting unit test results and coverage scores/diffs? You can do that too! For a simple example using Docker, Travis CI and codecov.io,
see https://github.com/timleavitt/ObjectScript-Math ; I'm planning to write this up in a future article that looks at a few different approaches.

#Integración continua #InterSystems Package Manager (IPM) #InterSystems IRIS #InterSystems IRIS for Health
Ir a la aplicación en InterSystems Open Exchange