Subject: Unit Testing
Posted by Robbie on Wed, 03 Jan 2007 02:14:09 GMT
View Forum Message <> Reply to Message

Hi,

Are there any implementations of Unit Testing in IDL yet? I've put together a white paper for unit testing in IDL. It is fairly straight forward, and I've tried to make it easy to run tests of currently compiled code. However, there is no point implementing it if someone has a better idea of how to go about unit testing.

http://www.barnett.id.au/idl/UnitRun.html

Robbie

UnitRun is an adaptation of testing frameworks such as NUnit, JUnit and PyUnit. The final aim of this project is to fully support xUnit testing automation, including support for fixtures.
Unit test procedures
Unit tests are called using specially named test procedures.
Expected procedure names:

* hashtable__test
* hashtable__test0
* hashtable__test1
* hashtable__test0000445

A simple test case

unitSearch

```
pro hashtable__test1
  obj = obj_new('hashtable')
  obj -> Add, 'one', 1
  unitAssert, obj -> isContained('one')
  obj_destroy, obj
end
```

The unitSearch directive indicates that all subsequent test procedures should be included as unit tests. The unitAssert procedure reports the result of the test.

1. The unitAssert procedure reports the name of the test procedure

(see HELP, CALLS=calls)
  2. The unitAssert procedure reports <success> if the argument is greater than one
  3. The unitAssert procedure reports <fail> in any other circumstance including unhandled execptions

Running tests

reslove_routine, 'hashtable__test1'
unitRun

The unitRun procedure looks for all specially named test procedures. All test procedures are re-resolved. Only procedures with the unitSearch directive will be included in unit test.

unitRun, ['hashtable__test1','hashtable__test2']

The unitRun procedure can be used to manually call a sequence of test procedures. In this case procedures are called directly without paying attention to unitSearch.

unitRun, LOG_FILE='unitRun.log'

The unitRun procedure can dump the test results to a log file instead of dumping the results to the IDL command line.
Expected exceptions

```
pro hashtable__test2
  obj = obj_new('hashtable')
  obj -> Add, 'one', 1
  unitException, 'obj -> Add, 5, 6'
  obj_destroy, obj
end
```

The unitException procedure executes a single IDL statement which is expected to result in an exception. The unitException does not currently distinguish between message blocks or names.

  1. The unitException procedure reports <success> if an exeception was encountered
  2. The unitException procedure reports <fail> if no exception was encountered

---

Subject: Re: Unit Testing
Posted by Michael Galloy on Wed, 07 Feb 2007 16:28:08 GMT
View Forum Message <> Reply to Message

On Feb 6, 7:10 pm, "Qing" <c...@bigpond.net.au> wrote:
> These are very interesting stuff. How practical do you think this can
> be applied to a real program?
> For example, one writes a routine accepting a list of parameters and
> keywords. Inside the routine,
> there are many calls to other native IDL libraries and user routines,
> and the parameters/keywords
> passed can include constarts/arrays/structures/pointers/objects...
> Does anyone know if all native IDL libraries have been tested this
> way?

I think the framework posted here is practical for real routines. I have written tests for small sets of routines using this framework and haven't had any problems. That being said, definitely let me know if you have difficulties.

Mike
--
www.michaelgalloy.com

Subject: Re: Unit Testing
Posted by Robbie on Thu, 08 Feb 2007 12:34:09 GMT
View Forum Message <> Reply to Message

I think that the problem with applying unit testing to IDL is that a lot of us tend to fart about (http://cow.mooh.org/idl.shtml) . There is often no planning process, so there is no foresight about designing a test bed before writing the application.

As a rule of thumb, most arguments to a procedure are converted to float() on the first line of arithmetic.
I would argue that a good unit test does not have to iterate over all combinations of heap variable types.
Perhaps you could create a wrapper which cycles through all heap variable combinations.

Robbie
