Subject: IDL objects and names
Posted by s[1] on Wed, 12 Feb 2003 11:32:12 GMT
View Forum Message <> Reply to Message

Hi all,

when I create an IDL object, I can give it a name, like
oImg = OBJ_NEW('IDLgrImage',Name='myImage').

I thought that this might be nice for debugging, but neither help nor
heap_gc,/VERBOSE display the names of the objects.

Is there a way to make heap_gc and help display the names?
Are there any other techniques to get more info about objects that are
removed by heap_gc?
I mean, with an heap_gc,/VERBOSE output like
 <ObjHeapVar37> STRUCT   = -> IDLGRPALETTE Array[1]
I can't do much when I have a lot of IDLgrPalettes around.

Thanks a lot,

Sebastian

Subject: Re: IDL objects and names
Posted by David Fanning on Wed, 12 Feb 2003 13:33:01 GMT
View Forum Message <> Reply to Message

Sebastian (s@visita2.die.upm.es) writes:

> when I create an IDL object, I can give it a name, like
> oImg = OBJ_NEW('IDLgrImage',Name='myImage').
>
> I thought that this might be nice for debugging, but neither help nor
> heap_gc,/VERBOSE display the names of the objects.
>
> Is there a way to make heap_gc and help display the names?
> Are there any other techniques to get more info about objects that are
> removed by heap_gc?
> I mean, with an heap_gc,/VERBOSE output like
> <ObjHeapVar37> STRUCT   = -> IDLGRPALETTE Array[1]
> I can't do much when I have a lot of IDLgrPalettes around.

Yes, it can be discouraging after you write your first couple
of object graphics programs to get that e-mail from an important
colleague suggesting that your programs might be leaking a bit
of memory. Leaking!? More like a memory sieve! :-(

What to do with that huge long list of objects left on the heap? First, of course, is to start paying more attention to memory management as you write your programs. You will do this automatically in a few more weeks. Human beings can only take so much pain. :-)

The second step is to try to figure out where these leaks are coming from. If you have had the foresight to name your objects, you are in pretty good shape. First, you will have to recover the objects. Here is an example using two palette objects:

```
pal1 = Obj_New('IDLgrPalette', Name='GrayScale')
pal2 = Obj_New('IDLgrPalette', Name='Std Gamma')
pal2 -> LoadCT, 5
pal1 = 0
pal2 = 0
```

Ok, now my palettes are lost on the heap:

```
IDL> Help, /Heap
   Heap Variables:
      # Pointer: 2
      # Object : 2

   <ObjHeapVar2>   STRUCT    = -> IDLGRPALETTE Array[1]
   <ObjHeapVar3>   STRUCT    = -> IDLGRPALETTE Array[1]
   <PtrHeapVar4>   BYTE      = Array[768]
   <PtrHeapVar5>   BYTE      = Array[768]
```

Notice there are pointers, too, probably associated with the palettes. (At least I *hope* they are!)

The first step is to recover the palette objects. We can do this with OBJ_VALID and no arguments:

```
IDL> lostObjects = Obj_Valid()
IDL> Help, lostObjects
   LOSTOBJECTS     OBJREF    = Array[2]
```

Now we can just get and print the names of the objects:

```
IDL> FOR j=0,N_Elements(lostObjects)-1 DO BEGIN $
       lostObjects[j] -> GetProperty, Name=theName &$
       Print, theName & ENDFOR

     GrayScale
```

```
      Std Gamma
```

Or, because that darn FOR loop is so hard to write, I just
might add a PrintName method to the IDLgrPalette object.

```
 PRO IDLgrPalette::PrintName
   Print, self.name
 END
```

Save this as idlgrpalette__printname.pro. Then, just type
this:

```
  IDL> FOR j=0,N_Elements(lostObjects)-1 DO lostObjects[j] -> PrintName
     GrayScale
     Std Gamma
```

Now, maybe, you have some clue about where these darn things come
from. :-)

Cheers,

David

P.S. Of course, you must also destroy these objects.

```
  IDL> Obj_Destroy, lostObjects
  IDL> Help, /Heap
    Heap Variables:
       # Pointer: 0
       # Object : 0
```

Hooray, we got those pesky pointers, too! :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objects and names
Posted by s[1] on Wed, 12 Feb 2003 14:37:25 GMT

View Forum Message <> Reply to Message

Hi,

that lostObject thing is cool, made me re-read the OBJ_VALID docs....

Now I am of course a little bit lazy and don't want to write a PrintName
procedure for each object type I am using.
Is there something like an IDLClass that all IDLgr* classes inherit from?
I know that the reference manual says for most of the IDLgr* classes "This
class has no superclasses", but is it true? I mean, what's the point of
having "objects" and making no use of their (theoretical) capabilities?

Regards,

Sebastian

Subject: Re: IDL objects and names
Posted by David Fanning on Wed, 12 Feb 2003 15:09:33 GMT
View Forum Message <> Reply to Message

Sebastian (s@visita2.die.upm.es) writes:

> that lostObject thing is cool, made me re-read the OBJ_VALID docs....
> Now I am of course a little bit lazy and don't want to write a PrintName
> procedure for each object type I am using.
> Is there something like an IDLClass that all IDLgr* classes inherit from?
> I know that the reference manual says for most of the IDLgr* classes "This
> class has no superclasses", but is it true? I mean, what's the point of
> having "objects" and making no use of their (theoretical) capabilities?

No, I'm afraid it is true. One of the huge disadvantages of
IDL objects is that they are based on named structures, which
means the field names in the structure can't be duplicated.
In practice, this makes it hard to have deep inheritance
hierarchies. But what is even worse, is it makes it extremely
difficult to have multiple inheritance hierarchies.

So, for example, if IDLgrSurface and IDLgrAxis both inherited,
say, IDLgrBasicThing, then I could not make an IDLgrPlot object
that inherited both IDLgrSurface and IDLgrAxis because the
IDLgrBasicThing structure would be added to the definition
twice. It is a very difficult limitation to get around, especially
with large object programs. I think this is one of the reasons
we haven't seen RSI offer much in the way of higher level
object graphics functionality yet.

Cheers,

David

--
David W. Fanning, Ph.D.

Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objects and names
Posted by David Fanning on Wed, 12 Feb 2003 15:16:35 GMT
View Forum Message <> Reply to Message

Sebastian (s@visita2.die.upm.es) writes:

> Now I am of course a little bit lazy and don't want to write a PrintName
> procedure for each object type I am using.

Oh, well then, Sebastian, here is a generic PrintName
procedure that works with *all* IDL-supplied objects
that can have a name:

```
PRO PrintName, object
  object -> GetProperty, Name=theName
  Print, theName
END
```

I'll leave it as an exercise for the reader to figure out
what to do if the object *doesn't* have a name field. :-)

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objects and names
Posted by Randall Skelton on Wed, 12 Feb 2003 16:42:13 GMT
View Forum Message <> Reply to Message

> No, I'm afraid it is true. One of the huge disadvantages of
> IDL objects is that they are based on named structures, which
> means the field names in the structure can't be duplicated.
> In practice, this makes it hard to have deep inheritance

> hierarchies. But what is even worse, is it makes it extremely
> difficult to have multiple inheritance hierarchies.

IMHO, this isn't simply a huge disadvantage, but a colossal problem. If you look at some of tricks employed in Martin Shultz's scientific objects or the HESSI objects, you begin to realize much of the power of objects is squandered by RSI's limited implementation. This combined with the lack of public methods and operator overloading seriously reduces the usefulness of OOP in IDL for actual computing. I agree with David that you can write relatively simple objects with strict naming for an inherited level or two; alas, as soon as you start to work with complex objects with many levels, you end up chasing name clashes and conflicting containers.

I can only hope that this will be fixed once and for all in IDL 6.

Subject: Re: IDL objects and names
Posted by Pavel A. Romashkin on Wed, 12 Feb 2003 18:38:59 GMT
View Forum Message <> Reply to Message

Randall Skelton wrote:
>
> alas, as soon as you start to work with complex objects
> with many levels, you end up chasing name clashes and conflicting
> containers.

I am not arguing !
But the best object applications I've learned on (in VBA, in particular, where they are well established, with event hooks and everything else) were not complex objects with many levels, but systems of small interacting objects. I realized that the idea is to modularize the code to a reasonably atomic level. Complex, large objects become too inflexible. I do agree that IDL's object system is not the most sophisticated of all, but even as it is, it is much better than none.
Cheers,
Pavel

Subject: Re: IDL objects and names
Posted by Rick Towler on Wed, 12 Feb 2003 20:50:47 GMT
View Forum Message <> Reply to Message

"Sebastian" <s@visita2.die.upm.es>

> Is there something like an IDLClass that all IDLgr* classes inherit from?
> I know that the reference manual says for most of the IDLgr* classes "This

> class has no superclasses", but is it true? I mean, what's the point of
> having "objects" and making no use of their (theoretical) capabilities?

FWIW, all atomic graphics objects inherit IDLgrGraphic (structure objects do not).

```
IDL> obj = obj_new('IDLgrPolyline')
IDL> obj -> IDLgrGraphic::GetProperty, ALL=a
IDL> help, a, /struct
** Structure <195dc18>, 11 tags, length=128, data length=123, refs=1:
   COLOR           BYTE      Array[3]
   HIDE            LONG                 0
   NAME            STRING    ''
   PALETTE         OBJREF    <NullObject>
   PARENT          OBJREF    <NullObject>
   XCOORD_CONV     DOUBLE    Array[2]
   XRANGE          DOUBLE    Array[2]
   YCOORD_CONV     DOUBLE    Array[2]
   YRANGE          DOUBLE    Array[2]
   ZCOORD_CONV     DOUBLE    Array[2]
   ZRANGE          DOUBLE    Array[2]
```

COLOR, HIDE, NAME, PALETTE, and PARENT are obvious. [XYZ]RANGE contains the min and max values of the object's position in object space.
[XYZ]COORD_CONV values are as documented in the subclass's documentation and are used in constructing a transformation matrix which transforms the object from object space to normalized space (or some other scaled space).

I can dig up the methods if you like.

What you can do with it is another question...

-Rick

Subject: Re: IDL objects and names
Posted by David Fanning on Wed, 12 Feb 2003 21:38:52 GMT

View Forum Message <> Reply to Message

Rick Towler (rtowler@u.washington.edu) writes:

> FWIW, all atomic graphics objects inherit IDLgrGraphic (structure objects do
> not).

Well, there you go. That's why you can't make an IDLgrSurfacePlot object:

```
   Function idlgrsurfaceplot::init
```

```
  RETURN, 1
  END

  PRO idlgrsurfaceplot__define
  str = {idlgrsurfaceplot, inherits idlgrsurface, inherits idlgraxis}
  end
```

And when I try to create the object, I get this:

```
IDL> a=obj_new('idlgrsurfaceplot')
% Conflicting or duplicate structure tag definition: IDLITCOMPONENT_TOP.
% Execution halted at: IDLGRSURFACEPLOT__DEFINE    6 C:\IDL\David
\idlgrsurfaceplot__define.pro
%                       OBJ_NEW
%                       $MAIN$
```

Humm. This explains an awful lot to me. :-(

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: IDL objects and names
Posted by dick on Wed, 12 Feb 2003 21:47:14 GMT

View Forum Message <> Reply to Message

Sebastian <s@visita2.die.upm.es> wrote in message
news:<Pine.LNX.4.44.0302121521400.2435-100000@visita2.die.upm.es>...
> Hi,
>
> that lostObject thing is cool, made me re-read the OBJ_VALID docs....
> Now I am of course a little bit lazy and don't want to write a PrintName
> procedure for each object type I am using.
> Is there something like an IDLClass that all IDLgr* classes inherit from?
> I know that the reference manual says for most of the IDLgr* classes "This
> class has no superclasses", but is it true? I mean, what's the point of
> having "objects" and making no use of their (theoretical) capabilities?

As Rick Towler points out, IDLgrGraphic may be just what you want, but
for more interesting details on the hidden classes, this is

interesting reading:

```
IDL> help,/obj,/full
** Object class IDLGRAXIS, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRGRAPHIC  <Direct>
     IDLGRCOMPONENT
     IDLITCOMPONENT
** Object class IDLGRBUFFER, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRSRCDEST  <Direct>
     IDLITCOMPONENT
** Object class IDLGRCLIPBOARD, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRSRCDEST  <Direct>
     IDLITCOMPONENT
** Object class IDLGRCOMPONENT, 1 direct superclass, 0 known methods
  Superclasses:
     IDLITCOMPONENT  <Direct>
** Object class IDLGRCONTAINER, 2 direct superclasses, 0 known methods
  Superclasses:
     IDL_CONTAINER  <Direct>
     IDLGRCOMPONENT  <Direct>
     IDLITCOMPONENT
** Object class IDLGRCONTOUR, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRGRAPHIC  <Direct>
     IDLGRCOMPONENT
     IDLITCOMPONENT
** Object class IDLGRDATA, 1 direct superclass, 0 known methods
  Superclasses:
     IDL_CONTAINER  <Direct>
** Object class IDLGRGRAPHIC, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRCOMPONENT  <Direct>
     IDLITCOMPONENT
** Object class IDLGRIMAGE, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRGRAPHIC  <Direct>
     IDLGRCOMPONENT
     IDLITCOMPONENT
** Object class IDLGRLIGHT, 1 direct superclass, 0 known methods
  Superclasses:
     IDLGRGRAPHIC  <Direct>
     IDLGRCOMPONENT
     IDLITCOMPONENT
** Object class IDLGRMODEL, 1 direct superclass, 0 known methods
  Superclasses:
```

IDLGRCONTAINER <Direct>
IDL_CONTAINER
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRPLOT, 1 direct superclass, 0 known methods
Superclasses:
IDLGRGRAPHIC <Direct>
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRPOLYGON, 1 direct superclass, 0 known methods
Superclasses:
IDLGRGRAPHIC <Direct>
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRPOLYLINE, 1 direct superclass, 0 known methods
Superclasses:
IDLGRGRAPHIC <Direct>
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRPRINTER, 1 direct superclass, 0 known methods
Superclasses:
IDLGRSRCDEST <Direct>
IDLITCOMPONENT
** Object class IDLGRSCENE, 1 direct superclass, 0 known methods
Superclasses:
IDLGRCONTAINER <Direct>
IDL_CONTAINER
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRSRCDEST, 1 direct superclass, 0 known methods
Superclasses:
IDLITCOMPONENT <Direct>
** Object class IDLGRSURFACE, 1 direct superclass, 0 known methods
Superclasses:
IDLGRGRAPHIC <Direct>
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRSYMBOL, 0 direct superclasses, 1 known method
Known Function Methods:
IDLGRSYMBOL::INIT
** Object class IDLGRTEXT, 1 direct superclass, 0 known methods
Superclasses:
IDLGRGRAPHIC <Direct>
IDLGRCOMPONENT
IDLITCOMPONENT
** Object class IDLGRVIEW, 1 direct superclass, 0 known methods
Superclasses:
IDLGRCONTAINER <Direct>

```
      IDL_CONTAINER
      IDLGRCOMPONENT
      IDLITCOMPONENT
** Object class IDLGRVIEWGROUP, 1 direct superclass, 0 known methods
   Superclasses:
      IDLGRCONTAINER  <Direct>
      IDL_CONTAINER
      IDLGRCOMPONENT
      IDLITCOMPONENT
** Object class IDLGRVOLUME, 1 direct superclass, 0 known methods
   Superclasses:
      IDLGRGRAPHIC  <Direct>
      IDLGRCOMPONENT
      IDLITCOMPONENT
** Object class IDLGRVRML, 1 direct superclass, 0 known methods
   Superclasses:
      IDLGRSRCDEST  <Direct>
      IDLITCOMPONENT
** Object class IDLGRWINDOW, 1 direct superclass, 0 known methods
   Superclasses:
      IDLGRSRCDEST  <Direct>
      IDLITCOMPONENT
** Object class IDLITCOMPONENT, 0 direct superclasses, 0 known methods
** Object class IDLITPARAMETERDESCRIPTOR, 1 direct superclass, 0 known
methods
   Superclasses:
      IDLITCOMPONENT  <Direct>
** Object class IDLITWINDOW, 1 direct superclass, 0 known methods
   Superclasses:
      IDLGRWINDOW  <Direct>
      IDLGRSRCDEST
      IDLITCOMPONENT
** Object class IDL_CONTAINER, 0 direct superclasses, 0 known methods
```

Play at your own risk, I suppose these undocumented things could change someday. Or perhaps Karl could set our mind at ease about whether some of these are really cast in stone?

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: IDL objects and names
Posted by Rick Towler on Wed, 12 Feb 2003 22:17:34 GMT
View Forum Message <> Reply to Message

"David Fanning" wrote
> Rick Towler writes:
>
>> FWIW, all atomic graphics objects inherit IDLgrGraphic (structure
objects do
>> not).
>
> Well, there you go. That's why you can't make an IDLgrSurfacePlot
> object:
>
> Function idlgrsurfaceplot::init
> RETURN, 1
> END
>
> PRO idlgrsurfaceplot__define
> str = {idlgrsurfaceplot, inherits idlgrsurface, inherits idlgraxis}
> end
>
> And when I try to create the object, I get this:
>
> IDL> a=obj_new('idlgrsurfaceplot')
> % Conflicting or duplicate structure tag definition: IDLITCOMPONENT_TOP.
> % Execution halted at: IDLGRSURFACEPLOT__DEFINE 6 C:\IDL\David
> \idlgrsurfaceplot__define.pro
> % OBJ_NEW
> % $MAIN$
>

I know you did this as an academic exercise to prove the point but we don't want to mislead people into thinking they can't build complex objects from IDL's atoms because of the inheritance issue. A composite object like this should inherit IDLgrModel and then "Add" atoms such as the surface, axes and text.

-Rick

Subject: Re: IDL objects and names
Posted by David Fanning on Thu, 13 Feb 2003 01:36:40 GMT
View Forum Message <> Reply to Message

Rick Towler (rtowler@u.washington.edu) writes:

> I know you did this as an academic exercise to prove the point but we don't

> want to mislead people into thinking they can't build complex objects from
> IDL's atoms because of the inheritance issue. A composite object like this
> should inherit IDLgrModel and then "Add" atoms such as the surface, axes and
> text.

Right. I guess I thought everyone knew about my award-winning
FSC_Surface program. :-)

Cheers,

David

P.S. Let's just say I've always wondered why RSI didn't write
that program. But then I remembered how much effort it took. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155