Subject: Re: Which like command for IDL?
Posted by Vapuser on Fri, 05 Jan 2001 21:42:04 GMT
View Forum Message <> Reply to Message

Sorry for superseding my post, but I had left a rant in about
routine_info/resolve_routine that I really didn't want to send,
since I'd discovered some information about those two routines that
made my rant a bit too splenetic, if you know what I mean.

Anyway, I do have some problems with those two routines which I'll
indicate below.

davidf@dfanning.com (David Fanning) writes:

> David Fanning (davidf@dfanning.com) writes:
>
>> It is a moot point anyway, in this case, since the program
>> uses some of the neat new SWITCH, BREAK, etc. stuff that
>> comes in IDL 5.4, and will not compile in earlier versions.
>
> Interestingly, the FILE_WHICH program supplied in IDL 5.4
> calls a built-in, but undocumented, program STRTOK, which
> appears to separate the path subdirectories based on
> a delimiter supplied to the function. I'll leave it
> to the expert sleuths in the group to tell us what it
> *really* does. :-)
>

<snip>

If it's like the C function of the same name, it 'tokenizes' the
string using any delimiter which appears in a particular set, which
is input to the function. It's like repeated calls to strsplit with
different delimiters.

So, *IIRC* you could say 'stuff=strtok(path,':/\') and it would
split the string up regardless of whether you were on a Windows of
Unix machine. (I forget what the delimiter is for Vaxen)

By the way, here's my entry into the (pre 5.4) field. It works by
trying it as a system routine first, then it looks in the output
from help,/source for an *exact* match of the input name (stopping
at the first, see my <rant> below), then an object (if it has a ::
in it) then procedure, a function and, if all these fail, it appends
a '__define' on the input name and tries that, just in case someone
just passed the name of the object it.

It will even work if the object method is defined in it's own file,
provided one follows the obj__method.pro naming convention.

It has a *whole* slew (well, two actually) of GOTOs which I couldn't find a way to get rid of, mostly because resolve_routine/routine_info need to know whether the thing being resolved/asked-about is a procedure or a function beforhand.

<rant>

After I rewrote this routine to be a bit smarter I came to a better understanding of the problems associated with resolve_routine/routine_info. But I still think that the proper way to do this sort of thing is to ere on the side of accomodating the user and let them resolve necessary ambiguities rather then requiring them to do it *before* the call. (of course, in order to follow my own advice, I'll have to rewrite my `which.pro', which I am going to do in my copious free time!) If the user askes for information about two routines with the same name, one a function and one a procedure, I think routine_info should return information about *both* along with some way to tell which is which and let the user decide which he/she wants. Similarly, I wonder why routine_info doesn't resolve the routine(s) itself, instead of requiring it be done by the user before hand. If there is ambiguity, *resolve both* and default to the previous lemma.

If anyone can tell me why this wouldn't be a better way to do it, please do so but I don't see any *real* reason to do it except that it's harder to write the code. (and that's only a quasi-real reason ;->)

</rant>

William Daffer

```
;+
; NAME:  Which
; $Id: which.pro,v 1.2 2001/01/05 21:03:04 vapuser Exp $
; PURPOSE:  Like the Unix 'which' program. Tells you which source file
;          a given routine is in.
;
; AUTHOR:  William Daffer
;
; CATEGORY:  Utility
;
; CALLING SEQUENCE: which,'routine'
;
; INPUTS:  routine:  An IDL procedure/function
;
; OPTIONAL INPUTS:  None
```

```
;
; KEYWORD PARAMETERS:  None
;
; OUTPUTS:  Prints one line with the following info
;
;  "routine: System routine" if it's a system routine. -- or --
;  "routine: path" if it finds the routine -- or --
;  "routine: Doesn't exist" if the previous two fail.
;
;
; OPTIONAL OUTPUTS:  none
;
; COMMON BLOCKS:  none
;
; SIDE EFFECTS: The routine is compiled along with any possible
;               routines contained in the object definition, if this
;               circumstance applies.
;
; RESTRICTIONS:
;
; PROCEDURE: Look in the system routines for this name, if not there,
;            look in the output from help,/source, if it isn't there,
;            try various calls to resolve_routine and routine_info.
;            If `routine' has a '::' in it (e.g. foo::bar), `which'
;            will resolve will be foo__define and see if bar is a
;            method defined in that file, otherwise it will assume
;            that the routine is defined in the file `foo__bar.'
;
;            If these no '::' and `routine' doesn't resolve either as
;            a procedure or a function, `which' will attempt to
;            revolve 'routine__define' and see if someone just passed
;            an object name in.
;
;
;
; EXAMPLE:
;
; IDL> which,'foo'
;           foo: /path/to/foo.pro
;
; IDL> which,'foo::init'
;       foo::init: /path/to/foo__define.pro
;
;    if init is defined in foo__define.pro
;
;       -- or --
;
; IDL> which,'foo::init'
;       foo::init: /path/to/foo__init.pro
;
;       if init is defined in foo__init.pro
;
; IDL> which,'contour'
;      contour: SYSTEM ROUTINE!
;
; IDL> which,'foobar'
;      foobar: DOESN'T EXIST!
;
; MODIFICATION HISTORY:
;
; $Log: which.pro,v $
; Revision 1.2  2001/01/05 21:03:04  vapuser
; Reworked completely
;
; Revision 1.1  1999/10/06 21:54:32  vapuser
; Initial revision
;
;
;Copyright (c) 1999, William Daffer
;-

PRO which, procname
  usg = "Usage: which,`procname' (with `procname' a nonempty STRING)"
  IF n_params() LT 1 OR n_elements(procname) EQ 0 THEN BEGIN
    Message,USG,/cont
    return
  ENDIF

  IF size(procname,/type) NE 7 THEN BEGIN
    Message,usg,/cont
    return
  ENDIF

  tproc =  strupcase(strtrim( procname,2))

  IF strlen(tproc) EQ 0 THEN BEGIN
    Message,usg,/cont
    return
  ENDIF
  savequiet = !quiet
  !quiet = 1
  system_routines = routine_info(/system)

  catch,/cancel
  errcnt = -1
  is_func = 0
```

```
is_obj = 0

  ;; Look in the SYSTEM routines first
pos = strpos( system_routines, tproc)
x = where(pos NE -1,nx)
IF nx NE 0 THEN BEGIN
  found = 0
  ii = 0
  REPEAT BEGIN
  ;; check for possible false positives!
    tmp = strcompress(system_routines[x[ii]])
    tmp = strsplit(tmp,' ',/extract)
    test = tmp[0]
    IF test EQ  tproc THEN found = 1
    ii = ii+1
  ENDREP UNTIL found OR ii GE nx
  IF found THEN BEGIN
    outmsg = procname + ': SYSTEM ROUTINE!'
    print,outmsg
    !quiet = savequiet
    return
  ENDIF
ENDIF

;; Then in the already compiled routines

help,/source,out=out
out = strupcase(out)
pos = strpos(out,tproc)
x = where(pos NE -1, nx )
found = 0
ii = 0

IF nx NE 0 THEN BEGIN
  REPEAT BEGIN
  ;; check for false positives!
    tmp = strcompress(out[x[ii]])
    tmp = strsplit(tmp,' ',/extract)
    test = tmp[0]
    IF test EQ  tproc THEN found = 1
    ii = ii+1
  ENDREP UNTIL found OR ii GE nx
  IF found THEN BEGIN
    catch, error
    IF error NE 0 THEN BEGIN
      catch,/cancel
      is_func = 1
    ENDIF
    info = routine_info(tproc,/source,FUNC=is_func)
    outmsg = info.path
  ENDIF
ENDIF

  ;; And finally, try to compile it!

errcnt = -1
is_func = 0
is_obj = 0

IF NOT found THEN BEGIN

  IF strpos(procname,'::') NE -1 THEN BEGIN

      ;; Damn! object reference!

    tmp = strsplit(tproc,':',/extract)
    procs_to_resolve = [tmp[0] + "__DEFINE", procname]
    message,/reset

    errcnt2 = -1
    is_func2 = 0

    catch, error1
    IF error1 NE 0 THEN BEGIN
      errcnt2 = errcnt2 + 1
      CASE errcnt2 OF
        0: BEGIN
          is_func2 = 1
          message,/reset
        END
        1: GOTO, own_file
      ENDCASE
    ENDIF
    IF errcnt2 LT 0 THEN $
      resolve_routine,procs_to_resolve[0] ; the __define routine, always a proc

    info = routine_info(procname,/source,func=is_func2)

      ;; If we've made it this far, it's defined in the
      ;; tmp[0]__define file, so go to the end

    outmsg = info.path
    GOTO, endit

    OWN_FILE:
```

```
    errcnt2 = -1
    is_func2 = 0
    catch,error2
    IF error2 NE 0 THEN BEGIN
      error2 = 0
      errcnt2 = errcnt2 + 1
      CASE errcnt2 OF
        0: BEGIN
          is_func2 = 1
          message,/reset
        END
        1: BEGIN
          print, procname + ": DOESN'T EXIST!"
          return
        END
      ENDCASE
    ENDIF
    resolve_routine,procs_to_resolve[1],is_func=is_func2 ;
    info = routine_info(procname,/source,func=is_func2)
    outmsg = info.path

  ENDIF ELSE BEGIN

    ;; Doesn't have a "::" in it. May still be an object name, though!
    catch, error
    IF error NE 0 THEN BEGIN
      errcnt = errcnt+1
      CASE errcnt OF
        0: BEGIN
          ; won't compile as a procedure,
          ; try as funtion
          is_func = 1
          message,/reset
        END
        1: BEGIN
          is_obj = 1
          is_func = 0
          tproc  =  tproc + "__DEFINE"
          message,/reset
          ;resolve_routine, tproc[jj]
        END
        ELSE : BEGIN
          ;; can't resolve it as either procedure
          ;; function or object.
          ;; Must not exist!
          !quiet = savequiet
          print, procname + ": DOESN'T EXIST!"
          return
        END
      ENDCASE
    ENDIF
    resolve_routine, tproc, is_func= is_func

    info = routine_info(tproc,/source,FUNC=is_func)
    IF !error_state.code NE 0 THEN BEGIN
      !quiet = savequiet
      outmsg = procname + ": DOESN'T EXIST!"
      print, outmsg
      return
    ENDIF
    outmsg = info.path
  ENDELSE
ENDIF

ENDIT:
outmsg =  procname + ': ' + outmsg
print, outmsg
!quiet = savequiet
return

END
```

--
William Daffer: 818-354-0161: William.Daffer@jpl.nasa.gov