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
```