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