Subject: Re: XSTRETCH and Library Lessons
Posted by news.verizon.net on Sat, 22 Apr 2006 16:05:09 GMT
View Forum Message <> Reply to Message

Coincidentally, I've been wrestling with a library name conflict but in this case it is a conflict with myself.

The IDL Astronomy Library (http://idlastro.gsfc.nasa.gov/) includes several procedures for FITS I/O, where FITS is the big-endian data format widely used in astronomy. The procedures currently check whether the user is on a little-endian machine, and if so, will perform the necessary byteswapping when reading or writing to a FITS file. However, by using the /SWAP_IF_LITTLE_ENDIAN keyword to OPENR(W) this can all be done transparently, and this "on the fly" byteswapping can improve the speed by a factor of two for large arrays on my Linux box.

However, for several of the FITS I/O routines there is a separate procedure for opening the FITS file (e.g. fits_open.pro, fxposit.pro) and another for reading/writing (e.g. fits_read.pro, mrdfits.pro) and so both procedures need to be updated. Essentialy the byteswapping has moved from the I/O routine to the Open routine. So if the user downloads the new fits_open.pro but still has an old version of fits_read.pro layng around, then byteswapping will occur twice. Of if the user downloads the new mrdfits.pro but still has an old copy of fxposit.pro laying around, then no byteswapping will occur.

I had hoped that FSTAT() might return information about whether unit has been opened with endian swapping but this doesn't seem to be the case. I was also tempted to add a new keyword to the "Open" procedures on the theory that the error message

"Keyword SWAP_ENDIAN not allowed in call to: FITS_OPEN"

gives a better clue to the origin of the problem, than just finding that the FITS I/O routines are returning wrong endian garbarge with no error message. In the end, I decided that the best solution was to give multiple warnings in the documentation and elsewhere (thus this message), though I'd be interested if anyone has other suggestions. --Wayne

Subject: Re: XSTRETCH and Library Lessons
Posted by George N. White III on Sun, 23 Apr 2006 15:51:48 GMT
View Forum Message <> Reply to Message

On Sat, 22 Apr 2006, Wayne Landsman wrote:

> Coincidentally, I've been wrestling with a library name conflict but in

> this case it is a conflict with myself. [...] finding that the FITS I/O
> routines are returning wrong endian garbarge with no error message. In
> the end, I decided that the best solution was to give multiple warnings
> in the documentation and elsewhere (thus this message), though I'd be
> interested if anyone has other suggestions. --Wayne

Take the necessary steps to ensure that Googling "Why does my FITS I/O routine return wrong endian garbage with no error message?" finds appropriate information.

The first hit for Google "FITS endian" is:

<http://www.ifa.hawaii.edu/~kaiser/imcat/byteorder.html>,

which indicates that not everyone follows the big-endian standard due to the efficiency concerns. Maybe your changes will remove those concerns so this web site can be updated.

Has anyone ever lobbied Google to do something about the many cases where searches turn up page after page of outdated or just plain wrong hits (all quoting the same original material) while the current correct answer is on page 8 that nobody has the patience to reach?

--
George N. White III <aa056@chebucto.ns.ca>

Subject: Re: XSTRETCH and Library Lessons
Posted by David Fanning on Mon, 24 Apr 2006 13:30:21 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> XSTRETCH is pretty fun. I especially like the curve plot for
> non-linear (but why not draw it for linear as well?). I downloaded
> the new version, and immediately had problems: the histogram didn't
> show up for me any longer, as it did in older versions. Just a gray
> background. The min/max lines showed, and could be interacted with,
> but no histogram.
>
> Surely, I thought to myself, the good Dr. Fanning would not distribute
> such a mal-configured tool.

Sigh...

> So I looked into a bit deeper. It turns
> out, among the bread and butter COYOTE routines like FSC_COLOR and
> TVIMAGE, and FSC_FILESELECT, I had 3-4 copies of each of these on my

> IDL_PATH, included directly in other libraries, like PAN, ICG, CM,
> etc. Presumably you have since updated those files, and a standard
> load path shadowing issue (aka name space collisions --- the wrong
> routine of the same name getting called) caused XSTRETCH to break in a
> most unilluminating way.

This is really starting to be a pain. And not just for people
who download the library. It's a pain for me. When I write an
application for a client, I ship a snapshot of my library. But
I have to save that snapshot somewhere, because if my library
evolves, there is a good chance the client code won't work as it
did before a year or so down the road. Trying to work with this
old code and get your IDL path set up correctly is a problem
only truly appreciated by masochists and others of that ilk.

And in my latest offerings (XSTRETCH and TERMINATOR_MAP) I am moving
into a new realm: code that uses not only my libraries, but code
from other libraries as well. I've decided NOT to distribute code from
other libraries, as you suggest, but I see only nightmares ahead of me.
Someone has to support this stuff, and we are fast approaching the
Microsoft moment where we start pointing fingers at everyone else.

And, as you also suggest, some of the problems created by
name space collisions are not easily solved. In fact, not
even reproducible on my system! VERY hard to solve.

(The standard solution to a non-reproducible problem, as anyone
who has ever called a technical support line knows, is to reinstall
everything and see if this helps. About 90% of the time it does,
which ought to tell us something. Incidentally, the other 10% of
the problems we can reliably blame on idiotic system administrators.)

> The final solution, if you feel your users are too lazy to sort any of
> this out, is just to compile a .sav file of your entire library, and
> distribute that. These don't suffer name space collisions. As long
> as they are loaded first, their versions of, e.g., TVIMAGE, will trump
> any others, and since they have to explicitly or implicitly loaded,
> the true-blue TVIMAGE would continue to load and work as expected in
> sessions where your tool isn't being used.

Humm. Then we only have to maintain save files for 5e15 different
combinations of IDL and the library. I like it!! :-)

> P.S. IDLWAVE can help you identify offenders.

This is a help, but I think RSI should be investing some thought
on making it easier for people to maintain their IDL paths. In my
experience this is the real problem. Especially in UNIX systems,

a great many users have NO idea how to manipulate their IDL paths.
And even those who do know can't seem to get it right a surprisingly
large percentage of the time.

In one famous case, I had to make a trip to Munich to get my
Catalyst Library installed. And even though I worked two doors
down from a nice man and extremely competent IDL programmer,
it took us nearly two weeks to be working from the same page! Lord
knows just *thinking* about making a change to this library caused
me to sweat! :-)

(I will say that the excellent conversations and companionship
on long walks in the country or when fumbling with our prayer beads
in the Andechs Monastery more than made up for the disadvantages
of IDL library installation, however.)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: XSTRETCH and Library Lessons
Posted by David Fanning on Mon, 24 Apr 2006 13:32:11 GMT
View Forum Message <> Reply to Message

Wayne Landsman writes:

> In the end, I decided that the best solution was
> to give multiple warnings in the documentation and elsewhere (thus
> this message), though I'd be interested if anyone has other
> suggestions.

Documentation is always good -- although usually pointless.
No real suggestions, but I have a LOT of sympathy, if that
does you any good. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: XSTRETCH and Library Lessons
Posted by mmiller3 on Mon, 24 Apr 2006 19:36:29 GMT
View Forum Message <> Reply to Message

>>>> > "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 can be a shell script or an IDL script.) To handle releases, I use CVS[1] and release tags. This problem is not intrinsically different than release/version-ing problems with any other language. In any case it requires some discipline. Compared to getting some of my colleagues to properly name files[2], this is a piece of cake!

Here's how I handle it... When ever I make a release, I commit my changes to the CVS repository and create a new release tag, usually using a sequential, increasing number, say 2-72. Then I check out that release and put it a directory named using that tag, say mylib-2.72. If I want that to be the default release for anyone using release 2.x, I can link mylib-2.72 to mylib-2. If I want that to be the default release for anything that doesn't care about versions (pretty rare!), I can link mylib-2.72 to mylib.

This is purely patterned after what I see in my /usr/lib directories, and can, I think, be done under any OS. This can be done with Subversion[3] as well as CVS, or any other version control system. (Note that version is a different thing from release. Version means version of a file - release means snap shot of a collection of files).

Here's how I would do this for a collection of codes called myIDLlib

0 - make sure my version control system is set up (in this case CVS)

1 - get a copy of myIDLlib and put it into a directory that I've cleverly named myIDLlib

2 - change directory to myIDLlib. If there is cruft in this directory that I don't want in the repository, delete it now.

3 - add everything to the repository:
> cvs import -m "Imported sources" myIDLlib yoyo start

4 - Make the initial release tag. Tags are not allowed to have
names with periods in them, so I use dashes instead:

> cvs tag rel-1-0

5 - Move this directory to where ever I store production code and
name it myIDLlib-1.0
> cd ..
> mv myIDLlib myIDLlib-1.0

6 - set up application scripts to have myIDLlib-1.0 in the IDL_PATH

7 - Now when I want to change my code, I check out a fresh copy:
> cvs checkout myIDLlib

8 - I corrupt it/change it/improve it...

9 - I check in my changes frequently so I can undo them as
needed...

10 - repeat 8 and 9 until I decide I'd better hang onto a snap
shot of this before I get myself into deeper trouble.

11 - Make a new release tag:
> cvs tag rel-1-1

12 - Move this directory to myIDLlib-1.1
> cd ..
> mv myIDLlib myIDLlib-1.1

13 - change the start up scripts for applications that need the
latest and greatest. Applications that need older releases
already have the correct release in their IDL_PATH, so nothing
needs to be changed.

Regards, Mike

[1] http://www.nongnu.org/cvs/
[2] http://www.dfanning.com/tips/namefiles.html
[3] http://subversion.tigris.org/

--
Michael A. Miller mmiller3@iupui.edu
Imaging Sciences, Department of Radiology, IU School of Medicine

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

Subject: Re: XSTRETCH and Library Lessons
Posted by Karsten Rodenacker on Tue, 25 Apr 2006 07:24:46 GMT
View Forum Message <> Reply to Message

Am Mon, 24 Apr 2006 15:30:21 +0200 schrieb David Fanning
<davidf@dfanning.com>:

> In one famous case, I had to make a trip to Munich to get my
> Catalyst Library installed. And even though I worked two doors
> down from a nice man and extremely competent IDL programmer,
> it took us nearly two weeks to be working from the same page!

That is a bit exaggerated, the 'extremely competent IDL programmer' used after some initial problems 'Routine Shadows' from IDLWave! I would say the problem was again and again Windows in combination with the IDL Project. Both hide unclean programming and link/search path concerns.

By the way, I consider the proposed solutions with savefiles as extremly dangerous. To find out if a and which savefile was (automatically) used is not easy. Is it possible that a restore of a savefile with routines overwrites routines already compiled?

Regards
Karsten

Subject: Re: XSTRETCH and Library Lessons
Posted by David Fanning on Tue, 25 Apr 2006 13:40:51 GMT
View Forum Message <> Reply to Message

Karsten Rodenacker writes:

> That is a bit exaggerated, the 'extremely competent IDL programmer' used
> after some initial problems 'Routine Shadows' from IDLWave! I would say
> the problem was again and again Windows in combination with the IDL
> Project. Both hide unclean programming and link/search path concerns.

Humm, it could have been 10 days instead of two weeks, I guess,
but it was longer than it should have been. :-)

I agree that projects might be intimately involved in the
difficulties here. I think the real problem is that it
is impossible to reconcile both a project approach and
a PATH approach to resolving program names. If you had
(strictly) one or the other life would be easier to
understand. The hodgepodge we have now in the IDLDE
is difficult and getting worse, not better. (In the
version I've been using lately I can't even tell which
directory I am saving files into!! Directories seem to
change underneath me!)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

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
