Subject: Re: Saved objects.
Posted by davidf on Tue, 08 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Imanol Echave (ccaeccai@sc.ehu.es) writes:

> When you create a new object with OBJ_NEW, IDL searchs automatically the file
> object__define.pro and compiles it, but... What do you have to do when you
> RESTORE an object? The object definition and methods aren't compiled, and an
> error is raised when you use the object. I can compile the file "manually", but

Here is an edited version of a newsgroup exchange
on saving and restoring objects that J.D. Smith and I
carried on recently. It describes a manual way of
compiling the object's methods, but I don't think
there is any way around this, short of putting each
method routine in its own file.

Interestingly, I just ran into a project this week
where it is important that we save the graphical output
in a database along with the information we used to
construct the plot. It must be stored in such as way
that we can reconstruct the exact graphic window.
I am thinking of implementing this functionality
as a Save/Restore object.

Cheers,

David

Subject: Objects, File Names, and the Save command.
From: "J.D. Smith" <jdsmith@astrosun.tn.cornell.edu>

I am exploring a very promising use of the save/restore commands in
conjunction with objects. Given some complex object which contains a
host of different types of data (with pointers, etc.), as part of a
class method, one adds:

save, self, FILENAME=fname

to register on disk an accurate snapshot of the object. To restore,
later, use:

restore,pname,RESTORED_OBJECTS=obj,/RELAXED_STRUCTURE_ASSIGN MENT and the object is in obj, but also brought back as the local variable *self*. I'm not sure the relaxed structure assignment flag works for objects, but I don't see why it wouldn't. So this can be used in two ways ...

1. To allow an object to replace *itself* with another, perhaps older copy (or even an altogether different type of object -- but the utility of self-transmogrifying objects is not yet apparent to me). This works because the implicit self variable is passed by reference (as it has to be). This will lead to at least one unreferenced heap variable unless garbage collection steps are taken, I.e. by saying:

```
 oldself=self
 restore, pname,/RELAXED_STRUCTURE
 obj_destroy,oldself
```

2. To allow a program module to load up an object on the fly, through the obj variable in the above statement (only one should be loaded if only one was saved).

This is all very convenient but leads to the strange situation of a loaded object in memory which exists there *before* any of the class methods, and/or the __define procedure for that object class are compiled. Therefore, the usual paradigm of putting all class methods in the __define procedure file before this procedure (suggested by RSI itself in the manual) fails. How can the method be found if the __define doesn't have to be compiled and isn't in it's own file? I would like to come up with a solution which doesn't involve a separate class__method.pro file for each method. Any ideas?

Thanks,

JD

To which I replied like this:

How about something like this:

```
   thisClass = Obj_Class(self)
   Resolve_Routine, thisClass + '_define'
```

I haven't tested this, but don't see any reason it wouldn't work. Resolve_Routine is the way IDL procedures and functions can be compiled from *within* other procedures and functions.

Cheers,

David

J.D. replied with this:

This will certainly work, but has the unfortunate side-effect of
re-compiling every method each time an object is read from disk... I
thought of modifying it slightly to the tune of:

```
thisdef=Obj_Class(self)+'__DEFINE'
if (where(routine_info() eq thisdef))[0] eq -1 then
resolve_routine,thisdef
```

So that it would only compile if presently undefined.

JD
