Subject: Does this make sense? (scalar objects)
Posted by marc schellens[1] on Wed, 03 Dec 2003 14:33:20 GMT
View Forum Message <> Reply to Message

check this out:

```
file tt.pro:
pro o::test

help,self[[0]]
help,(self[[0]])

print,self[[0]].a
print,(self[[0]]).a ;; ???
end

pro tt

s={o,a:0}

print,s[[0]].a
print,(s[[0]]).a

obj=obj_new('o')

obj->test
end


IDL> tt
% Compiled module: TT.
       0
       0
<Expression>    OBJREF    = Array[1]
<Expression>    OBJREF    = Array[1]
       0
% Object reference must be scalar in this context: <OBJREF    Array[1]>
% Execution halted at: O::TEST            7 /home/marc/idl/tt.pro
%                      TT                19 /home/marc/idl/tt.pro
%                      $MAIN$
```

Doesn't make sense, does it?

cheers,
marc

Subject: Re: Does this make sense? (scalar objects)
Posted by JD Smith on Wed, 03 Dec 2003 22:25:26 GMT
View Forum Message <> Reply to Message

On Wed, 03 Dec 2003 07:33:20 -0700, Marc Schellens wrote:

```
> check this out:
>
> file tt.pro:
> pro o::test
>
> help,self[[0]]
> help,(self[[0]])
>
> print,self[[0]].a
> print,(self[[0]]).a ;; ???
> end
>
> pro tt
>
> s={o,a:0}
>
> print,s[[0]].a
> print,(s[[0]]).a
>
> obj=obj_new('o')
>
> obj->test
> end
>
>
> IDL> tt
> % Compiled module: TT.
>        0
>        0
> <Expression>    OBJREF    = Array[1]
> <Expression>    OBJREF    = Array[1]
>        0
> % Object reference must be scalar in this context: <OBJREF    Array[1]>
> % Execution halted at: O::TEST            7 /home/marc/idl/tt.pro %
>                 TT                 19 /home/marc/idl/tt.pro %
>           $MAIN$
>
>
> Doesn't make sense, does it?
```

Well, given that self is always a scalar, your attempts to index it are confusing. In any case, the notation a[[b]] creates a single element vector:

```
IDL> a=1
IDL> print,size(a[[0]],/DIMENSIONS)
           1
```

You cannot do anything to more than one object at a time (e.g. no objarr method calls or instance variable dereference). Hence the error. The reason why self[[0]].a works, is that there is probably special code to handle instance variable derefence for a single element vector, which does not or cannot operate with (self[[0]]).a. Method calls don't like a vector no matter what: try

```
obj[[0]]->test
```

Confusing issues like this have lead at least one RSI programmer to long for the abolishment of the scalar as a separate type from a single element vector. Sadly, the chance to do this without breaking lots of code has long passed.

JD

---

## Subject: Re: Does this make sense? (scalar objects)
## Posted by marc schellens[1] on Fri, 05 Dec 2003 09:52:10 GMT

View Forum Message <> Reply to Message

```
JD Smith wrote:
>  On Wed, 03 Dec 2003 07:33:20 -0700, Marc Schellens wrote:
>
>
>> check this out:
>>
>> file tt.pro:
>> pro o::test
>>
>> help,self[[0]]
>> help,(self[[0]])
>>
>> print,self[[0]].a
>> print,(self[[0]]).a ;; ???
>> end
>>
>> pro tt
>>
>> s={o,a:0}
>>
>> print,s[[0]].a
>> print,(s[[0]]).a
```

```
>>
>> obj=obj_new('o')
>>
>> obj->test
>> end
>>
>>
>> IDL> tt
>> % Compiled module: TT.
>>        0
>>        0
>> <Expression>    OBJREF    = Array[1]
>> <Expression>    OBJREF    = Array[1]
>>        0
>> % Object reference must be scalar in this context: <OBJREF    Array[1]>
>> % Execution halted at: O::TEST            7 /home/marc/idl/tt.pro %
>>                 TT                19 /home/marc/idl/tt.pro %
>>           $MAIN$
>>
>>
>> Doesn't make sense, does it?
>
>
>  Well, given that self is always a scalar, your attempts to index it are
>  confusing.  In any case, the notation a[[b]] creates a single element
>  vector:
>
>  IDL> a=1
>  IDL> print,size(a[[0]],/DIMENSIONS)
>            1
>
>  You cannot do anything to more than one object at a time (e.g. no objarr
>  method calls or instance variable dereference).  Hence the error.  The
>  reason why self[[0]].a works, is that there is probably special code to
>  handle instance variable derefence for a single element vector, which
>  does not or cannot operate with (self[[0]]).a.  Method calls don't like
>  a vector no matter what: try
>
>  obj[[0]]->test
>
>  Confusing issues like this have lead at least one RSI programmer to long
>  for the abolishment of the scalar as a separate type from a single
>  element vector.  Sadly, the chance to do this without breaking lots of
>  code has long passed.
```

I cannot guess any example about which (IDL) code would be broken,
if single element vectors and scalars would be treated the same.

Do you have an example?
Or did you mean binary code linked to IDL?

marc

---

## Subject: Re: Does this make sense? (scalar objects)
Posted by marc schellens[1] on Fri, 05 Dec 2003 09:58:05 GMT
View Forum Message <> Reply to Message

> I cannot guess any example about which (IDL) code would be broken,
> if single element vectors and scalars would be treated the same.
> Do you have an example?
> Or did you mean binary code linked to IDL?

Sorry, please forget. I read your reply not careful enough.
As you were talkin gabout the abolishment of scalar type,
of course you are right.
Nevertheless, apart from indexing there should not be any
difference in behaviour.

---

## Subject: Re: Does this make sense? (scalar objects)
Posted by JD Smith on Fri, 05 Dec 2003 17:55:52 GMT
View Forum Message <> Reply to Message

On Fri, 05 Dec 2003 02:58:05 -0700, Marc Schellens wrote:

>> I cannot guess any example about which (IDL) code would be broken, if
>> single element vectors and scalars would be treated the same. Do you
>> have an example?
>> Or did you mean binary code linked to IDL?
>
> Sorry, please forget. I read your reply not careful enough. As you were
> talkin gabout the abolishment of scalar type, of course you are right.
> Nevertheless, apart from indexing there should not be any difference in
> behaviour.

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
```

Single element vectors are different than scalars in several ways: they can be transposed, reformed, and rebinned, whereas scalars cannot, and they can have matrix multiplications applied to them, etc. A better way of asking the question is "What can't you do with scalars that you can do with vectors?". The answer to this consists of the long list of IDL vector operations discussed here daily. There may not be any *useful* distinctions between scalars and single-element vectors, but there are certainly plenty of programmatic distinctions, which would break backward compatibility if ignored --- hence, we are stuck with both.

JD

---

Subject: Re: Does this make sense? (scalar objects)
Posted by marc schellens[1] on Sat, 06 Dec 2003 08:33:07 GMT
View Forum Message <> Reply to Message

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.

> Single element vectors are different than scalars in several ways:
> they can be transposed, reformed, and rebinned, whereas scalars
> cannot, and they can have matrix multiplications applied to them, etc.
> A better way of asking the question is "What can't you do with scalars
> that you can do with vectors?". The answer to this consists of the
> long list of IDL vector operations discussed here daily. There may not
> be any *useful* distinctions between scalars and single-element vectors,
> but there are certainly plenty of programmatic distinctions, which would
> break backward compatibility if ignored --- hence, we are stuck with
> both.

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?

marc

---

Subject: Re: Does this make sense? (scalar objects)
Posted by marc schellens[1] on Mon, 08 Dec 2003 15:10:07 GMT
View Forum Message <> Reply to Message

---

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.

>>> Single element vectors are different than scalars in several ways: they
>>> can be transposed, reformed, and rebinned, whereas scalars cannot, and
>>> they can have matrix multiplications applied to them, etc. A better way
>>> of asking the question is "What can't you do with scalars that you can
>>> do with vectors?". The answer to this consists of the long list of IDL
>>> vector operations discussed here daily. There may not be any *useful*
>>> distinctions between scalars and single-element vectors, but there are
>>> certainly plenty of programmatic distinctions, which would break
>>> backward compatibility if ignored --- hence, we are stuck with both.
>>
>> 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?

marc

---

Subject: Re: Does this make sense? (scalar objects)
Posted by JD Smith on Mon, 08 Dec 2003 18:34:13 GMT
View Forum Message <> Reply to Message

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.

>> Single element vectors are different than scalars in several ways: they
>> can be transposed, reformed, and rebinned, whereas scalars cannot, and
>> they can have matrix multiplications applied to them, etc. A better way
>> of asking the question is "What can't you do with scalars that you can
>> do with vectors?".  The answer to this consists of the long list of IDL

>>  vector operations discussed here daily.  There may not be any *useful*
>>  distinctions between scalars and single-element vectors, but there are
>>  certainly plenty of programmatic distinctions, which would break
>>  backward compatibility if ignored --- hence, we are stuck with both.
>
>  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.

JD

---

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

---

## 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