Subject: Re: HELP: Multiple-file Applications
Posted by djackson on Mon, 05 Dec 1994 22:35:16 GMT
View Forum Message <> Reply to Message

In article <3c04bp$pap@canopus.cc.umanitoba.ca>
djackson@ibd.nrc.ca (Dick Jackson) writes:

> Problems:
> - I don't see any way for an IDL routine to cause a file to be compiled
> (e.g. OK = EXECUTE(".RUN foo.pro") does not work)

Just to clarify, I want the compilation to be done somewhere at
run-time, say when a user selects a given menu item. Note that the '@'
command to include a file doesn't do this, it just includes another
file as part of the current file.

-Dick

Dick Jackson djackson@ibd.nrc.ca Institute for Biodiagnostics
Opinions are mine alone. National Research Council Canada, Winnipeg

Subject: Re: HELP: Multiple-file Applications
Posted by sjt on Tue, 06 Dec 1994 11:06:33 GMT
View Forum Message <> Reply to Message

Dick Jackson (djackson@ibd.nrc.ca) wrote:
: In article <3c04bp$pap@canopus.cc.umanitoba.ca>
: djackson@ibd.nrc.ca (Dick Jackson) writes:

: > Problems:
: > - I don't see any way for an IDL routine to cause a file to be compiled
: > (e.g. OK = EXECUTE(".RUN foo.pro") does not work)

: Just to clarify, I want the compilation to be done somewhere at
: run-time, say when a user selects a given menu item. Note that the '@'
: command to include a file doesn't do this, it just includes another
: file as part of the current file.

: -Dick

: Dick Jackson djackson@ibd.nrc.ca Institute for Biodiagnostics
: Opinions are mine alone. National Research Council Canada, Winnipeg

I think that you are stuck here, as I understand it, executive commands
(those that start with a dot) can only be used at the command line or in
an include file at the top level, so even if the filename were a constant
you wouldn't be able to force compilation from within a routine or

program.

However if your code is a routine rather than a main program then you can
use CALL_PROCEDURE or CALL_FUNCTION to compile (if not already compiled)
and execute it (these are more efficient than EXECUTE).

```
--
 +-----------------------+--------------------------------- --+---------+
| James Tappin,          | School of Physics & Space Research |  O__    |
| sjt@star.sr.bham.ac.uk | University of Birmingham       | --  \/` |
| "If all else fails--read the instructions!"           |         |
 +--------------------------------------------------------- --+---------+
```

Subject: Re: HELP: Multiple-file Applications
Posted by djackson on Tue, 06 Dec 1994 16:05:45 GMT
View Forum Message <> Reply to Message

James Tappin writes:

> However if your code is a routine rather than a main program then you
> can use CALL_PROCEDURE or CALL_FUNCTION to compile (if not already
> compiled) and execute it (these are more efficient than EXECUTE).

Norbert Hahn writes:

```
>  I played a little with CALL_PROCEDURE and it seems to do what you like.
>  I wrote
>     sub = 'fader'
>   and typed
>     call_procedure, sub, out=0.25
```

[note: I'm not actually worried about variable procedure compilation,
 I know what I want to be compiled, but _when/if_ it's compiled
 is to be left to run-time. No harm done, it's the same
otherwise.]

> and noticed that fader.pro was found in the search path, compiled and
> executed.
>
> Note that IDL only compiles those procedure that haven't been compiled
> before in this session
>
> and
>
> that IDL only compiles the file including all procedure contained there
> *until* it has reached the END statement of the procedure requested.

>
> Thus, if you have nested procedures, it is best to put the outermost
> procedure (that's the one you call explicitly) at the end of the file.

This sounds good, and I didn't come to the same conclusion, since I
confused myself (and found another 'gotcha' here) by writing these test
routines, caller.pro and helper.pro: (bear with me!)

```
::::::::::::::
caller.pro
::::::::::::::
pro caller
  a = randomu(seed) + 1.0
  help, a, helper(a)
  pre_pre_helper, a
  help, pre_helper(a)
end
::::::::::::::
helper.pro
::::::::::::::
pro pre_pre_helper, x
  print, "In pre_pre_helper, X =", x
end

function pre_helper, x
  return, x+42.4242
end

function helper, x
  return, pre_helper(x)/2.02
end

function post_helper, x
  return, -x
end
```

Then, to test them in a fresh IDL session:

```
IDL> caller
% Compiled module: CALLER.
% Compiled module: HELPER.
A               FLOAT     =       1.23121
<Expression>    FLOAT     =       21.6116
In pre_pre_helper, X =      1.23121
```

---***--- OK, in CALLER, that one was compiled as a procedure call,
then the procedure was compiled.

```
% Variable is undefined: PRE_HELPER.

---***--- This is the 'gotcha': when CALLER was compiled, PRE_HELPER
looked
          like an array variable, since no function yet existed, I
suppose.

% Execution halted at CALLER <caller.pro(   7)> .
%       Called from $MAIN$ .
IDL> help,pre_helper(3)
<Expression>    FLOAT     =       45.4242
IDL> help,post_helper(3)
% Variable is undefined: POST_HELPER.
% Execution halted at CALLER <caller.pro(   7)> .
%       Called from $MAIN$ .
IDL>
```

So, having multiple routines in a 'subordinate' file, and calling the
last one found in there first, will cause all the others to work
thereafter, unless there are functions, in which case they'll look like
array variables. It's a bit constraining, but if I keep it strictly
modular, so only the last pro/function in the 'subordinate' file is
called from outside, then I'll be OK.

Thanks so far, any other tips? There must be lots of big widget-app
builders out there.

Cheers,
-Dick

Dick Jackson   djackson@ibd.nrc.ca   Institute for Biodiagnostics
Opinions are mine alone.   National Research Council Canada, Winnipeg

---

Subject: Re: HELP: Multiple-file Applications
Posted by hahn on Tue, 06 Dec 1994 21:00:51 GMT
View Forum Message <> Reply to Message

In article <3c04j4$pap@canopus.cc.umanitoba.ca> djackson@ibd.nrc.ca (Dick Jackson) writes:
> From: djackson@ibd.nrc.ca (Dick Jackson)
> Subject: Re: HELP: Multiple-file Applications
> Date: 5 Dec 1994 22:35:16 GMT

>> Problems:
>> - I don't see any way for an IDL routine to cause a file to be compiled
>>   (e.g. OK = EXECUTE(".RUN foo.pro") does not work)

> Just to clarify, I want the compilation to be done somewhere at
> run-time, say when a user selects a given menu item. Note that the '@'
> command to include a file doesn't do this, it just includes another
> file as part of the current file.

> -Dick

> Dick Jackson   djackson@ibd.nrc.ca   Institute for Biodiagnostics
> Opinions are mine alone.   National Research Council Canada, Winnipeg

I played a little with CALL_PROCEDURE and it seems to do what you like.
I wrote

```
   sub = 'fader'
```

and typed

```
   call_procedure, sub, out=0.25
```

and noticed that fader.pro was found in the search path, compiled and
executed.

Note that IDL only compiles those procedure that haven't been compiled
before in this session

and

that IDL only compiles the file including all procedure contained there
until it has reached the END statement of the procedure requested.

Thus, if you have nested procedures, it is best to put the outermost
procedure (that's the one you call explicitly) at the end of the file.

One exception to this: .run always compiles the entire file.

Hope this helps
Norbert Hahn

Subject: Re: HELP: Multiple-file Applications
Posted by steinhh on Wed, 07 Dec 1994 12:13:27 GMT
View Forum Message <> Reply to Message

In article <3c224p$c7h@canopus.cc.umanitoba.ca>, djackson@ibd.nrc.ca (Dick Jackson) writes:

|> % Variable is undefined: PRE_HELPER.
|>
|> ---***--- This is the 'gotcha': when CALLER was compiled, PRE_HELPER
|>   looked
|>   like an array variable, since no function yet existed, I

|> suppose.

[....]

|> So, having multiple routines in a 'subordinate' file, and calling the
|> last one found in there first, will cause all the others to work
|> thereafter, unless there are functions, in which case they'll look like
|> array variables. It's a bit constraining, but if I keep it strictly
|> modular, so only the last pro/function in the 'subordinate' file is
|> called from outside, then I'll be OK.
|>
|> Thanks so far, any other tips? There must be lots of big widget-app
|> builders out there.

The modular approach is a good choice, but of course it could be
quite a bit of work to split a very large file into such modules, if
the program isn't already well organized.

Personally, I always have widget programs looking much like the
this:
File: application.pro

;
Auxiliary event routines ; In order to have a "tidy" application_event routine

pro application_event,event

pro application,parameters

Note that all the "auxiliary event routines" are ONLY called from within
this file -- they have no use what so ever in other applications -- if
they do, I make them into a separate file (one for each multi-use routine).

You should note that when IDL compiles statements like "help,pre_helper(a)",
it looks through the path for any file called "pre_helper.pro", and examines
them for a potential declaration of the function pre_helper(). So, even if
you compile a program referring to a function that's not compiled, you
will avoid the problem you mentioned if it's placed in a file (in the path)
that has the name of the function.

A warning: This also spells trouble if your'e using a variable name
that by coincidence is identical to a function name.

Try e.g.:

```
IDL> vel = 0
IDL> vel(0) = 0  ; This works ok, but during compile-time, the variables
```

```
            ; aren't known that well, so:
IDL> delvar,vel
IDL> vel(0) = 1

vel(0) = 1
 ^
% Syntax error.
-------------------
```
The lesson is, of course: Don't oversimplify function names, you'l
want to save that for your variables.

Regards,

Stein Vidar

Subject: Re: HELP: Multiple-file Applications
Posted by Serdar Manizade on Wed, 07 Dec 1994 19:35:45 GMT
View Forum Message <> Reply to Message

> James Tappin writes:
..<stuff deleted>
> So, having multiple routines in a 'subordinate' file, and calling the
> last one found in there first, will cause all the others to work
> thereafter, ...
>
> Thanks so far, any other tips? There must be lots of big widget-app
> builders out there.

I may not be a big widget-app builder, but I have used the approach
you describe. In my case the routines had a natural organization,
where an initialization routine had to run before the other routines
could be used. I put the collection into a file, keeping the init
routine for the last routine in the file. The name of the file was
the name of the init routine with a .pro on the end. worked great.