Subject: Re: structure arguments sometimes behave like value types - why?
Posted by JD Smith on Mon, 23 Apr 2007 17:31:17 GMT
View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 13:26:42 -0700, justspam03 wrote:

>
> erg, addendum.
> Admittedly, in the second case the structure is not a function
> argument but a return value - I still assumed that it doesn't matter
> in the case (just as it doesn't for pointers or object references).
> Does it?
> Thanks again

In the second case, you are returning a copy of the structure from inside the object, and then modifying that copy. The copy actually occurred at the statement "self.val". Had you instead used a pointer to a structure, ala:

```
pro structtest__define
 obj = { STRUCTTEST , val: ptr_new({nullableString})   }
end

function structtest::getStruct
 return, self.val
end
```

you could then return that *pointer*, and then modify directly the object's internal copy. Note that you're still returning a copy of *something* with:

```
b=x->getStruct()
```

but that something in this case is a (lightweight) copy of the pointer to the internal structure, rather than a full copy of the structure. However, just as it's dangerous to hand out too many sets of house keys, it's often not a good idea to pass pointers to your important internal data out to whomever may happen by.

Note that objects are similar to pointers in that they are (always, unlike C++) lightweight references to variables on the global IDL memory heap (it may help if you call them "object pointers"). They are accessed differently, but otherwise serve a similar function: you can make many copies, all of which refer to the same object.

JD