
Subject: Re: how does /no_copy work???

Posted by [steinhh](#) on Thu, 03 Jun 1999 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

In article <MPG.11bfc3654e35a3b89897cd@news.frii.com>

davidf@dfanning.com (David Fanning) writes:

[...]

> Uh, well, that's because what you are passing into and out of
> procedures and functions by arguments and keywords *IS* a pointer.
> That is to say, a variable in IDL is, among other things, a
> C pointer. This is what is passed into a method like SetProperty:
>
> myobject->SetProperty, Data=thisData
>
> The variable thisData is, essentially, the pointer to the data.
> We say the data is "passed by reference", meaning that what the
> procedure received was the actual physical address of the data
> in memory (I.e. the pointer to the data). If the data is copied
> before it is passed into the procedure, we say it is "passed by
> value". For example, to pass this data by value we could create \an
> expression. Expressions are passed by value:
>
> myobject->SetProperty, Data=thisData * 1
>
> (An IDL variable is actually a structure that contains information
> about the size and type of data, etc. as well as the actual C
> pointer to the memory location of the data. So it is a little more
> complicated than saying a variable is a pointer.)

A few comments, though: When an operation like

```
myobject->setproperty,data=thisdata
```

is performed, the setproperty method will usually end up making a copy of the data set, even though it receives the data by reference. There will be one copy inside the object, and one copy in the calling routine, still available after the call.

The way to avoid this is by either using a /no_copy keyword, or (more generally available) to use the call

```
myobject->setproperty,data=temporary(thisdata)
```

After this call, "thisdata" will be undefined, and the data will not have been copied. Likewise, one could avoid a lot of copying in David's second example, by using

```
myobject->SetProperty, Data=temporary(thisData) * 1
```

..bearing in mind, though, that thisData will be undefined after the call.

> Where No_Copy is useful is when you are transferring some
> information from one memory location to another. For example,
> from a local variable in an event handler to the user value
> of the top-level base, or from a local variable to an IDL
> pointer (heap variable). These operations actually copy the
> data to another memory location, unless you tell IDL not
> to with the NO_COPY keyword. Then all IDL transfers is the
> C pointer to the data that already exists in memory.

Though it's simpler to use the temporary() function, as in:

```
out_data = temporary(internal_data)
```

```
local_data = temporary(*in_data) ;; Shorthand to avoid *'es
```

```
:
```

```
*in_data = temporary(local_data) ;; Put it back
```

or similar.

> I've never had occasion to need or use NO_COPY, except
> where they are already supplied by IDL. Pass variables, or
> pass pointers to variables, and you will be fine.

I guess all (?) instances of /no_copy could be replaced by using the temporary() function instead...?

Regards

Stein Vidar
