Subject: Re: "include" a file
Posted by JDS on Fri, 26 Jun 2009 19:54:09 GMT
View Forum Message <> Reply to Message

On Jun 26, 3:50 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
> Various other languages have the option to include and evaluate the
> program contents of another file at runtime. IDL has the "@"
> operator, but that happens at compile time, so you need to know which
> file to include in advance. I find myself needing to drop small
> "parameter" files in individual directories for a routine to process
> as it crawls through. I could certainly prepare an IDL .sav file, or
> some other data format, parse that, and set-up structures and
> variables as needed, but that makes editing and updating the file very
> painful. What IDL needs is a way to "include" a file directly, and
> evaluate its contents. Finding nothing, I came up with the following
> concept:
>
> ;; include -- Include and evaluate the IDL command contents of a file.
> ;; To use, give the variable "include_file" in the same scope the name
> ;; of a valid file containing IDL commands (batch syntax only), then
> ;; batch include this file, ala:
> ;;
> ;; include_file='/path/to/file'
> ;; @include
> ;;
> _inc_lines=replicate('',file_lines(include_file))
> openr,_inc_un,include_file,/get_lun
> readf,_inc_un,_inc_lines
> _inc_wh=where(~stregex(_inc_lines,'\$(;.*)?[\t]*$',/
> BOOLEAN),_inc_cnt)
> _inc_start=0L
> for _inc_i=0L,_inc_cnt-1 do begin
> _inc_parts=_inc_lines[_inc_wh[_inc_i]]
> if _inc_wh[_inc_i] gt _inc_start then $
> _inc_parts=strjoin($
> reform((stregex(_inc_lines[_inc_start:_inc_wh
> [_inc_i]-1],$
> ' *(.*) *\$(;.*)?[\t]*$', $
> /SUBEXPR,/EXTRACT))[1,*])) +
> _inc_parts
> _inc_void=execute(_inc_parts)
> _inc_start=_inc_wh[_inc_i]+1L
> endfor
> free_lun,temporary($; Clean-up all variables
> (_inc_parts=temporary($
> (_inc_wh=temporary($
> (_inc_lines=temporary($
> (_inc_void=temporary($

> (_inc_cnt=temporary($
> (_inc_i=temporary($
> (_inc_start=temporary(_inc_un))))))))))))))))
>
> This works just fine, collapsing multi-line commands, and executing
> them, at the cost of temporarily polluting the current scope with
> "_inc_" variables (these are left undefined after the @include). You
> have to use "batch syntax", aka as "standalone single line command"
> syntax, but for my purposes this isn't a major limitation. It uses
> execute, so won't work in the IDL_VM, and if you try to do it many
> times in a loop, you might regret it. But for quickly setting up
> human-editable parameter lists, I find it works great.
>
> Do others encounter this problem, and has anyone solved it in a
> different way?

OK, that got poorly formatted by the news relay:

http://tir.astro.utoledo.edu/idl/include.pro

Subject: Re: "include" a file
Posted by Michael Galloy on Fri, 26 Jun 2009 20:23:45 GMT
View Forum Message <> Reply to Message

JDS wrote:
> On Jun 26, 3:50 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
>> Various other languages have the option to include and evaluate the
>> program contents of another file at runtime. IDL has the "@"
>> operator, but that happens at compile time, so you need to know which
>> file to include in advance. I find myself needing to drop small
>> "parameter" files in individual directories for a routine to process
>> as it crawls through. I could certainly prepare an IDL .sav file, or
>> some other data format, parse that, and set-up structures and
>> variables as needed, but that makes editing and updating the file very
>> painful. What IDL needs is a way to "include" a file directly, and
>> evaluate its contents. Finding nothing, I came up with the following
>> concept:
>>
>> ;; include -- Include and evaluate the IDL command contents of a file.
>> ;; To use, give the variable "include_file" in the same scope the name
>> ;; of a valid file containing IDL commands (batch syntax only), then
>> ;; batch include this file, ala:
>> ;;
>> ;; include_file='/path/to/file'
>> ;; @include
>> ;;
>> _inc_lines=replicate('',file_lines(include_file))

```
>>  openr,_inc_un,include_file,/get_lun
>>  readf,_inc_un,_inc_lines
>>  _inc_wh=where(~stregex(_inc_lines,'\$(;.*)?[ \t]*$',/
>>  BOOLEAN),_inc_cnt)
>>  _inc_start=0L
>>  for _inc_i=0L,_inc_cnt-1 do begin
>>     _inc_parts=_inc_lines[_inc_wh[_inc_i]]
>>     if _inc_wh[_inc_i] gt _inc_start then $
>>        _inc_parts=strjoin( $
>>                 reform((stregex(_inc_lines[_inc_start:_inc_wh
>>  [_inc_i]-1],$
>>                                ' *(.*) *\$(;.*)?[ \t]*$', $
>>                                /SUBEXPR,/EXTRACT))[1,*])) +
>>  _inc_parts
>>     _inc_void=execute(_inc_parts)
>>     _inc_start=_inc_wh[_inc_i]+1L
>>  endfor
>>  free_lun,temporary( $          ; Clean-up all variables
>>          (_inc_parts=temporary( $
>>          (_inc_wh=temporary( $
>>          (_inc_lines=temporary( $
>>          (_inc_void=temporary( $
>>          (_inc_cnt=temporary( $
>>          (_inc_i=temporary( $
>>          (_inc_start=temporary(_inc_un))))))))))))))))
>>
>>  This works just fine, collapsing multi-line commands, and executing
>>  them, at the cost of temporarily polluting the current scope with
>>  "_inc_" variables (these are left undefined after the @include).  You
>>  have to use "batch syntax", aka as "standalone single line command"
>>  syntax, but for my purposes this isn't a major limitation.  It uses
>>  execute, so won't work in the IDL_VM, and if you try to do it many
>>  times in a loop, you might regret it.  But for quickly setting up
>>  human-editable parameter lists, I find it works great.
>>
>>  Do others encounter this problem, and has anyone solved it in a
>>  different way?
>
>  OK, that got poorly formatted by the news relay:
>
>  http://tir.astro.utoledo.edu/idl/include.pro
```

Cool. I'm playing around writing this as a regular routine and exporting variables back using SCOPE_VARFETCH (code below). Then the include can be done as:

```
  IDL> mg_include, 'my_batchfile'
```

Mike
--
www.michaelgalloy.com
Associate Research Scientist
Tech-X Corporation

```
;+
; Includes the contents of the given batch file at the calling level. The
; call::
;
;    IDL> mg_include, 'test'
;
; is equivalent to::
;
;    IDL> @test
;
; except that the filename is specified as a string variable instead of
; required to be known at compilation time.
;
; :Params:
;    _mg_include_filename : in, required, type=string
;       filename to include
;-
pro mg_include, _mg_include_filename
  compile_opt strictarr
  on_error, 2

  _mg_include_nlines = file_lines(_mg_include_filename + '.pro')
  _mg_include_lines = strarr(_mg_include_nlines)
  openr, _mg_include_lun, _mg_include_filename + '.pro', /get_lun
  readf, _mg_include_lun, _mg_include_lines
  free_lun, _mg_include_lun

  for _mg_include_i = 0L, _mg_include_nlines - 1L do begin
    _mg_include_result = execute(_mg_include_lines[_mg_include_i])
  endfor

  _mg_include_names = scope_varname(count=_mg_include_count)

  for _mg_include_i = 0L, _mg_include_count - 1L do begin
    if (strmid(_mg_include_names[_mg_include_i], 0, 12) ne
'_MG_INCLUDE_') then begin
      (scope_varfetch(_mg_include_names[_mg_include_i], level=-1,
/enter)) $
        = scope_varfetch(_mg_include_names[_mg_include_i])
    endif
  endfor
```

end

Subject: Re: "include" a file
Posted by David Fanning on Fri, 26 Jun 2009 20:42:25 GMT
View Forum Message <> Reply to Message

Mike Galloy writes:

> Cool. I'm playing around writing this as a regular routine and exporting
> variables back using SCOPE_VARFETCH (code below). Then the include can
> be done as:
>
> IDL> mg_include, 'my_batchfile'

I don't know. I'm thinking about retiring again. :-(

Cheers,

David

P.S. Let's just say that anything that makes my head hurt
to read is coming in second these days to tennis.

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: "include" a file
Posted by Heinz Stege on Sat, 27 Jun 2009 02:37:06 GMT
View Forum Message <> Reply to Message

On Fri, 26 Jun 2009 12:50:56 -0700 (PDT), JD Smith wrote:

...

> This works just fine, collapsing multi-line commands, and executing
> them, at the cost of temporarily polluting the current scope with
> "_inc_" variables (these are left undefined after the @include). You

...

Here is another idea to avoid the "_inc_" variables within the current
scope. (Therefore you have to accept the existence of a common

block.) I changed JDs code a little bit and put it into the following
functions:

```
function include_command
;----------------------
;
common  include_function,status,_inc_lines,_inc_wh,_inc_start,_inc_i
;
_inc_parts=_inc_lines[_inc_wh[_inc_i]]
if _inc_wh[_inc_i] gt _inc_start then $
   _inc_parts=strjoin( $

reform((stregex(_inc_lines[_inc_start:_inc_wh[_inc_i]-1],$
                          ' *(.*) *\$(;.*)?[ \t]*$', $
                          /SUBEXPR,/EXTRACT))[1,*])) + _inc_parts
_inc_start=_inc_wh[_inc_i]+1L
_inc_i++
;
return,_inc_parts
end


function include_status,include_file
;-----------------------------------
;
common  include_function,status,_inc_lines,_inc_wh,_inc_start,_inc_i
;
if n_elements(status) le 0 then status=0
;
if status eq 0 then begin
   _inc_lines=replicate('',file_lines(include_file))
   openr,_inc_un,include_file,/get_lun
   readf,_inc_un,_inc_lines
   free_lun,_inc_un

 _inc_wh=where(~stregex(_inc_lines,'\$(;.*)?[\t]*$',/BOOLEAN) ,_inc_cnt)
   _inc_start=0L
   _inc_i=0L
   status=1
   end
;
if _inc_i ge n_elements(_inc_wh) then begin
   ; Clean-up the arrays in the common block
   temp=temporary(_inc_lines)
   temp=temporary(_inc_wh)
   status=0
   end
```

;
return,status
end

After that you can use the following command within your program or
command line:

while include_status('commands.pro') do $
  if not execute(include_command()) then message,/cont,'Error.'

Here "commands.pro" is the include file with the commands to execute.
There are no unwanted variables anymore.

If you prefer an easier command than the long while-command, you can
do the file reading part of the include_status function within another
procedure and use something like

include_file,'commands.pro'
@include_execute

With the modified while-command written into the file
"include_execute.pro".

Heinz

Subject: Re: "include" a file
Posted by Craig Markwardt on Sun, 28 Jun 2009 20:12:58 GMT
View Forum Message <> Reply to Message

On Jun 26, 3:50 pm, JD Smith <jdtsmith.nos...@yahoo.com> wrote:
> Various other languages have the option to include and evaluate the
> program contents of another file at runtime. IDL has the "@"
> operator, but that happens at compile time, so you need to know which
> file to include in advance. I find myself needing to drop small
> "parameter" files in individual directories for a routine to process
> as it crawls through. I could certainly prepare an IDL .sav file, or
> some other data format, parse that, and set-up structures and
> variables as needed, but that makes editing and updating the file very
> painful. What IDL needs is a way to "include" a file directly, and
> evaluate its contents. Finding nothing, I came up with the following
> concept:
>
...
> Do others encounter this problem, and has anyone solved it in a
> different way?

I have encountered this style of problem before, and I have solved it your way as well. To be honest, I've pretty much always regretting doing this because I let the "configuration" code get out of hand, setting extra variables, running little bits of code, etc. I'd want to call the config code in two different places for two different purposes, and then it just started getting too complicated.

These days, my preferred way is to turn the "configuration" file into a real IDL function (or procedure). Then I can compile that and interogate it directly.

```
  ;; Compile my configuration file
  my_config_file = '/data/run1/config1.pro'
  file_compile, my_config_file, pro_name=mypro

  ;; Interogate my configuration function, in this case retrieve a
struct
  config_struct = call_function(mypro)

  ;; Use config_struct accordingly...
```

It takes very little extra effort to slap a function declaration at the top of one's file to make something like that work, and it hands the parsing wizardry to IDL where it belongs.

Here I'm just returning a structure from my configuration function, but the possibilities are not really limited to that. The procedure could be used to do actual work rather than just to return a static structure. Or in the case where I wanted to call my config function twice, once for initialization, and once for clean-up, it might be done something like this,

```
  call_procedure, mypro, 'INIT'
  ... do whatever ...
  call_procedure, mypro, 'CLEANUP'
```

The point is to apply a little discipline so things don't get too out of hand.

Craig

FILE_COMPILE is here: http://cow.physics.wisc.edu/~craigm/idl/misc.html