Subject: Re: Unit Testing
Posted by Michael Galloy on Wed, 03 Jan 2007 23:16:46 GMT
View Forum Message <> Reply to Message

Robbie wrote:
> Thanks for the feedback. I think my key problem is that I haven't used
> unit testing in other languages before. I would like to make something
> which compliments the incremental compiling nature of IDL and I want to
> avoid writing a .pro file parser.
>
>> 1. Why isn't it object-oriented?
> I think that writing objects in IDL is quite clumsy. I guess I haven't
> had the need for unit tests to be based inside objects yet. I'm
> concerned that using OO would deviate from unit tests being short and
> sweet.

I'm including an example test (a rather silly one, for instructional purposes only) at the bottom of this post. I don't think it has much "extra fat."

>> 2. You say "The final aim of this project is to fully support xUnit
>> testing automation, including support for fixtures." How will you
>> support fixtures?
> I was thinking of getting fixtures to SetUp() and TearDown() a common
> block. I could also use keywords to do the same sort of thing. This is
> where I should probably be using OO.

I definitely like objects for this.

>> 3. How does "unitException" have access to local variables?
> I can't believe I missed that one! I should probably stick to using
> wrappers of CALL_PROCEDURE, CALL_FUNCTION and CALL_METHOD.
> unitExceptionFunction just doesn't roll off the tongue too well :)

I have a couple batch files that have error handling in them. Put "@error_is_pass" in your test if the test is supposed to cause an error.

>> 4. Why do the test names end in an ordinal? Why not
>> hashtable__testAdding, etc?
> I've developed a nasty habit of using ordinals in the suffix. I guess I
> shouldn't tempt anyone else to do the same thing. Any procedure,
> function or method with __test in it would become a unit test. Perhaps
> I should allow unitSearch specify exclusions.

No big deal, I'm going to try to polish what I have up a bit (and add some documentation) and get it posted on my website soon. I'll let you know when it's up.

Mike
--
www.michaelgalloy.com

Here are the tests:

```
;+
; This test fails because the assertion is wrong.
;-
function findgentest::test1
  a = findgen(5)
  assert, n_elements(a) eq 6, 'Wrong number of elements'

  return, 1
end


;+
; This test should pass the assertion and return 1 (i.e. success).
Tests can
; also return 0 or generate an error to indicate failure.
;-
function findgentest::test2
  a = findgen(5)
  assert, array_equal(a, [0.0, 1.0, 2.0, 3.0, 4.0]), 'Correct elements'

  return, 1
end


;+
; This is a test that will pass because the code of the test is
supposed to
; cause an error. To do this kind of test, use the "error_is_pass"
batch file.
;-
function findgentest::test3
  @error_is_pass

  a = findgen('string')

  return, 1
end


;+
```

```
; This is a test that will fail on an io error because of the use of
the
; "error_is_fail" batch file. IO errors don't normally cause a test to
fail.
;-
function findgentest::test4
  @error_is_fail

  a = findgen('another_string')

  return, 1
end



;+
; Inherit from MGtestCase.
;
; @file_comments To create a test case just inherit from MGtestCase and
create
;               method with names that start with "test". This test
can be run
;               with the command: mgunit, cases='findgentest'
;-
pro findgentest__define
  define = { findgentest, inherits MGtestCase }
end
```