Subject: Re: Leaking objects...
Posted by David Fanning on Wed, 27 Mar 2002 16:29:39 GMT
View Forum Message <> Reply to Message

Randall Skelton (rhskelto@atm.ox.ac.uk) writes:

> I gave some IDL object code to someone who has no idea what object
> programming is. Despite giving her a quick lesson on object life-cycles,
> she did the inevitable and directly overwrote an object, creating dangling
> pointers.
>
> My question is, how do I prevent this from happening in my code? This
> behavior seems a little fragile to me. Ideally, I would like obj_new to
> either block the creation of a new object or cleanup pre-existing objects.
> What I don't want to do is try and educate the user again... sorry mom.

You may *think* you want Obj_New to block the creation
of a new object, but I can assure you, you *don't*. :-)
Having only one of any particular kind of object would
be just a tad limiting, don't you think?

IDL is an inherently dangerous language. You really don't
have any choice but to make an attempt to educate the
people who use it. If we didn't believe this, why else
would we all be hanging out here on the newsgroup?

Cheers,

David

P.S. You might suggest to your user that she become
more familiar with HEAP_GC. That might help. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Leaking objects...
Posted by btupper on Wed, 27 Mar 2002 17:15:43 GMT
View Forum Message <> Reply to Message

On Wed, 27 Mar 2002 09:29:39 -0700, David Fanning <david@dfanning.com>
wrote:

>
> You may *think* you want Obj_New to block the creation
> of a new object, but I can assure you, you *don't*. :-)
> Having only one of any particular kind of object would
> be just a tad limiting, don't you think?
>
> IDL is an inherently dangerous language. You really don't
> have any choice but to make an attempt to educate the
> people who use it. If we didn't believe this, why else
> would we all be hanging out here on the newsgroup?

Hi,

Since I am from the brute-force programming school, I immediately thought 'What a great idea!' Having fiddled with this for a few minutes, now I can see David's point. But a lot can be learned from trying... now I know what that CAST keyword does.

Here's a hack that has half the behavior you describe.

If you set the destroy_previous keyword, the object will kill destroy any existing objects of the TEST class that are not the new object. However, if you do not set the /DESTROY_PREVIOUS keyword a new object will *not* be created, but you will lose your reference, since the failed object initialization returns a null object.

Ben

*****make an object

```
IDL> a = obj_new('test', findgen(20))
% Compiled module: TEST__DEFINE.
IDL> help, /heap
Heap Variables:
   # Pointer: 1
   # Object : 1

<ObjHeapVar23>  STRUCT    = -> TEST Array[1]
<PtrHeapVar24>  FLOAT     = Array[20]
IDL> help, a
A               OBJREF    = <ObjHeapVar23(TEST)>
```

*****try to make a new one of the same class

```
IDL> b = obj_new('test', findgen(30))
% TEST::INIT: An instance of this class already exists
IDL> help, a,b
A               OBJREF    = <ObjHeapVar23(TEST)>
B               OBJREF    = <NullObject>
IDL> help, /heap
Heap Variables:
   # Pointer: 1
   # Object : 1

<ObjHeapVar23>  STRUCT    = -> TEST Array[1]
<PtrHeapVar24>  FLOAT     = Array[20]


*****make a new object and destroy existing objects of the same class
*****note that a has been cleaned up
IDL> b = obj_new('test', findgen(30), /destroy_previous)
IDL> help, a,b
A               OBJREF    = <ObjHeapVar23>
B               OBJREF    = <ObjHeapVar26(TEST)>
IDL> help, /heap
Heap Variables:
   # Pointer: 1
   # Object : 1

<ObjHeapVar26>  STRUCT    = -> TEST Array[1]
<PtrHeapVar27>  FLOAT     = Array[30]


****** now try to redefine b (an existing object of class TEST
****** note that even though a new object is not created, the
****** null object defintion is returned to variable b
****** so you are really no further ahead

IDL> b = obj_new('test', findgen(40))
% TEST::INIT: An instance of this class already exists
IDL> help, b
B               OBJREF    = <NullObject>
IDL> help, /heap
Heap Variables:
   # Pointer: 1
   # Object : 1

<ObjHeapVar26>  STRUCT    = -> TEST Array[1]
<PtrHeapVar27>  FLOAT     = Array[30]
```

START HERE

```
pro test::cleanup

  ptr_free, self.data

end

function test::init, data, destroy_previous = destroy_previous

 objs = Obj_Valid(cast =1, count = count)
  For i = 0L, Count-1 Do Begin
   If Obj_Valid(objs[i]) EQ 1 Then Begin
   If OBJ_CLASS(objs[i]) EQ 'TEST' AND objs[i] NE self
Then Begin
    If KeyWord_Set(Destroy_Previous) Then Begin
     Obj_Destroy, objs[i]
    EndIf Else Begin
     message, 'An instance of this class
already exists', /info
     return, 0
    EndElse
   EndIf
  EndIf
 EndFor

  if n_params() ne 1 then begin
      message, 'Expecting 1 parameter', /info
      return, 0
  endif

  self.data = ptr_new(data)

  return, 1

end

pro test__define

   j = { test, data: ptr_new() }

end
```

END HERE