Subject: Re: IDL interpreter questions - can someone (D.Fanning) explain - TIA
Posted by <mankoff[1] on Sat, 19 May 2001 00:17:33 GMT
View Forum Message <> Reply to Message

On 18 May 2001, Craig Markwardt wrote:
> <mankoff@I.HATE.SPAM.lasp.colorado.edu> writes:
>> On Fri, 18 May 2001, JD Smith wrote:
>>> dadada wrote:
>>>> How are variables referenced by default?
>>>
>>> I'm not sure what you mean here. Pointer references? They are explicit
>>> only... i.e. you can't create a reference of an existing variable.
>>
>> Not sure either, but here is my interpretation of the question/answer:
>> In functions, variables are *always* 'by value'
>> In procedures, they are 'by value' unless you put a "return" statement
>> anywhere in the procedure. If this exists, then they are passed 'by
>> reference'
>
> Sorry Ken I'm going to have to take you to task for a few things.
> First of all, pass by value vs. pass by reference:
>
> * all variables are passed by reference, *except*
> * subscripted arrays, structure tags, and (I believe) system variables,
> which are passed by value
>
> It doesn't make a difference whether you have a return statement or
> not.

Hi JD,

No need to apologize for a correction. But i have some questions about this that maybe you can answer.

I thought 'by value' meant that the called routine gets a copy of the variable, and cannot modify the contents of the variable in the calling routine.

And that 'by reference' means that the called routine gets a pointer to the variable from the calling routine. Any changes that the called makes, appear in the caller.

Now its true that I don't know anything about the actual IDL implementation (though I have written RPC code for IDL). I actually answered based upon the behavior of IDL, not the implementation. That is, functions won't modify the callers variables, and neither will procedures, unless you add the 'return'.

> As for continuations, closures, etc., these are computer science
> jargon for specific language behaviors. IDL has none of them. I
> understand continuations to be a way for execution contexts to be
> suspended, saved, and later restored. Perhaps the CATCH error
> handling technique is a nascent continuation. Alas, this has nothing
> to do with the CONTINUE reserved word recently added to FOR and WHILE
> loops.

ok, i get it.

But if you want to "suspend, save, restore" the execution state, wouldn't
"save, /vars" and "save, /all" simulate this to some extent?

thanks for any clarifications,

-k.

--
Ken Mankoff
LASP://303.492.3264
http://lasp.colorado.edu/~mankoff/
