Subject: Re: XSTRETCH and Library Lessons
Posted by mmiller3 on Tue, 25 Apr 2006 15:05:45 GMT
View Forum Message <> Reply to Message

>>>> > "JD" == JD Smith <jdsmith@as.arizona.edu> writes:

> On Mon, 24 Apr 2006 19:36:29 +0000, Michael A. Miller
> wrote:

>> For our local libraries, I've had to define release tags
>> for them. Then, every "application," which means "every
>> thing that we expect to work the same way each time, gets
>> started from a script that includes setting the IDL_PATH
>> to include the proper release.

> This is a very heavy handed approach, because it requires
> your colleagues to use only your package in a given IDL
> session, and not mix and match. It is, alas, an approach
> many people take.

I don't see how that is heavy handed. The IDL_PATH can handle more than one directory at once and users are they are welcome to add anything they like to their IDL_PATH. We regularly use applications that use out libraries, Fannings, mpfit, textoidl and the Juelich libraries, all in one applications.

The whole point of the IDL_PATH is to allow flexible loading, so multiple "packages" can be used. You are right that they get only one version of a particular routine, but that is the whole point. If parts of a library don't change, then it doesn't matter which version they use. If parts do change, then they pick which behavior they want. This is no different with IDL than with any other system, be it put together with a linker or an interpreted language. The same sort issue comes up all the time for libc, for python, for java, for <insert system here>, especially around major release increment time.

> Assuming library coders kept a (quasi-)fixed calling
> interface and backward-compatible behavior for their
> routines (which is mostly true of most of the big
> libraries), the best approach would be if:

What I was presenting is how I handle the case where backward compatability is broken (sometimes willfully, sometimes unintentionally).

> 1. External libraries are mentioned, by version number
> required, and the user or site has the responsibility to

> install them.

I think that is exactly what we do here. Would you elaborate on what you mean by "install," if it doesn't mean, make sure IDL can find them by setting the appropriate the IDL_PATH?

> 2. Everyone uses likely-to-be-unique names for their
> routines... object programming helps here (since it's not
> weird seeming to hide everything behind a long unique class
> name).

Absolutely required - only gets hard-ish as multiple incompatible releases get promulgated, which is the case that I was talking about, as was the original poster (who, now that I look back, was you :-)

> 3. Nobody messes with IDL_PATH via shell scripts or IDL
> scripts. Your package should work no matter where it is on
> the path, and should not make specific assumptions about
> where it is in the heirarchy.

How would IDL find the code then? If I don't mess with (= add the neccessary directories to) the path, my package cannot work, becuase IDL can't find it. If there are multiple versions of a routine, or even just one, there must be some way for IDL to find the code. Whether that is handled by using the built in IDL_PATH mechanism, or some new feature that is invented to replace it, it seems unavoidable. I must be missing something here - would you elaborate?

> 4. No one redistributes other people's libraries, without
> first renaming them (and getting permission).

Here, here!

> The problem is, this system is only as strong as its
> weakest link. It's one of these "trivial, if every last
> person cooperates" problems that in practice just aren't.
> To solve this, other languages have "name spaces" which are
> similar to, but separate from classes. Imagine if at the
> top of your file full of routines you could say:

Name spaces solve a superset of the problems that include multiple version conflicts. I agree that a name space mechanism for IDL would be very handy, but I'm not holding me breath until RSI puts it in place!

[...]

> etc.  Then it wouldn't matter how many "blah's" there were,
> you'd always be getting the 'blah' that you wanted.

Actually, I'll bet I'd always get the "blah" that I specified, regardless of what I wanted!  If I specified the wrong version, I'd still get the wrong version ;-)

> Note this works as well for normal procedural programming
> as object-oriented programming. It also makes it trivial to
> "fork" a version of a library, and re-distribute.  So you
> might have to change "Package AstroLib" to "Package
> AstroLib-FooBar" and reference that package in your code
> instead.  Our only equivalent would be to go through and
> change all the routine definitions and calls to routines in
> the library from routine to foobar_routine.  Not exactly
> maintainable.

> Sadly, IDL doesn't really offer any help like this, so it's
> up to the community to approximate, by convention, a system
> with some of these properties.

I see it a bit differently - IDL offers a simple method to specify which fork I want.  If I want AstroLib, I put a !path = '/dir/AstroLib'+!path in my code.  If I want AstroLib-FooBar, I put a !path = '/dir/AstroLib-FooBar'+!path in my code.  Both AstroLib and AstroLib-FooBar contain do_this.pro and do_that.pro, so I continue to call them without changing my code.  This is easy to do for any IDL code if I know where the codes are installed.  I don't know how to do it if I adhere to your point 3.

Regards, Mike