Subject: Re: How to traverse/inquire a class object structure in IDL?
Posted by J.D. Smith on Fri, 15 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Paul van Delst wrote:
>
> First off, thanks to David and Pavel for their insights. Before I could check the
> newsgroup for replies, one of our younger go-getter science types came and told me
> something about object oriented programming that made good sense:
>
> The data should be an attribute of the object, not the object itself.
>
> Hmm. Anyway, he and I sat down for about 15 minutes and came up with the following
> class structure definition and cleanup method:
>
> PRO nasti__define
>
> ; -- Define the NAMED data structure attribute
> data = { data, $
> 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() }
>
> ; -- Create object CLASS structure
> nasti = { nasti, $
> data : data }
>
> END
>
> I like this becuase now I can add additional attributes whenever I want, e.g.
> global attribute data read from the netCDF data file containing instrument
> calibration information and/or processing software CVS/RCS info etc.
>
> The cleanup method is now:
>
> PRO nasti::cleanup
>
> PRINT, FORMAT = '(/5x, "Clean up...")'

>
> ; -- Free up pointers
>   n_data_fields = N_TAGS(self.data)
>   FOR i = 0, n_data_fields - 1 DO $
>   IF (PTR_VALID(self.data.(i))) THEN $
>   PTR_FREE, self.data.(i)
>
> END
>
> I just couldn't bring myself to typing PTR_FREE, self.whatever a bunch of times
> because if I ever change the data structure definition, I would have to change the
> cleanup as well. I like changes in my code to have as small a footprint as
> possible, i.e. change is required in as few places as possible. Dunno if that's a
> great idea but for my simple little example but it's a start. Right?
>
> I wish I'd "discovered" objects earlier.....all that code I wrote that *needs* the
> data to be encapsulated. Crikey.

I hate to add yet another level of dereferencing, which can get pretty
ugly in code, but I have had some instances where an array of structure
"data records" (or "attribute records" just as well) can be employed for
just this purpose. If your attributes are really changing that much,
then they shouldn't be "hard-coded" into any structure itself,
class-defining or otherwise. An advantage of the method below is that if
the data pointer field is the same in all records, then cleanup is
trivial.

This is best illustrated with an example:

```
;; The class definition procedure...
pro MyClassDef__define
  struct={MyClassDef,
 Recs:ptr_new()} ; a pointer to a list of structs of type
MyClassData_Rec

  ; Define an auxilliary structure for new Data Records
  struct={MyClassData_Rec,
 Name:'',
 Data:ptr_new()}
end
```

Then for each new type of data record to include, simply use something
like:

```
pro MyClassDef::AddRec, name, data
   rec={MyClassData_Rec,Name:name,Data:ptr_new(data)}
   if ptr_valid(self.Recs) then $
      *self.Recs=[*self.Recs,rec] else self.Recs=ptr_new([rec])
```

end

You can easily also put in code to remove data records during run-time or do any other attribute manipulation, based on the Name field (or other relevant fields), etc. Obviously this can grow quite powerful, but be forewarned that such power is easily misused.

When it's time to cleanup, we have simply:

```
pro MyClassDef::Cleanup
  if ptr_valid(self.Recs) then ptr_free,(*self.Recs).Data, self.Recs
end
```

Note that ptr_free doesn't care if the pointer is a null pointer (in fact it's faster just to free it without testing for this), but dereferencing does -- hence the first ptr_valid() test is the only one necessary.

Good Luck,

JD

P.S. The proper way to reference things is then:

```
; A pointer to a single attribute's data
thedataptr=(*self.Recs)[element].Data

; all pointers to all attributes' data
alldataptrs=(*self.Recs).Data

; a single attribute's data vector (or array, or ....)
thedata=*(*self.Recs)[element].Data

;an vector of names of all the attributes
attrs=(*self.Recs).Name
```

etc.

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|