Subject: Recursive object destruction, Was: IDL Shapefile Object
Posted by Richard Younger on Fri, 30 Nov 2001 15:56:48 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> I've never used it. (Guess I should make a habit
> of reading the documentation that *is* there!)
> But I can imagine a case for it.
>
[...]
>
> In this case a HANG_ON_DONT_DO_IT keyword on the cleanup
> method might be appropriate.

Sort of on this topic, I'm looking for some advice on a routine that I wrote before IDL 5.5 came out to do this sort of thing. Maybe someone else will find it useful. It has has a few minor advantages and a couple of disadvantages over HEAP_FREE.

In particular, I have objects with pointers to structures with arrays of pointers to structures with pointers. In a pseudo-IDL syntax, that's Object->(*struct1[].(*struct2.(*data[]))). And I wanted to write a routine to delete these suckers generically. I call it Destroy_Thing.

First as an advantage, this routine doesn't access IDL's internal tables, so one of the previously mentioned "frightener" clauses on HEAP_FREE searching the entire heap table is circumvented. This may be nice for those of us who have verified that IDL can handle more than an int's worth of heap variables. But I don't know exactly how frightening that clause is, so I don't know how big an advantage this is.

In the 'mixed' category, the Obj_destroy call in has keyword inheritance. So keywords, like HANG_ON_DONT_DO_IT, in your cleanup routines are fine, but not parameters. I've never actually used this feature (my top level objects have no destructor keywords), but it was cheap to add, so I did.

On the down side, there's some issues with the recursive passing of structure members. Since structure members are always passed by value, I'm a little worried that using this destroy routine on, say, structures with large arrays in them would use up more memory (and time spent copying data to be destroyed anyway) than is reasonable. Does anybody have suggestions on this? Am I right to be worried?

So I don't know if some of the choices I made are the correct ones. Am I using _REF_EXTRA right? Is there any point to checking whether I'm dealing with a (structure member) copy or the real McCoy? What about undefining vs. setting the variable to zero and redundant destruction?

Are there flaws that I haven't seen?

Boy, I'm just posting up a storm this week.

Thanks,
Rich

--
Richard Younger

```
;***********************************
;   Program:   Destroy_Thing.pro
;   Purpose:   Take some variable of any type, turn it into a NULL
;              reference or undefine it, and free all of its
;              constituents.
;   Inputs:    thing = something of any type to be undefined.
;              keywords to be passed to any and all object cleanup
;              methods required
;
;   Author:    Richard D. Younger
;   Date:      August, 2001
;
;***********************************

PRO destroy_thing, thing, copy_flag=copy_flag, _REF_EXTRA=extr

  datatype = Size(thing, /TName)

  CASE datatype OF
    'STRUCT':  BEGIN
          FOR j=0, N_Elements(thing)-1 DO BEGIN
             N = N_Tags(thing[j])
             FOR i=0, N-1 DO BEGIN
                Destroy_Thing, thing[j].(i), /copy_flag, $
                    _REF_EXTRA=extr
             ENDFOR
          ENDFOR
      END

      'POINTER':   BEGIN
           pval = Ptr_Valid(thing)
           FOR j=0, N_Elements(pval)-1 DO BEGIN
              IF pval[j] THEN BEGIN
                 Destroy_Thing, *(thing[j]), copy_flag=copy_flag, $
                    _REF_EXTRA=extr
                 PTR_FREE, thing
          ENDIF
        ENDFOR
        thing = PTR_NEW()
    END

    'OBJREF':   OBJ_DESTROY, thing, _REF_EXTRA=extr

    'UNDEFINED': RETURN

    ELSE: ;do nothing: no recursion req'd
  ENDCASE

  IF NOT Keyword_Set(copy_flag) THEN $
    dummy = TEMPORARY(thing) ;thing = 0

END
```