Subject: Re: Does this make sense? (scalar objects)
Posted by marc schellens[1] on Tue, 09 Dec 2003 15:59:54 GMT
View Forum Message <> Reply to Message

>>> 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

Better not. Consider:

objectArr=objarr(3,4)

tArr=indgen(5,3,4) ;; each object gets a 5 element vector

;; fill with objects

w=where(object_in)

res = objectArr[w]->DoSomething(T=tArr)

You would need a reform to index tArr appropriately here. Messy.

> 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

For my taste

myobjarr->Print

looks nicer nevertheless.
These array memeber function calls would be there
to make some expressions more elegant. In the other cases you would be
still able to use a loop.
Of course the behaviour must be defined.
And the user must of course know what he is doing.
My point is that I don't see any disadvantage if it would be possible.
The main aim would be to apply it to arrays of same object type anyway.

>>>> 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?
>
>
> 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.

I see your point, but try this with <6.0 and 6.0:

```
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
   if a eq 0 then print,'a eq 0 (and scalar)'
   if b eq 0 then print,'b eq 0 (and scalar if <6.0)'
end
```

So this kind of backward compatibility is already broken (fortuantely in this example as I think). I don't know how often if at all such this kind of code was used. Maybe its even ok to continue with this scalar/one element array distinction to prevent potential errors.

marc
