Subject: Leaking objects...
Posted by Randall Skelton on Wed, 27 Mar 2002 16:08:23 GMT
View Forum Message <> Reply to Message

Hi all,

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. A simplified example is:

The correct way:

```
IDL> a = obj_new('test', findgen(20))
% Compiled module: TEST__DEFINE.
IDL> obj_destroy, a
IDL> help, /heap  ; see Mom, no leaks!
Heap Variables:
   # Pointer: 0
   # Object : 0
```

The incorrect way:

```
IDL> a = obj_new('test', findgen(20))
IDL> ; Dang, did I write findgen(20) there?  I meant findgen(30) ...
IDL> a = obj_new('test', findgen(30))
IDL> obj_destroy, a
IDL> help, /heap  ; oops
Heap Variables:
   # Pointer: 1
   # Object : 1

<ObjHeapVar51>  STRUCT    = -> TEST Array[1]
<PtrHeapVar52>  FLOAT     = Array[20]
```

At this point, a variable named 'A' perpetually exists, and is immune to obj_destroy.

```
DL> obj_destroy, a
IDL> help
% At  $MAIN$
A               OBJREF    = <ObjHeapVar57>
Compiled Procedures:
   $MAIN$  TEST::CLEANUP          TEST__DEFINE

Compiled Functions:
          TEST::INIT
```

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.

Thanks in advance,
Randall

-- start: test__define.pro --

```
pro test::cleanup

  ptr_free, self.data

end

function test::init, data

  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: test__define.pro --