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