Subject: Re: Ptr_Wrapper (Useful?)
Posted by JD Smith on Tue, 23 Nov 2004 16:43:12 GMT
View Forum Message <> Reply to Message

On Tue, 23 Nov 2004 08:01:35 -0800, Robert Gamble wrote:

> Greetings all. One of the first pieces of 'useful' code I wrote was
> an object that encapsulates pointer creation and deletion. It's a
> piece of simple code that utilizes David Fanning's "linkedlist".
> Everytime a pointer is created using the object, it's attached to a
> linked list. When the object is destroyed, the linked list is unwound
> and each pointer in the list is freed. There is also an explicit
> "free_ptrs" routine that allows destruction to be more controlled.
> The simplest way of using it is to create one ptr_wrapper object in
> each *.pro file and destroy it at the end. If you prefer to have
> different wrappers, each of which maintains its pointers for more
> specialized time frames for better efficiency, you can do that too.
> And finally, you can create and free pointers using the object
> methods just as you would normally in IDL.
>
> Quick example:
>
> PRO testptrs
>
> oPtr_Wrapper = Obj_New('ptr_wrapper')
> ptrA = oPtr_Wrapper->Ptr_New(/ALLOCATE_HEAP) ; Uses same keywords
> as Ptr_New
>
> .
> .
> .
>
> ptrB = oPtr_Wrapper->Ptr_New(/ALLOCATE_HEAP)
> .
> .
> .
> Obj_Destroy, oPtr_Wrapper
>
> END
>
>
> This bit of code is obviously simplistic and using the wrapper here
> would be wasteful. I've found it to be much more useful in objects
> where pointers are created in different routines. Each can be
> assigned to self.oPtr_Wrapper and then self.oPtr_Wrapper can be
> destroyed in the cleanup routine...
>
> If there's interest in the code, I'll be happy to e-mail the file or

> submit it to the RSI site.

Sounds very interesting. Just a note that RSI provides something of somewhat similar functionality in HEAP_FREE. I've taken to pointing HEAP_FREE at large opaque data structures in CLEANUP routines, and letting it do the job of hunting through for dynamically allocated resources to free for me. I've found that it's actually *faster* in many cases than the more pedantic method of testing and descending the data structure freeing known heap data as you go. It's also much easier to maintain: when I add more heap data to the class, I don't need to update the Cleanup code.

JD
