Subject: Re: new to IDL - question about modularity
Posted by little davey on Mon, 25 Jun 2007 23:19:47 GMT
View Forum Message <> Reply to Message

On Jun 25, 10:17 am, Paul van Delst <Paul.vanDe...@noaa.gov> wrote:
> David Fanning wrote:
>> Josh writes:
>
>>> despite having five manuals on IDL in front of me, I'm having some
>>> trouble with writing modular code. I think it lies in my lack of
>>> understanding of the structure, but hopefully it's a simple issue...
>
>>> I'm using IDL 6.3, and have written a single .pro file that looks
>>> like,
>
>>> pro xxx
>>> ...
>>> end
>
>>> pro yyy
>>> ...
>>> end
>
>>> pro zzz
>>> ...
>>> end
>
>>> in an effort to break up the tasks to be done. However, within the
>>> zzz procedure I would like to access an array that was built in the
>>> xxx procedure. I realize that's not possible, and was hoping for some
>>> guidance as to a better way to do this.
>
>> The simplest way to do this, and maybe the only way if
>> there is really no connection between the three modules
>> in your example, is to use a common block to store the array.
>
> A common block? Shock, horror, and gasp! :o0
>
> Just kidding :o) But, I don't think the OP has provided enough information to suggest
> them as the first possible solution in a list. I would prefer encapsulation of all the
> relevant data in a structure (or object) depending on, among other things, computational
> cost, complexity, etc.
>
> - If the array in question that was built in xxx is cheap to construct, then perform that
> construction in a separate function/pro and call it where it's needed - in both xxx and zzz.
> - If the array built in xxx is *not* cheap to construct, then consider the use of a
> structure (think of it like an object, let's call it the "abc" object) to carry all your
> information - that is, the result of xxx and the intermediate data that other "abc"

> methods may need.
>
>> If there is a connection (e.g., zzz calls xxx), then you
>> can pass information around by means of output keywords
>> or even pass a pointer variable around that can be filled
>> out by individual modules.
>
>> There really should be some kind of connection, or these
>> procedures shouldn't be in the same file. That is to say,
>> the only reason xxx and yyy should be in the same file
>> as zzz is that they are utility routines for zzz. Thus,
>> a common block is rarely (I almost said never) needed.
>> They should be able to pass all their information via
>> keywords and parameters. :-)
>
> I reckon a structure is the go. Or maybe even an object depending on the comfort level of
> the OP in using them.
>
> cheers,
>
> paulv
>
> --
> Paul van Delst   Ride lots.
> CIMSS @ NOAA/NCEP/EMC   Eddy Merckx

I like common blocks, and I'll tell you why. I am a meteorologist who
fully takes advantage of the big "I" in IDL...interactive. I develop
algorithms, that is, I play the "Let's Try This" game professionally.
COMMON blocks are the way to go for this kind of work.

I write modules that I want to type in at the command line. If I have
to append either a parameter list of variables, or a structure such as
temp.level.pres.850, this defeats the purpose of Interactive.

If you WANT or NEED to write software that has attributes of being
fully independent, maintainable, algorithm is hidden, or any of the so-
called advantages that I learned in my C++ class, fine. However, as a
"Try This" person, I can honestly say that none of those have much
relevance during scientific development. (Getting the final algorithm
put into operations, ok, THEN re-write).

-- Dave K --