Subject: Re: include file?
Posted by promashkin on Thu, 29 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

JD, as always, put out every word of wisdom out there in a single, comprehensive message. This is how we all should write, but no, we have our preferred ways of doing things, and send them out as advice :-(

Cheers,
Pavel

Subject: Re: include file?
Posted by John-David T. Smith on Thu, 29 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

As you can see from the various postings, there are a variety of ways to include data "globally" to be shared among many routines, all of which I use in some form. Each has it's strengths and weaknesses. In case people are confused by which they should select, I thought I'd outline the various advantages/disadvantages of each:

1. Variables in an include File (@filename)

* Advantages: Easy access to variables. Values can be initialized within the file itself, and need no explicit initialization. Any arbitrary code can execute, not just variable setting. This can be especially convenient when used in concert with common blocks (see below).

* Disadvantages: Changes in the values cannot be shared among routines or multiple calls to a single routine -- works best for read-only constants and/or shared code segments. Updating the value requires recompiling each including routine, or restarting the session.

2. System variables (!FOO).

* Advantages: Available everywhere for both reading and writing.

* Disadvantages: Difficulty setting -- IDL used to validate system variables at compile time, which means you'd need to define them with "defsysv" *before* any routine referencing them was compiled. At least with version 5.3, validity checking is done at run time, eliminating this problem. Once a system variable is defined, much like a named structure, it cannot be redefined with a different data type or size in that session. Each system variable you define must be initialized explicitly.

3. Restoring a .sav. This is really equivalent to #1 for variable definition only, except the file to be included has actually been compiled.

*Advantages: Loads more quickly than file includes. Variables are pre-initialized.

*Disadvantages: Unlike method #1, only variables are set ... you can't run arbitrary code. In order to update the variable data, you must redefine them and then recompile the .sav, which is typically more difficult than recompiling a routine (though perhaps not more difficult than recompiling many routines). This need not even be a .sav file... any file (such as flat ascii) can be read and parsed and variables assigned, but .sav's are somewhat more convenient (if not portable among different programs), and preserve variable names simply. Since you can't just peek in a .sav to see what variables it defines, you risk variable name collision in routines which restore it. Restoring object variables introduces a whole rash of subtleties (see previous postings on the subject).

4. Common blocks.

* Advantages: RSI designed common blocks exactly for the case of sharing global data. Common block variables can be both read and written to, and assigned values of different size/type (unlike system variables). Shorthand allows you to avoid explicitly mentioning each variable in a common block usage statement (but see below).

* Disadvantages: Common blocks must be initialized and cannot be redefined within a given session. The shortcut usage statement requires that the declaring "common" statement (with all the variables listed) be *compiled* before any routine invoking it. This compile-ordering restriction is similar to the case for system variables in older versions of IDL. The only solution is making each common block statement a declaring statement. This then makes adding or deleting common block variables difficult, if its use is spread throughout many routines/files. Combining with method #1 achieves single-point updates (still for only one session), and solves the compile order issue.

Hope this helps clear things up.

JD

--
J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842
304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */

Subject: Re: include file?
Posted by R.G. Stockwell on Thu, 29 Jun 2000 07:00:00 GMT

Stu <stu.c@freeuk.com> wrote in message news:395B0B0C.C0A5715D@freeuk.com...
> Hi,
> I have several different routines that all set up the same or slightlly
> different constants etc at the beginning,
> is it possible to have all the constants in one file, and have all my
> functions/procedures reference this file
> when they start? It would make for easier maintainence this way. How
> would it be done? I can't see a
> normal function or procedure doing the job as you would have to ask for
> each constant explicitly, which is a
> bit of a pain.
>
> cheers,
> Stu
>

You can make system variables in your startup.pro file
For instance, in mine I have the following:

DEFSYSV, '!imag', complex(0,1)
DEFSYSV, '!twopi', 2*!dpi

(this also avoids the problem of having common blocks,
where you may inadvertently confuse local and global variables
(artificially I guess, but the " ! " at the front of the name is
a good way to identify global variables))

Cheers,
bob stockwell

Subject: Re: include file?
Posted by promashkin on Thu, 29 Jun 2000 07:00:00 GMT

I think that by far the easiest way is to define all your constants
interactively from command line (or from a routine) and save them into
IDL .sav file. Then, you can RESTORE that file from whatever routine you
want and use the contents of the .sav file in any way you want.
Cheers,
Pavel

Stu wrote:
>

> Hi,
> I have several different routines that all set up the same or slightlly
> different constants etc at the beginning,
> is it possible to have all the constants in one file, and have all my
> functions/procedures reference this file
> when they start? It would make for easier maintainence this way. How
> would it be done? I can't see a
> normal function or procedure doing the job as you would have to ask for
> each constant explicitly, which is a
> bit of a pain.
>
> cheers,
> Stu

Subject: Re: include file?
Posted by John-David T. Smith on Thu, 29 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Stu wrote:
>
> Hi,
> I have several different routines that all set up the same or slightlly
> different constants etc at the beginning,
> is it possible to have all the constants in one file, and have all my
> functions/procedures reference this file
> when they start? It would make for easier maintainence this way. How
> would it be done? I can't see a
> normal function or procedure doing the job as you would have to ask for
> each constant explicitly, which is a
> bit of a pain.
>
> cheers,
> Stu

```
pro my_pro
 @include_file
```

This mechanism is also very good for the (rare) time when a common block is needed by a suite of routines with multiple possible entry points, requiring the full variable in each "common" statement. Just be careful to keep track of which variables you've introduced.

JD

--
J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842

304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */

Subject: Re: include file?
Posted by Robert Weiss on Thu, 29 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Hello,

what you are looking for are common blocks. Start the onlice documentation
and look up 'common blocks' in the index. If you're familiar with
FORTRAN: the concept is quite the same.

Another possibility (one that I prefer because I'm always suspicious of
global variables) is to use a structure of constants (with pointers if the
the constants are larege arrays) and explicitly include the structure in
the call to your functions and procedures.

Hope that helps,

Robert.

P.S.: If it's physical constants your using it's most convenient to define
them in the idl_startup.pro file, sice you can then always refer to them
without problems (on a UNIX system it is the necessary to set the
environment variable IDL_STARTUP to this file, for WinDO$ and
MAC I dunno...).

Stu wrote:

> Hi,
> I have several different routines that all set up the same or slightlly
> different constants etc at the beginning,
> is it possible to have all the constants in one file, and have all my
> functions/procedures reference this file
> when they start? It would make for easier maintainence this way. How
> would it be done? I can't see a
> normal function or procedure doing the job as you would have to ask for
> each constant explicitly, which is a
> bit of a pain.
>
> cheers,
> Stu

Subject: Re: include file?

Posted by Steve Hartmann on Fri, 30 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

This topic has been explained very well with lots of good ideas (I've saved J.D. Smith's message for future reference). I ran into a different type of problem when I wanted to change some default program parameters of a run-time application, such as the number and size of some display windows, default directory locations, etc. I wanted to give the user the .sav program file as well as a 'defaults' file that the user could change as desired. The problem is that a run-time application can not compile anything, so I just put all the defaults in an ascii file and had the application program open and parse the values. My defaults file is of the form:

parameter1:value1
parameter2:value2

and I just search for the parameter string and then read in the value for that parameter. If it's not found, I load some default value.

I think this is an easy way to change the parameters, and probably the best way to do it for a run-time application, especially for values that you might want to change frequently.

-Steve Hartmann

Stu wrote:

> Hi,
> I have several different routines that all set up the same or slightlly
> different constants etc at the beginning,
> is it possible to have all the constants in one file, and have all my
> functions/procedures reference this file
> when they start? It would make for easier maintainence this way. How
> would it be done? I can't see a
> normal function or procedure doing the job as you would have to ask for
> each constant explicitly, which is a
> bit of a pain.
>
> cheers,
> Stu

Subject: Re: include file?
Posted by Martin Schultz on Fri, 30 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"J.D. Smith" wrote:
>
> As you can see from the various postings, there are a variety of ways to include
> data "globally" to be shared among many routines, all of which I use in some
> form. Each has it's strengths and weaknesses. In case people are confused by
> which they should select, I thought I'd outline the various
> advantages/disadvantages of each:
>
This is very valuable. Thanks JD!

Just two additional comments that affect all three solutions to some extent:

The problem is not as easy as asking "What is the best way to handle globally
shared information?". It is important to consider the following aspects:
1.) do i know the number of variables and their type in advance?
2.) are the "constants" constant over one session, one project, or my entire
career? (or until i get a new computer or the dept. a new server)
3.) do i need the same constants for all applications or would i like to use
the same variables with different values in other applications (e.g.
path names and file names)

Furthermore, you have to decide whether you prefer a "once and for all" approach or
a "quick and clean" solution. In my experience, both suffer from the fact that, the
better they work, the more likely I am to forget how exactly I store my information, so when it comes to the unavoidable event that I have to change something, I need to start searching (this is a typical catch 22: you write a program to eliminate the need to think about something, then you find it more difficult to think about this when needed). Anyway: in practice there are the solutions that JD gives us, and each of them allows at least two levels of complexity: you can either store the data directly
(in which case you fix the number of elements and their type), or you can use
a structure which is stored as a pointer. This allows you to store arbitrary data with one single "access point", and you can still change things during runtime. The next (and final?) level of sophistication would be some sort of container object which is initialized during setup (from the startup file) and its reference stored in a global
variable. Then you would query the "constant" values e.g. as
```
e = !Constants->Get('e')
```
or
```
default_path = !Constants->Get('Default_Path')
```

And you can add values, rename them, delete them, etc. Just don't forget
to deal with the case of undefined variables, i.e. don't forget a method
asking for the validity of an entry:
   if not !Constants->IsValid('e') then ...

Cheers,
Martin

> 1. Variables in an include File (@filename)
>
> * Advantages: Easy access to variables. Values can be initialized
> within the file itself, and need no explicit initialization. Any arbitrary code
> can execute, not just variable setting. This can be especially convenient when
> used in concert with common blocks (see below).
>
> * Disadvantages: Changes in the values cannot be shared among routines or
> multiple calls to a single routine -- works best for read-only constants and/or
> shared code segments. Updating the value requires recompiling each including
> routine, or restarting the session.
>
> 2. System variables (!FOO).
>
> * Advantages: Available everywhere for both reading and writing.
>
> * Disadvantages: Difficulty setting -- IDL used to validate system variables at
> compile time, which means you'd need to define them with "defsysv" *before* any
> routine referencing them was compiled. At least with version 5.3, validity
> checking is done at run time, eliminating this problem. Once a system variable
> is defined, much like a named structure, it cannot be redefined with a different
> data type or size in that session. Each system variable you define must be
> initialized explicitly.
>
> 3. Restoring a .sav. This is really equivalent to #1 for variable definition
> only, except the file to be included has actually been compiled.
>
> *Advantages: Loads more quickly than file includes. Variables are
> pre-initialized.
>
> *Disadvantages: Unlike method #1, only variables are set ... you can't run
> arbitrary code. In order to update the variable data, you must redefine them
> and then recompile the .sav, which is typically more difficult than recompiling
> a routine (though perhaps not more difficult than recompiling many routines).
> This need not even be a .sav file... any file (such as flat ascii) can be read
> and parsed and variables assigned, but .sav's are somewhat more convenient (if
> not portable among different programs), and preserve variable names simply.
> Since you can't just peek in a .sav to see what variables it defines, you risk

> variable name collision in routines which restore it. Restoring object
> variables introduces a whole rash of subtleties (see previous postings on the
> subject).
>
> 4. Common blocks.
>
> * Advantages: RSI designed common blocks exactly for the case of sharing global
> data. Common block variables can be both read and written to, and assigned
> values of different size/type (unlike system variables). Shorthand allows you
> to avoid explicitly mentioning each variable in a common block usage statement
> (but see below).
>
> * Disadvantages: Common blocks must be initialized and cannot be redefined
> within a given session. The shortcut usage statement requires that the
> declaring "common" statement (with all the variables listed) be *compiled*
> before any routine invoking it. This compile-ordering restriction is similar to
> the case for system variables in older versions of IDL. The only solution is
> making each common block statement a declaring statement. This then makes adding
> or deleting common block variables difficult, if its use is spread throughout
> many routines/files. Combining with method #1 achieves single-point updates
> (still for only one session), and solves the compile order issue.
>
> Hope this helps clear things up.
>
> JD
>
> --
> J.D. Smith /*\ WORK: (607) 255-6263
> Cornell University Dept. of Astronomy */ (607) 255-5842
> 304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
> Ithaca, NY 14853 */

--
[[[[[[[[[[
[[Dr. Martin Schultz Max-Planck-Institut fuer Meteorologie [[
[[Bundesstr. 55, 20146 Hamburg [[
[[phone: +49 40 41173-308 [[
[[fax: +49 40 41173-298 [[
[[martin.schultz@dkrz.de [[
[[[[[[[[[[