Subject: COMMON block question
Posted by Paul van Delst on Tue, 30 Jan 2001 15:53:46 GMT
View Forum Message <> Reply to Message

Hey there,

I am doing the following in an initialisation function:

```
  COMMON rtm_transmittance_data, predictor_index, $
                          transmittance_coefficients

  result = read_transmittance_coefficients( 'transmittance_coefficients', $
                                    predictor_index,             $
                                    transmittance_coefficients    )
  IF ( result NE SUCCESS ) THEN $
    MESSAGE, 'Error reading transmittance coefficients.', $
           /NONAME, /NOPRINT
```

If I run the function I see the following:

```
IDL> print, initialize_rtm()
% Compiled module: COMPUTE_ABSORBER_SPACE.
% Compiled module: READ_TRANSMITTANCE_COEFFICIENTS.
% Compiled module: OPEN_COEFFICIENT_FILE.
% Compiled module: READ_SPECTRAL_COEFFICIENTS.
       1
IDL> common rtm_transmittance_data
IDL> help
% At  $MAIN$
PREDICTOR_INDEX (RTM_TRANSMITTANCE_DATA)
              LONG      = Array[6, 142, 3]
TRANSMITTANCE_COEFFICIENTS (RTM_TRANSMITTANCE_DATA)
              DOUBLE    = Array[6, 301, 142, 3]
Compiled Procedures:
   $MAIN$

Compiled Functions:
  COMPUTE_ABSORBER_SPACE  INITIALIZE_RTM         OPEN_COEFFICIENT_FILE
    READ_SPECTRAL_COEFFICIENTS      READ_TRANSMITTANCE_COEFFICIENTS
```

How does IDL go about creating COMMON blocks when the size/type of the common block elements are not known until, in this case, they're read in. Understand I'm coming from Fortran-90 background where I declare everything up front in its own module (sort of like a f77/IDL common block but better). Does IDL simply create pointers to the data elements up front and then fills in the blanks after the data has actually been read in/defined? The documentation doesn't give much info.

Thanks for any insights. All metphysical replies forwarded to DF. :o)

paulv

--
Paul van Delst A little learning is a dangerous thing;
CIMSS @ NOAA/NCEP Drink deep, or taste not the Pierian spring;
Ph: (301) 763-8000 x7274 There shallow draughts intoxicate the brain,
Fax: (301) 763-8545 And drinking largely sobers us again.
Email: pvandelst@ncep.noaa.gov Alexander Pope.

Subject: Re: common block question
Posted by David Fanning on Wed, 13 Apr 2011 21:17:00 GMT
View Forum Message <> Reply to Message

geoff writes:

> I am writing a program which will call a number of subroutines. I
> have a number of arrays, the largest of which (and will take a
> significant part of the memory) will be used in most (but not all) of
> the subroutines, will be created at the beginning and be retained
> until the end of the program.
>
> Is it easiest for me to define a common block in the main program with
> this main array and make this block available to the other sub
> routines when necessary? Is there any overhead in making the block
> available to all sub routines?
>
> I really just want to use a few global variables which are available
> to the whole list of programs

Sounds like a legitimate use of a common block
to me. Although passing the routine by reference
or passing a light-weight pointer would always
be my preference. It makes the code easier to
understand months later, for one thing. And it
allows me to run multiple instances of my program
(maybe not important here).

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.

Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thue. ("Perhaps thos speakest truth.")

Subject: Re: common block question
Posted by penteado on Wed, 13 Apr 2011 23:21:12 GMT
View Forum Message <> Reply to Message

On Apr 13, 6:17 pm, David Fanning <n...@dfanning.com> wrote:
> Sounds like a legitimate use of a common block
> to me. Although passing the routine by reference
> or passing a light-weight pointer would always
> be my preference. It makes the code easier to
> understand months later, for one thing. And it
> allows me to run multiple instances of my program
> (maybe not important here).

To me it sounds like the classical case where there are much better options than global variables. That is, any case when there is a way around using global variables.

Depending on the relation of the variables and how they are going to be used, I would consider one of:

1) Putting then into a structure (of the arrays themselves or pointers to them), and passing it around.
2) Passing around an array of pointers, where each element pointed to one of the arrays.
3) Passing around a list or hash, depending on which makes it easier to organize the arrays.
4) Making an object of the whole thing and keeping those arrays in one or more fields of self.

Subject: Re: common block question
Posted by pgrigis on Thu, 14 Apr 2011 14:20:17 GMT
View Forum Message <> Reply to Message

On Apr 13, 3:15 pm, geoff <oxfordenergyservi...@googlemail.com> wrote:
> I am writing a program which will call a number of subroutines. I
> have a number of arrays, the largest of which (and will take a
> significant part of the memory) will be used in most (but not all) of
> the subroutines, will be created at the beginning and be retained
> until the end of the program.

I would make this an object with the subroutines as methods.

Ciao,
Paolo

>
> Is it easiest for me to define a common block in the main program with
> this main array and make this block available to the other sub
> routines when necessary? Is there any overhead in making the block
> available to all sub routines?
>
> I really just want to use a few global variables which are available
> to the whole list of programs
>
> Thanks!

Subject: Re: common block question
Posted by oxfordenergyservices on Fri, 15 Apr 2011 10:02:17 GMT
View Forum Message <> Reply to Message

On Apr 14, 3:20 pm, Paolo <pgri...@gmail.com> wrote:
> On Apr 13, 3:15 pm, geoff <oxfordenergyservi...@googlemail.com> wrote:
>
>> I am writing a program which will call a number of subroutines. I
>> have a number of arrays, the largest of which (and will take a
>> significant part of the memory) will be used in most (but not all) of
>> the subroutines, will be created at the beginning and be retained
>> until the end of the program.
>
> I would make this an object with the subroutines as methods.
>
> Ciao,
> Paolo
>

Thanks for all your above comments. One of my issues is that I may have to re-code in fortran or C and run on a parallel machine (or may not!). I would prefer to write the program in IDL in a way that is relatively transferrable to fortran95 ultimately, which is why I considered common blocks which turn into modules that can be called in fortran.

Subject: Re: common block question
Posted by penteado on Fri, 15 Apr 2011 15:33:39 GMT
View Forum Message <> Reply to Message

On Apr 15, 7:02 am, geoff <oxfordenergyservi...@googlemail.com> wrote:
> Thanks for all your above comments. One of my issues is that I may
> have to re-code in fortran or C and run on a parallel machine (or may
> not!). I would prefer to write the program in IDL in a way that is
> relatively transferrable to fortran95 ultimately, which is why I
> considered common blocks which turn into modules that can be called in
> fortran.

I also had on many occasions to suffer from global variables through
Fortran modules.

Structures, pointers and objects as I suggested above are all pretty
much possible to directly translate to Fortran.

Subject: Re: common block question
Posted by Paul Van Delst[1] on Fri, 15 Apr 2011 16:13:34 GMT
View Forum Message <> Reply to Message

Paulo Penteado wrote:
> On Apr 15, 7:02 am, geoff <oxfordenergyservi...@googlemail.com> wrote:
>> Thanks for all your above comments. One of my issues is that I may
>> have to re-code in fortran or C and run on a parallel machine (or may
>> not!). I would prefer to write the program in IDL in a way that is
>> relatively transferrable to fortran95 ultimately, which is why I
>> considered common blocks which turn into modules that can be called in
>> fortran.
>
> I also had on many occasions to suffer from global variables through
> Fortran modules.

And, not to pile on, but module variables in Fortran are inherently not threadsafe.

> Structures, pointers and objects as I suggested above are all pretty
> much possible to directly translate to Fortran.

As well as some of the OOP stuff as well (assuming you have a compiler that is compliant with the OOP stuff in Fortran2003).

cheers,

paulv
