Subject: How to traverse/inquire a class object structure in IDL?
Posted by Paul van Delst on Wed, 13 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Hello,

Last night I entered into the world of IDL objects. I was amazed at how much
easier it is to keep control of a data object rather than using a regular
structure.

Anyway, since I have been programming IDL in an Object Oriented mode for about 8
hours, I have some questions that I hope someone out there can help me with. The
documentation (on-line and printed) was not useful.

I have a class structure definition in nasti__define.pro:

```
PRO nasti__define, nasti_structure

; -- Define the data structure
  nasti_structure = { NASTI, $
                    wavenumber     : PTR_NEW(), $
                    radiance       : PTR_NEW(), $
                    altitude       : PTR_NEW(), $
                    fov_angle      : PTR_NEW(), $
                    fov_index      : PTR_NEW(), $
                    latitude       : PTR_NEW(), $
                    longitude      : PTR_NEW(), $
                    aircraft_roll  : PTR_NEW(), $
                    aircraft_pitch : PTR_NEW(), $
                    scan_line_index : PTR_NEW(), $
                    date           : PTR_NEW(), $
                    time           : PTR_NEW(), $
                    decimal_time   : PTR_NEW() }

END
```

which will eventually contain an aircraft instrument data time series of unknown
length - hence the pointers.

I create the object:

```
IDL> n=OBJ_NEW( 'nasti' )
```

and read in some data from a netCDF file (only the first data structure field,
wavenumber, is filled for now):

```
IDL> print, n->read_nasti( ncdf_filename )
```

This all works fine. I have an simple inquire method:

```
PRO nasti::inquire_nasti

  PRINT, FORMAT = '( /5x, "Inquiring..." )'
  PRINT, PTR_VALID(), OBJ_VALID()

END
```

which when I run it, gives:

```
IDL> n->inquire_nasti
     Inquiring...
<PtrHeapVar2>
<ObjHeapVar1(NASTI)>
```

where the PtrHeapVar2 is the pointer to "self.wavenumber" and the object reference is for the object. Cool.

I assumed that if I destroyed the object, the pointer references would still be there, dangling away, which turns out to be true:

```
IDL> obj_destroy, n
IDL> print, ptr_valid(), obj_valid()
<PtrHeapVar2>
<NullObject>
```

Not good. As more objects are created and destroyed, the valid pointer list grows. I would like to do the following in a CLEANUP method:

```
  FOR i = 0, n_object_structure_elements - 1 DO $
    IF ( PTR_VALID( self.(i) ) ) THEN $
      PTR_FREE, self.(i)
```

that is, *explicitly* free up the pointers. This works great if I have a value for n_object_structure_elements.

QUESTIONS:

1) Is my technique valid? That is, I want to do the following:
- create a data object
- read some amount of data into that object
- do stuff with the data object
- delete the data object INCLUDING any pointers in the object.
I don't know how much data I have ahead of time so I used pointers. Can I create data objects on the fly, based on how much data is in a datafile or requested from a datafile?

2a) If my technique is o.k., how do I free up the pointers in my object before I
destroy it?

..OR..

2b) Is the above code stub a valid/smart way to free up the pointers in a data
object and, if so, how do I determine the value of n_object_structure_elements?
(You can't use N_TAGS() on an object but you can use the self.(i) type of
structure reference so I'm confused.)

If you know how to solve my problem, please let me know. And, given my neophyte
object programming status, be kind.

:o)

paulv

--
Paul van Delst
Space Science and Engineering Center | Ph/Fax: (608) 265-5357, 262-5974
University of Wisconsin-Madison | Email: paul.vandelst@ssec.wisc.edu
1225 W. Dayton St., Madison WI 53706 | Web: http://airs2.ssec.wisc.edu/~paulv