Subject: Re: Arrays of Structures
Posted by JD Smith on Thu, 08 Feb 2007 20:42:48 GMT
View Forum Message <> Reply to Message

On Thu, 08 Feb 2007 14:02:08 -0500, Paul van Delst wrote:

> Mick Brooks wrote:

> You know, I'm confused now too. Check out the precedence in the IDL help:

> So, you see that the the structure field dereference operator, ".", and the array
> subscript operator, "[]" , have the same precedence. Operators with equal precedence are
> evaluated from left to right.
>
> So,
>
>   structs.a[23]
>
> means FIRST dereference the structure field (structs.a), THEN index the array ([23]). The
> way I see it,
>
>   structs.a[23]
> and
>   (structs.a)[23]
>
> should be equivalent.

The problem is that structs.a is not an array until the ".a" part has been applied. What if structs had been;

structs={a:indgen(25)}

what should structs.a[23] do then, and how should IDL know the difference. Until IDL knows what shape "structs.a" will have, it cannot make any informed decision about how to index it. You might object, saying that IDL should just evaluate that to begin with always, but think how expensive that is. Creating "structs.a" would cause a large temporary array to be created, only to finally index a single element. Compare the memory usage of the following:

```
IDL> struct=replicate({a:lindgen(100,100,100)},100)
IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
      382.472MB
IDL> val=struct[10].a[4]
IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
      0.00000MB extra
IDL> val2=(struct.a)[10,4]
IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
```

    381.470MB extra

So the latter method first creates a temporary variable of size 100,100,100,100, and then pulls a single element out of it. Not exactly good form.

JD