Subject: IDL project build scripts?
Posted by Matt Francis on Mon, 07 Jul 2014 06:10:15 GMT
View Forum Message <> Reply to Message

I have a large eco-system of IDL code, mostly implemented using custom objects, that a large number of different applications are built from and run as regular jobs on a cron or as post-processed tasks. At the moment each 'application' is run from a script that first populates the !path as appropriate and then runs whatever top level procedure is required. Compilation happens on the fly. This is becoming a bit of a pain in terms of version control (i.e. ensuring no one touches the operational code which the cron jobs require and instead work on the development code, but then remember to push changes to the operational area). It would clearly be more efficient to instead build .sav files for each application allowing code to be changed more freely rebuilding the .savs when modifications are complete (not to mention the runtime savings in not having to continually re-compile the same code over and over again each time each script is run).

Now, the issue I am having with this is ensuring that all required modules are compiled into the .sav file. Simple compiling the top level routine then using RESOLVE_ALL doesn't seem to work as it misses many of the objects (i.e. when I RESTORE the .sav and run the code more routines get compiled that weren't in the .sav). Looking at the docs it appears IDL doesn't resolve object definitions very well. There is the CLASS keyword to RESOLVE_ALL that is supposed to get around this, but you then need to manually list the classes to compile, which defies the point of having IDL resolve the routines in the first place. Even running the script once in order to ensure everything is compiled doesn't work, since there are occasional edge cases that causes some objects to be required that in normal runs won't be needed, so that's not a robust way of ensuring all routines are resolved.

I did see mention of a user written RESOLVE_EVENMORE routine on David Fannings page at https://www.idlcoyote.com/tips/compile.html that looked promosing, but the link to the code appears to be dead.

Can anyone suggest a good approach here?

I would really prefer to avoid having to create workbench projects for each of these applications since there are a lot of them and I'd like to be able to make global changes, such as changing where code is stored, for all applications easily by making build scripts that refer to environmental variables for all this deployment specific info. Basically, I want the IDL equivalent of GNU Make.

Subject: Re: IDL project build scripts?
Posted by Craig Markwardt on Wed, 09 Jul 2014 16:07:09 GMT
View Forum Message <> Reply to Message

On Sunday, July 6, 2014 11:10:15 PM UTC-7, Bogdanovist wrote:
> I have a large eco-system of IDL code, mostly implemented using custom objects, that a large number of different applications are built from and run as regular jobs on a cron or as post-processed tasks. At the moment each 'application' is run from a script that first populates the !path as appropriate and then runs whatever top level procedure is required. Compilation happens on the fly. This is becoming a bit of a pain in terms of version control (i.e. ensuring no one touches

the operational code which the cron jobs require and instead work on the development code, but then remember to push changes to the operational area). It would clearly be more efficient to instead build .sav files for each application allowing code to be changed more freely rebuilding the .savs when modifications are complete (not to mention the runtime savings in not having to continually re-compile the same code over and over again each time each script is run).
>
> ...

For years I built save files, mostly to reduce compilation time on a slow computer running a cron job. In your case, you are using it for configuration management.

It feels like the wrong approach though. You are relying on IDL's (or Gnu Make's) understanding of your project dependencies, and as you're finding, it's not very reliable. I had the same kinds of problems.

The customary way to do this is to have a development tree and a production tree. Each is a single path. You develop in one tree while the production tree has a stable version in place. When the new development code is tested and the production system is ready for change, you deploy (i.e. copy) the development code into the production tree. No need to change any paths. The moment of deployment is also the opportunity to update the code that writes version metadata embedded in the output products. Now you are free to work on more changes in the development tree without interfering with the production branch.

Of course there are more sophisticated ways to do this with source code management tools. For example, you can be working in the development branch and checking in your changes; deployment is then "checking out" the changes on the production side.

Best wishes,
Craig Markwardt