Subject: BP: convenience function for setting breakpoints
Posted by tom.grydeland on Thu, 28 Feb 2013 13:32:43 GMT
View Forum Message <> Reply to Message

I don't use the workbench or emacs, so setting breakpoints is quite a hassle: Finding the exact path to the file, and line numbers, making sure the routine is compiled etc. For inherited methods, one also needs to find out in what superclass a method is defined. All of this means that simply adding 'stop' to the line is so much faster that I hardly ever used breakpoints.

Enter the convenience function BP: All you need is a name. The routine will be compiled for you, the path is determined automatically, and the breakpoint is set to the line with the routine declaration (i.e. before the first statement of the routine). If you have the METHOD_NAMES and SUPERCLASSES functions I posted previously, you can also give 'CLASS::METHOD' as the name. Even if METHOD is inherited, the correct definition will be found and breakpoint set accordingly.

The routine uses the /EITHER and /IS_FUNCTION keywords of ROUTINE_INFO and friends, except that EITHER defaults to true when IS_FUNCTION is not set. If you have a function and a procedure by the same name in the same file, you can use these keywords to control which one gets the breakpoint. Otherwise, the breakpoint will be set on the topmost one.

If you find this useful, drop me a line.

--T

```
; docformat = 'rst'

;+
; :Copyright:
;    (c) 2013-2013, Northern Research Institute Tromsø AS (Norut).
;                   All rights reserved
;
; :Author: Tom Grydeland <tom.grydeland@norut.no>
;
; :Categories: debugging
;-

;+
; Set a breakpoint without having to know routine path or line numbers.
;
; Any keywords given in addition to the keywords listed below will be passed on
; to BREAKPOINT.
;
; This is a convenience routine, mostly useful on the command line.
;
; :Params:
;   routine: required, in, type=string
;     The name of the routine (procedure or function) in which to break.
;     Can also be the name of a method, on the format 'classname::methodname' in
;     which case the method will be found in the definition of the given class
;     or one of its superclasses (in inheritance order), and a breakpoint set
;     in whatever file the method is defined.
;   line: optional, in, type=integer
;     The line to set breakpoint at. Default is first line of routine
;
; :Keywords:
;   is_function: optional, in, type=boolean
;     If set, looks for a function (and not a procedure)
;   either: optional, in, type=boolean
;     If set, looks for either function or procedure.  Unless `IS_FUNCTION` is
;     set, the default is true.
;
; :Examples:
;   Set a breakpoint on the first line of the routine ALL::
;     IDL> bp, 'all'
;   Set a breakpoint for the _function_ (and not procedure) WHICH::
;     IDL> bp, 'which', /IS_FUNCTION
;   Set a breakpoint for the _procedure_ (and not function) WHICH::
;     IDL> bp, 'which', EITHER=0
;
; :See also:
;   BREAKPOINT
;-
pro bp, routine, line, either=either, is_function=is_function, _extra=extra
  compile_opt idl2, logical_predicate
  on_error, 2

  if n_elements(either) eq 0 && ~keyword_set(is_function) then either=1

  if strpos(routine, '::') eq -1 then begin
    resolve_routine, routine, either=either, is_function=is_function, /no_recompile
  endif else begin
    ;; method
    parts = strsplit(strupcase(routine), '::', /extract)
    methods = method_names(parts[0], whence)
    routine = whence[parts[1]] + '::' + parts[1]
  endelse

  path = routine_filepath(routine, either=either, is_function=is_function)

  if n_elements(line) eq 0 then begin
    ; lines = trimsource(readtextlinesfromfile(path))
    lines = readtextlinesfromfile(path)
    ;; apologies for the line noise, IDL regexes don't have word end anchors
    if keyword_set(either) then profun = '(pro|function)'       $
    else if keyword_set(is_function) then profun='function'     $
    else profun = 'pro'
    pattern = '(^|[^[:alnum:]$_])' + profun + '[[:space:]]+' + routine + '($|[^[:alnum:]_$])'

    line = (where(stregex(lines, pattern, /fold_case, /boolean)))[0]

    ;; the default line number is the _last_ of the function declaration lines,
    ;; so skip continuation lines in the declaration
    ;; (and remember that lines are counted from 1 while indices count from 0)
    while 1 do begin
      ll = lines[line++]
      if strpos(ll, ';') ne -1 then ll = (strsplit(ll, ';', /extract))[0]
      ll = strtrim(ll, 2)
      if stregex(ll, '[^[:alnum:]_$][$]$', /bool) then continue
      break
    endwhile
  endif

  breakpoint, path, line, _extra=extra
  print, string(format='("setting breakpoint on line ", i0, " of file ", a)', $
                line, path)
  return
end
```