Subject: Re: Does this make sense? (scalar objects)
Posted by JD Smith on Tue, 09 Dec 2003 19:16:37 GMT
View Forum Message <> Reply to Message

On Mon, 08 Dec 2003 08:10:07 -0700, Marc Schellens wrote:

> JD Smith wrote:
>> On Sat, 06 Dec 2003 01:33:07 -0700, Marc Schellens wrote:
>>
>>
>>> JD Smith wrote:
>>>
>>>> For objects, it's quite clear why you can't apply methods across a
>>>> vector of object variables:
>>>
>>>
>>>
>>>> IDL> objs=[obj_new('IDL_Container'), obj_new('MyFooObj')] IDL>
>>>> objs->DoSomeMethod ; WRONG
>>>>
>>>> Since objects are generic pointers, and a vectors of objects can
>>>> contain any combination of object classes, it's clear why you can't
>>>> use this notation. The same is true of pointer arrays, for nearly the
>>>> same reasons:
>>>>
>>>> IDL> ptrs=[ptr_new('string'),ptr_new(indgen(5))] IDL> print,*ptrs+5
>>>> ;WRONG
>>>
>>> With the pointers it would be messy indeed (if your data is that
>>> uniform that such an expression would really make sense, use an array).
>>> Another thing is of course that there is no reason to not allow your
>>> example for a single element pointer array.
>>>
>>>
>>> With the objects though there would be no problem: Just let IDL call
>>> the appropriate method for each individual object. I even would think
>>> that this is more along the IDL array oriented way.
>>>
>>>
>>>
>> And what if all of the objects in the array do not implement the same
>> method, and what if the values they return cannot be concatenated into
>> an array (e.g. one returns a string, another a floating vector). I
>> originally was of your opinion, but have come to see how painful things
>> could get.
>
> Very simple: An error message will be issued. Even now you can
> concatenate strings and numbers. And if you try to cancatenate arrays of

> non-matching dimensions you get an error message also. And that
> different objects have different method functions is in the sense of
> object orientation.

I think if you allowed this you'd be forced to allow the equivalent
pointer operations, since in both cases you're leaving it up to the
user to ensure the methods and returned types are compatible with
array access/storage. And what about output arguments or keyword
variables:

```
IDL> myobjarr=[obj_new('type1'), obj_new('type2')]
IDL> myobjarr->GetProperty,TYPE=t
```

does "t" get vectorized in the same way

```
IDL> t=myobjarr->ReturnType()
```

would? What if some objects implemented a keyword as input and others
as output? I think you'll see if you follow it all the way through to
the conclusions, you'll be causing yourself more trouble than it's
worth just to save the occassional:

```
IDL> for i=0,n_elements(myobjarr)-1 do myobjarr[i]->Print
```

>>> As I said, I agree that the cannot be abolished, but if from now on
>>> scalars could be transposed, rebined, etc. (and reffering to my OP:
>>> method called on single object arrays), This would not break any
>>> existing code, would it?
>>
>>
>> It's tough to say... probably none of my code, but I'm sure there are
>> examples where the very inability to treat a scalar like a vector is
>> capatalized upon.
>>
>>
> I am (almost) sure there is no example. Challenge: Can anybody reading
> this post one?

Well, to be pedantic:

```
pro test_scalar
  a=0
  b=[0]
  catch,err
  if err ne 0 then begin
     print,'This is never printed unless scalars cannot be treated as vectors'
     return
  endif
```

```
  b=rebin(b,5)
  a=rebin(a,5)
end
```

Yes, it's contrived, but backward compatibility isn't about ensuring only "reasonable usage" is kept compatible: witness the perverse applications of the _EXTRA structure which were still supported after _REF_EXTRA appeared. That's the curse of committing yourself to complete compatibility: all the ridiculous misuses of old misfeatures must always remain supported.

JD
