Subject: Re: XSTRETCH and Library Lessons
Posted by JD Smith on Mon, 24 Apr 2006 20:54:21 GMT
View Forum Message <> Reply to Message

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

>>>> >> "David" == David Fanning <davidf@dfanning.com> writes:
>
>> This is really starting to be a pain.
>
> 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.

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:

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

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).

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.

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

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:

Package MyUniqueFooBarPackage

pro blah
..

end

...

Then the end user could:

use MyUniqueFooBarPackage;
blah,'mirfq'

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

JD
