Subject: Re: IDLgrLegend broken
Posted by Pavel A. Romashkin on Wed, 06 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

I got Mark's message that covers it all after I wrote this one, but I
didn't want all that wrist effort for typing wasted :-)

Here is an explanation to this. In fact, this behavior is what you
expect to happen, although it is not what you really want.
IDLgrLegend is a subclass of IDLgrModel. When a saved instance of
IDLgrLegend is restored, the object class is re-created without
compiling IDLgrLegend_define, which contains all methods for IDLgrLegend
that are not inhereted from IDLgrModel. Then, the next time a method is
called on a Legend, only superclass' methods are considered, because
none specific for the subclass were ever compiled. And there are no such
things as GAP for setProperty method for IDLgrModel.
The workaround is to create a dummy IDLgrLegend object before restoring
the saved one, or to explicitly compile the needed file with all of the
(needed) methods.
The conclusion is, this will be true for any and all objects (especially
those that have superclasses). Save-restore is not going to (always)
give the results we want, unless we take special care.

Cheers,
Pavel

P.S. Here, David, is the downside of keeping the methods in the same
file as object definition. But, if you made separate files for all
methods, you'd never find your way through all 5000 of them :-(

Subject: Re: IDLgrLegend broken
Posted by davidf on Wed, 06 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning (davidf@dfanning.com) writes:

> I'll contact RSI. This looks like something they might
> have to fix.

Whoops! I didn't mean to write "I'll". I meant to
write "I'd", meaning you. :-)

Cheers,

David
--
David Fanning, Ph.D.

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

Subject: Re: IDLgrLegend broken
Posted by davidf on Wed, 06 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Pavel A. Romashkin (pavel.romashkin@noaa.gov) writes:

> I found out that if an instance of IDLgrLegend object is saved to a .sav
> file and then restored, the IDLgrLegend class definition is not restored
> correctly (unless IDLgrLegend is already instanced in the current IDL
> session). Moreover, attempts to use IDLgrLegend in the same IDL session
> fail if an instance of IDLgeLegend was first restored in that session.
> Here is a reproducible example (IDL 5.3 PPC), for those who want to try
> it, step by step.

Well, it works the same way for me in IDL 5.4 on Windows NT 4.0.
(I'll try it in Linux...well, later.)

I'll contact RSI. This looks like something they might
have to fix.

Cheers,

David

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

Subject: Re: IDLgrLegend broken
Posted by Mark Hadfield on Wed, 06 Dec 2000 21:33:52 GMT
View Forum Message <> Reply to Message

Pavel wrote:
> I found out that if an instance of IDLgrLegend object is saved to a .sav
> file and then restored, the IDLgrLegend class definition is not restored
> correctly (unless IDLgrLegend is already instanced in the current IDL
> session). Moreover, attempts to use IDLgrLegend in the same IDL session

> fail if an instance of IDLgrLegend was first restored in that session.

It's a fundamental problem of IDL objects, deriving from the way methods are resolved. It occurs because IDLgrLegend has a superclass (IDLgrModel). When IDL first tries to call a method (e.g. SomeMethod) of a restored IDLgrLegend object it doesn't know that this is available as a method of the IDLgrLegend class (IDLgrLegend::SomeMethod) so it searches for and finds the superclass's method (IDLgrModel::SomeMethod). Thereafter, until IDL is restarted or reset, it flatly refuses to be told that there is an IDLgrLegend::SomeMethod which is supposed to override the superclass's method, no matter how many times you compile the new method.

The simplest workaround is to call IDLgrLegend__Define *before* restoring your IDLgrLegend object (that is, of course, if you know you are about to restore one). This causes IDL to compile the file idlgrlegend__define.pro and, on the way, to compile all the methods for IDLgrLegend that are included in this file.

This problem is related to another one that was discussed in a thread called "Important object lesson" in June 1998: IDL's refusal to recognise a new method for an object that has already been instantiated without it.

Here is my understanding of how it works:

If IDL encounters

MyClass->MyMethod

the three situations are:

1. IDL finds a MyClass::MyMethod in memory and uses it. (In the normal course of events the method will have been included in the myclass__define.pro, before the myclass__define procedure, so it will have been compiled the first time an instance of the class was created.)

2. Not finding MyClass::MyMethod, IDL searches up the inheritance tree in a way described somewhere in the IDL documentation, finds ASuperClass::MyMethod in memory and uses it for the remainder of the session.

3. Failing 1 & 2, IDL searches the !path for myclass__mymethod.pro (and maybe then for similar files for all superclasses). This can take a long time.

Two relevant points are:

1. IDL searches--all the way up the inheritance tree--in memory before searching on the disk. (For performance reasons, obviously.)

2. Once a method binding has been established--i.e. a rule like "if method MyMethod is called on a object of class MyClass, call the superclass's method, ASuperClass::MyMethod--then this is never revised. (I think this is also done for performance reasons.)

I hope that explains it. I find that I can understand it just long enough to write
it down!

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: IDLgrLegend broken
Posted by Mark Hadfield on Thu, 07 Dec 2000 01:34:54 GMT
View Forum Message <> Reply to Message

A couple of corrrections/clarifications to my previous post:

> The simplest workaround is to call IDLgrLegend__Define *before* restoring
> your IDLgrLegend object (that is, of course, if you know you are about to
> restore one).

And the problem is that generally you don't know the class of all the objects you are about to restore. But actually, it's not quite that bad. You just have to call the __Define method *before calling any of the restored object's methods*. And I think this can be done programmatically, as follows:

1. When restoring a .sav file that might contain some objects, use the RESTORE procedure's RESTORED_OBJECTS keyword to get a list of references to the objects you have just restored. (In addition to the IDLgrLegend object you want to restore, there will likely be IDLgrText, IDLgrPlot, etc objects embedded in it.)

2. Immediately go through that list retrieving the class name for each object and for each one call that class's __DEFINE method using CALL_PROCEDURE. Surround this code with a catch block so that you don't trip up on classes (like IDL built-in ones) that don't have a __DEFINE method.

3. Call methods on your objects to your heart's content.

At least I think that will usually work--I haven't tried it. there are further problems that arise when saving & restoring objects, e.g.

parent-child relationships in graphics trees get broken.

And in regard to this one:

> It's a fundamental problem of IDL objects, deriving from the way methods
are
> resolved. It occurs because IDLgrLegend has a superclass (IDLgrModel)

The problem that Pavel reported *is* a fundamental problem of IDL objects,
but it would occur with classes that do not inherit from any superclass.
Let's say we restore an object of such a class (MyClass) and then call a
method (SomeMethod). Let's also assume that all the methods of MyClass are
in myclass__define.pro (as usual), and that MyClass__Define has never been
executed, so myclass__define.pro has never been compiled. What will IDL do?
It will look for SomeMethod amongst the functions that have been compiled
into memory, first for MyClass (doesn't find it) then for its superclasses
(there aren't any). Then it will look on disk for myclass__somemethod.pro.
But that doesn't exist. I don't think IDL is smart enough to look in
myclass__define.pro.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: IDLgrLegend broken
Posted by John-David T. Smith on Thu, 07 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:

> David:
>> But before I did this, I'd have a closer read of this
>> article, where JD and I (and probably Mark) discussed
>> this restore object problem and came up with a
>> "sorta" solution:
>>
>> http://www.dfanning.com/tips/saved_objects.html
>
> Hey nice one David. I don't know that I can claim any of the credit or blame
> for this article. I have read it before and I should have remembered its
> existence before posting very similar material.
>
> A comment/question on the RESOLVE_OBJ routine that's shown at the above
> link:
>

> The following code snippet ensures that each object's __define procedure is
> called only if it has not already been compiled. (The array ri holds a list
> of currently compiled routines, generated by a call to ROUTINE_INFO.)
>
> if (where(ri eq defpro))[0] eq -1 then begin
> ;; Compile and define the class.
> call_procedure,defpro
> endif
>
> My comment is: why bother? Once a __define method has been called once,
> further calls have no effect.

Well, I suppose I should offer my two bits, since I am primarily
reponsible for the method on David's page (though david hit on the
original idea of using class__define in some way).

There are two subtleties in this method. At first blush, you might
consider using resolve_routine to have all the methods compiled (as
David first thought). This works, but does not deal with the issue of
changing class definitions (e.g. adding another data member). The only
solution is to define your class prior to restoring the object (by
calling class__define). Then, relaxed restoration will (usually) do
what you want. This provides upward compatibility, but forces you to
specify the class name in advance. For me this is often not a problem,
since usually the class of the object I'm restoring is the same as the
class from which I'm restoring it (say that 10 times fast), so there is
no ambiguity.

AN IMPORTANT NOTE: saved objects contain in them implicit definitions of
their own class, all their superclasses, and the class+superclasses of
any other objects they contain!

But you can control this behavior:

To make life easier, you can extirpate the most rapidly changing class
data (or any data, such as unwanted objects, for that matter), that
doesn't really need to be saved. For instance, I detach all widget
interface specifications (ala the ancient revered "state" structure),
before saving an object. I can then feel free to redo the interface
entirely. This is done with pointers (or object references): simply
replace one of these with a "stub" -- ptr_new() or obj_new() -- if you
don't want it in the file. I have covered the simple method for doing
this many times, but I'll repeat it:

```
wSav=self.wInfo  ; detach all the widget state stuff.
self.wInfo=ptr_new()
; do whatever to save the object... please do some error checking
if no_error then self.wInfo=wSav ; reattach
```

That way, I can detach all parts which aren't central to my data structure, and be free to develop those as I like. The core components which are vital to the representation of the data are treated with more care.

This method also enables other really cool features. For instance, one of my applications has a "restore from disk" feature which simply updates, in place, an object's self variable (which workds because it's implicitly passed by *reference* to all methods) -- self-transmogrification. Besides sounding cool, it's a very useful technique.

With a bit of care, saved objects become a valid and simple tool with which to store very complex data structures.

JD

P.S.

As far as slugging around the routine_info() results, aside from some efficiency gain for large inheritance trees, this was a holdover from when compiling was done with resolve_routine, rather than (somewhat underhandedly) with call_procedure. Obviously, resolve_routine,'class__define' compiles all methods each and everytime it's called (which wouldn't be pretty if you had 100 objects to restore). Since call_procedure compiles nothing if class__define has already been defined (a feature), the overhead of making the repeated calls is probably minimal. An update would look like:

```
pro resolve_obj, obj, CLASS=class
  if n_params() ne 0 then begin
    if NOT obj_valid(obj) then begin
      message,'Object is not valid.'
    endif
    class=obj_class(obj)
  endif

  for i=0,n_elements(class)-1 do begin
    defpro=class[i]+'__DEFINE'
    ;; (maybe) compile and define the class.
    call_procedure,defpro
    supers=obj_class(class[i],/SUPERCLASS,COUNT=cnt)
    if cnt gt 0 then resolve_obj,CLASS=supers
  endfor
end
```

But I don't expect it to be faster, and occasionally be slower (haven't

tested).

Subject: Re: IDLgrLegend broken
Posted by Pavel A. Romashkin on Thu, 07 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

I just tested this, and I maintain that saving methods to explicitly
named files overrides precompiled superclasses methods with identical names.

What I mean is, for MyObject whos superclass is SuperObject, and both
have Get method, the following happens. IDL *first* searches for
compiled method for MyObject. Then, it searches for a *file* with the
right name, myobject__get.pro, and then, *if not found only*, it
searches for compiled GET method for SuperClass, and, if not found, for superclass__get.pro.

I tested it and it is exactly what happened. I made an instance of
IDLgrModel and an instance of a subclass MYOBJECT that inhereted from
it. I called SetProperty and GetProperty on IDLgrModel to make sure they
are compiled. In advance, I made a file called
myobject__getproperty.pro. When I called myobject -> getproperty *after*
using GetProperty on IDLgrModel, the MYOBJECT::GETPROPERTY module got
compiled and executed, despite the availability of precompiled method
for the superclass.
This means that saving methods in separate files will always work, as
long as they are in the IDL path.

Cheers,
Pavel

Subject: Re: IDLgrLegend broken
Posted by Pavel A. Romashkin on Thu, 07 Dec 2000 16:07:58 GMT
View Forum Message <> Reply to Message

I think the only really generic solution is *saving each method in a
separate file* (advice I received from someone else, which will *always*
work). I guess it is up to the programmer to decide if he wants to do it
that way.

Cheers,
Pavel

Subject: Re: IDLgrLegend broken
Posted by davidf on Thu, 07 Dec 2000 16:37:38 GMT

Pavel A. Romashkin (pavel.romashkin@noaa.gov) writes:

> I think the only really generic solution is *saving each method in a
> separate file*.

Oh, my Gosh!!!

Then I would really be looking into how to use
projects in IDL. :-)

But before I did this, I'd have a closer read of this
article, where JD and I (and probably Mark) discussed
this restore object problem and came up with a
"sorta" solution:

http://www.dfanning.com/tips/saved_objects.html

Cheers,

David

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

Subject: Re: IDLgrLegend broken
Posted by Mark Hadfield on Thu, 07 Dec 2000 20:21:16 GMT

"David Fanning" <davidf@dfanning.com> wrote in message
news:MPG.14996ac0b9d231b2989cb0@news.frii.com...

Responding to David's post, and his quote from Pavel's post, which I have
not yet seen:

Pavel, quoted by David:
>> I think the only really generic solution is *saving each method in a
>> separate file*.

I think that this is not a good idea, because of this rule:

IDL searches--all the way up the inheritance tree--in memory before

searching on the disk. (For performance reasons, obviously.)

So if we have a class MyClass with a method SomeMethod stored in a file myclass__somemethod.pro then the first time IDL wants to call SomeMethod on a MyClass object it may find and compile myclass__somemethod.pro, but *only* if the search through the already-compiled functions in memory fails to find a SomeMethod associated with one of MyClass's superclasses.

The reason for storing all object methods in myclass__define.pro is that myclass__define is always called (and so myclass__define.pro is always compiled) when the first instance of MyClass is created. (Unless you get sneaky and try to restore MyClass from disk.)

David:
> But before I did this, I'd have a closer read of this
> article, where JD and I (and probably Mark) discussed
> this restore object problem and came up with a
> "sorta" solution:
>
> http://www.dfanning.com/tips/saved_objects.html

Hey nice one David. I don't know that I can claim any of the credit or blame for this article. I have read it before and I should have remembered its existence before posting very similar material.

A comment/question on the RESOLVE_OBJ routine that's shown at the above link:

The following code snippet ensures that each object's __define procedure is called only if it has not already been compiled. (The array ri holds a list of currently compiled routines, generated by a call to ROUTINE_INFO.)

```
if (where(ri eq defpro))[0] eq -1 then begin
  ;; Compile and define the class.
  call_procedure,defpro
endif
```

My comment is: why bother? Once a __define method has been called once, further calls have no effect.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: IDLgrLegend broken
Posted by Mark Hadfield on Sun, 10 Dec 2000 21:23:43 GMT
View Forum Message <> Reply to Message

"Pavel A. Romashkin" <pavel.romashkin@noaa.gov> wrote in message news:3A300378.6A319185@noaa.gov...
> I just tested this, and I maintain that saving methods to explicitly
> named files overrides precompiled superclasses methods with identical names.
>
> What I mean is, for MyObject whos superclass is SuperObject, and both
> have Get method, the following happens. IDL *first* searches for
> compiled method for MyObject. Then, it searches for a *file* with the
> right name, myobject__get.pro, and then, *if not found only*, it
> searches for compiled GET method for SuperClass, and, if not found, for superclass__get.pro.

You are right and I was wrong.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: IDLgrLegend broken
Posted by Richard G. French on Mon, 11 Dec 2000 02:30:30 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>

> You are right and I was wrong.
> ---

So, Mark, are you volunteering to write the concession speech for the presidential runner-up?
Dick French

Subject: Re: IDLgrLegend broken
Posted by Pavel A. Romashkin on Mon, 11 Dec 2000 17:34:30 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>
> You are right .... snip

> ---
> Mark Hadfield

Oh, no. Mark, I never meant to make it sound like that. I apologize if you feel I insisted *you* were wrong. I only wanted to see that one can really use explicit naming to avoid *all* confusion. Who will follow this path, anyway, with dozens of methods for every object :-(

Cheers,
Pavel
