Subject: Re: Using NO_COPY with pointers
Posted by Liam E. Gumley on Mon, 14 Apr 2003 17:27:35 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.19048026a9265639989b4f@news.frii.com...
> Folks,
>
> This may be common knowledge, but I wasn't aware of it, and
> it is one of those things that makes you feel all warm and
> goose-pimply about IDL.
>
> I was adding a "user value" to all of my objects
> today, via a UVALUE field in the "atom" object that
> is inherited by all objects in my library. This field
> is, of course, a pointer.
>
> Naturally enough, I want to be able to get and set
> the "value" of this field sometimes without making
> a copy of the data. This is no problem when I am
> adding the information to the pointer, I simply use
> the NO_COPY keyword on PTR_NEW:
>
>   self.uvalue = Ptr_New(uvalue, /No_Copy)
>
> But it is a bit of a problem when I want to "get"
> the value back:
>
>   uvalue = *self.uvalue
>
> I was of the impression that pointer de-referencing
> *always* made a copy of the data. But on a whim, I
> tried this:
>
>   IF Keyword_Set(no_copy) THEN uvalue = Temporary(*self.uvalue)
>
> Lo and behold, this did *exactly* what I wanted it to do!!
>
>   Help, *self.uvalue
>   <PtrHeapVar2093>
>   UNDEFINED = <Undefined>
>
> Moreover,
>
>   Print, Ptr_Valid(self.uvalue)
>   1
>
> Perfect! A valid pointer that points to an undefined variable.

> Hooray! You gotta love IDL! :-)

Hi David,

I made use of the same behavior in the "get and set state" service procedures (kinda like methods) for the IMGUI sample program I wrote for my book. The complete code for IMGUI is part of the sample program archive available at

http://www.gumley.com/PIP/About_Book.html

Here's the relevant code:

```
PRO IMGUI_GET_STATE, EVENT, INFO, $
  NO_COPY=NO_COPY

;- Get pointer
widget_control, event.top, get_uvalue=infoptr
if (ptr_valid(infoptr) eq 0) then $
  message, 'State information pointer is invalid'

;- Get state information structure
if (n_elements(*infoptr) eq 0) then $
  message, 'State information structure is undefined'
if keyword_set(no_copy) then begin
  info = temporary(*infoptr)
endif else begin
  info = *infoptr
endelse

END

PRO IMGUI_SET_STATE, EVENT, INFO, $
  NO_COPY=NO_COPY

;- Get pointer
widget_control, event.top, get_uvalue=infoptr
if (ptr_valid(infoptr) eq 0) then $
  message, 'State information pointer is invalid'

;- Set state information structure
if (n_elements(info) eq 0) then $
  message, 'State information structure is undefined'
if keyword_set(no_copy) then begin
  *infoptr = temporary(info)
endif else begin
  *infoptr = info
endelse
```

END

Cheers,
Liam.
Practical IDL Programming
http://www.gumley.com/

Subject: Re: Using NO_COPY with pointers
Posted by JD Smith on Mon, 14 Apr 2003 22:12:02 GMT
View Forum Message <> Reply to Message

On Mon, 14 Apr 2003 08:22:40 -0700, David Fanning wrote:

> Folks,
>
> This may be common knowledge, but I wasn't aware of it, and it is one of
> those things that makes you feel all warm and goose-pimply about IDL.
>
> I was adding a "user value" to all of my objects today, via a UVALUE
> field in the "atom" object that is inherited by all objects in my
> library. This field is, of course, a pointer.
>
> Naturally enough, I want to be able to get and set the "value" of this
> field sometimes without making a copy of the data. This is no problem
> when I am adding the information to the pointer, I simply use the
> NO_COPY keyword on PTR_NEW:
>
>   self.uvalue = Ptr_New(uvalue, /No_Copy)
>
> But it is a bit of a problem when I want to "get" the value back:
>
>   uvalue = *self.uvalue
>
> I was of the impression that pointer de-referencing *always* made a copy
> of the data. But on a whim, I tried this:
>
>   IF Keyword_Set(no_copy) THEN uvalue = Temporary(*self.uvalue)
>
>
I think the assignment is what's making a copy there. Pointer
dereferencing by itself just hands you the relevant heap variable. In
other words, "*ptr_var", and "var" are interchangeable in terms of all
memory referencing issues. So if you had some direct use for uvalue, ala:

 print,total(4+(*self.uvalue))

then you needn't suffer the copy. Compare to:

```
uvalue=*self.uvalue
ptrint,total(4+uvalue)
```

where a copy *is* made, and you'll see the same issues hold for both
regular and heap variables.

JD
