Subject: Re: Initializing object array
Posted by Dick Jackson on Tue, 09 Mar 2004 05:45:12 GMT
View Forum Message <> Reply to Message

Hi David,

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.1ab6e521618e56ee9896f1@news.frii.com...
>
> I have an object. One of the fields of this objects
> is a object array. The field is called "contours":
>
> PRO myclass__define
> class = { MYCLASS, contours:Obj_New()}
> END
>
> Now, when I create the object, I want to pass an object
> array of ROI objects that I created somewhere else.
>
> FUNCTION myclass::INIT, Contours=contours
> self.contours = contours
> RETURN, 1
> END
>
> This doesn't work. Says contours must be a scalar
> in this context. Well!!! Shucks.
>
> Surely I have done this before. But I can't for the
> life of me remember how. How do I initialize a field
> as an object array?

  class = { MYCLASS, contours:ObjArr(nElements)}

would do it, but it will be a fixed number of elements, and the passed
'contours' would have to match that. If that's what you need, then fine,
but I bet you need flexibility. All I can see for a solution right now
is using a pointer:

  class = { MYCLASS, contours:Ptr_New(/Allocate_Heap)}

then, to assign it:
  *self.contours = contours

and to refer to one contour:
  (*self.contours)[i]

Sorry if I'm stating the obvious... or am *I* missing something?

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / +1-403-242-7398 / Fax: 241-7392

Subject: Re: Initializing object array
Posted by David Fanning on Tue, 09 Mar 2004 06:31:22 GMT
View Forum Message <> Reply to Message

Dick Jackson writes:

> class = { MYCLASS, contours:ObjArr(nElements)}
>
> would do it, but it will be a fixed number of elements, and the passed
> 'contours' would have to match that. If that's what you need, then fine,
> but I bet you need flexibility. All I can see for a solution right now
> is using a pointer:
>
> class = { MYCLASS, contours:Ptr_New(/Allocate_Heap)}
>
> then, to assign it:
> *self.contours = contours
>
> and to refer to one contour:
> (*self.contours)[i]
>
> Sorry if I'm stating the obvious... or am *I* missing something?

Well, after taking a nap I can see that it is going to
have to be a pointer, but I still can't see why. :-)

I often use object containers to store objects, but I guess
this might have been the first time (at least in a while)
that I tried to store an object array. Oddly, an object
array is an object reference:

```
IDL> a = ObjArr(5)
IDL> Help, a
     A      OBJREF    = Array[5]
```

So you might think that if b was initialized as an object reference,
you could store an object array in it. It should fit, it's just a
long integer.

```
IDL> struct = {b:Obj_New()}
IDL> struct.b = a
 % Expression must be a scalar in this context: A.
```

Of course, with a structure I can do this:

```
IDL> struct = {c:ObjArr(5)}
IDL> struct.c = a
```

But I can't see a way to initialize an *object* like that. For example,
this doesn't work:

```
FUNCTION MyProg::INIT, a
  self.c = ObjArr(5)
  self.c = a
  RETURN, 1
END

PRO MyProg__Define
  class = {MYPROG, c:Obj_New()}
END
```

When I run it, I get this:

```
IDL> d = Obj_New('myprog', a)
  % Expression must be a scalar in this context: <OBJREF Array[5]>.
```

Isn't that strange!?

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Initializing object array
Posted by marc schellens[1] on Tue, 09 Mar 2004 09:53:42 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Dick Jackson writes:
>
>

>>   class = { MYCLASS, contours:ObjArr(nElements)}
>>
>> would do it, but it will be a fixed number of elements, and the passed
>> 'contours' would have to match that. If that's what you need, then fine,
>> but I bet you need flexibility. All I can see for a solution right now
>> is using a pointer:
>>
>>   class = { MYCLASS, contours:Ptr_New(/Allocate_Heap)}
>>
>> then, to assign it:
>>   *self.contours = contours
>>
>> and to refer to one contour:
>>   (*self.contours)[i]
>>
>> Sorry if I'm stating the obvious... or am *I* missing something?
>
>
> Well, after taking a nap I can see that it is going to
> have to be a pointer, but I still can't see why. :-)
>
> I often use object containers to store objects, but I guess
> this might have been the first time (at least in a while)
> that I tried to store an object array. Oddly, an object
> array is an object reference:
>
>   IDL> a = ObjArr(5)
>   IDL> Help, a
>   A   OBJREF   = Array[5]

IDL> a=indgen(5)
IDL> help,a
A   INT   = Array[5]

Its an *array* of object references, as 'a' is an *array* of INT.

> So you might think that if b was initialized as an object reference,
> you could store an object array in it. It should fit, it's just a
> long integer.
>
>   IDL> struct = {b:Obj_New()}
>   IDL> struct.b = a
>   % Expression must be a scalar in this context: A.
>
> Of course, with a structure I can do this:
>
>   IDL> struct = {c:ObjArr(5)}

> IDL> struct.c = a
>
> But I can't see a way to initialize an *object* like that. For example,
> this doesn't work:
>
> FUNCTION MyProg::INIT, a
> self.c = ObjArr(5)
> self.c = a
> RETURN, 1
> END
>
> PRO MyProg__Define
> class = {MYPROG, c:Obj_New()}
> END
>
> When I run it, I get this:
>
> IDL> d = Obj_New('myprog', a)
> % Expression must be a scalar in this context: <OBJREF Array[5]>.
>
> Isn't that strange!?

Its all perfectly fine.
Maybe you used before a container?

Cheers,
marc

Subject: Re: Initializing object array
Posted by btt on Tue, 09 Mar 2004 16:49:02 GMT

View Forum Message <> Reply to Message

Marc Schellens wrote:

> David Fanning wrote:
>
>> Dick Jackson writes:
>>
>>
>>> class = { MYCLASS, contours:ObjArr(nElements)}
>>>
>>> would do it, but it will be a fixed number of elements, and the passed
>>> 'contours' would have to match that. If that's what you need, then fine,
>>> but I bet you need flexibility. All I can see for a solution right now
>>> is using a pointer:
>>>
>>> class = { MYCLASS, contours:Ptr_New(/Allocate_Heap)}

>>>
>>> then, to assign it:
>>>   *self.contours = contours
>>>
>>> and to refer to one contour:
>>>   (*self.contours)[i]
>>>
>>> Sorry if I'm stating the obvious... or am *I* missing something?
>>
>>
>>
>> Well, after taking a nap I can see that it is going to
>> have to be a pointer, but I still can't see why. :-)
>>
>> I often use object containers to store objects, but I guess
>> this might have been the first time (at least in a while)
>> that I tried to store an object array. Oddly, an object
>> array is an object reference:
>>
>>   IDL> a = ObjArr(5)
>>   IDL> Help, a
>>   A   OBJREF   = Array[5]
>
>
> IDL> a=indgen(5)
> IDL> help,a
> A   INT   = Array[5]
>
> Its an *array* of object references, as 'a' is an *array* of INT.
>
>
>> So you might think that if b was initialized as an object reference,
>> you could store an object array in it. It should fit, it's just a
>> long integer.
>>
>>   IDL> struct = {b:Obj_New()}
>>   IDL> struct.b = a
>>   % Expression must be a scalar in this context: A.
>>
>> Of course, with a structure I can do this:
>>
>>   IDL> struct = {c:ObjArr(5)}
>>   IDL> struct.c = a
>>
>> But I can't see a way to initialize an *object* like that. For example,
>> this doesn't work:
>>
>>   FUNCTION MyProg::INIT, a

```
>>       self.c = ObjArr(5)
>>       self.c = a
>>       RETURN, 1
>>     END
>>
>>     PRO MyProg__Define
>>       class = {MYPROG, c:Obj_New()}
>>     END
>>
>>  When I run it, I get this:
>>
>>     IDL> d = Obj_New('myprog', a)
>>       % Expression must be a scalar in this context: <OBJREF Array[5]>.
>>
>>  Isn't that strange!?
>
>
>  Its all perfectly fine.
>  Maybe you used before a container?
>
```

Hello,

I agree with Marc. The behaviour you are seeing is consistent with how other variables behave. That's good, isn't it? It's written down that it's good in one of these Coyote books sprawled on my desk.

I have a hunch you are going somewhere else with this, but would this do it?

```
   FUNCTION MyProg::INIT, a
      ok = self->IDL_CONTAINER()
      If ok Then $
 if n_elements(a) NE 0 then $
 For i = 0, n_elements(a)-1 Do self->Add, a[i]
      RETURN, ok
   END

   PRO MyProg__Define
      class = {MYPROG, INHERITS IDL_CONTAINER}
   END
```

Ben

Subject: Re: Initializing object array
Posted by David Fanning on Tue, 09 Mar 2004 17:38:37 GMT
View Forum Message <> Reply to Message

Ben Tupper writes:

> I agree with Marc. The behaviour you are seeing is consistent with how
> other variables behave. That's good, isn't it? It's written down that
> it's good in one of these Coyote books sprawled on my desk.

Yes, thank you everyone. It is clear in the light of day that
what I wanted was a container not an object array. But you know
how it is, you are frantic to finish, you have been programming
for hours and hours, and somehow you just get an idea stuck in
your mind that for some reason you *don't* want a container
here. I've got to get my mind off of how much money I'm
not making. :-(

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/