Subject: Re: structure arguments sometimes behave like value types - why?
Posted by justspam03 on Fri, 20 Apr 2007 20:26:42 GMT
View Forum Message <> Reply to Message

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
 Oliver

Subject: Re: structure arguments sometimes behave like value types - why?
Posted by justspam03 on Fri, 20 Apr 2007 20:36:55 GMT
View Forum Message <> Reply to Message

David,

you answer so quickly - sometimes I wonder whether there's more than
one of you.
I get your answer, but I have a hard time getting used to the IDL way.
Maybe I
should quit my C# and C programming. It only blocks the path to
enlightment ;-)
Thanks!
 Oliver

Subject: Re: structure arguments sometimes behave like value types - why?
Posted by David Fanning on Fri, 20 Apr 2007 21:26:17 GMT
View Forum Message <> Reply to Message

justspam03@yahoo.de writes:

> according to the IDL reference, structures (as a whole) are treated
> like reference types when supplied as an argument. A small test
> program of the kind
> confirms this - after the call to changeval, a.value has value '4'.
> However, when the structure is an object variable as in the example
> appended below, it seems that only a copy of structtest.val is
> exchanged, not a reference to it. The final call to printValue in
> 'main' prints a '1'.
> Could someone please explain why?

In one case you are passing an IDL variable, which is passed
by reference. In the other you are passing a structure reference,

which is NOT an IDL variable, and like everything else that is NOT
an IDL variable, is passed by value:

   http://www.dfanning.com/tips/read_subscripted_array.html

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

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