```
;; This is a collection of tests from the Debugger.NoteFile.  It tests the break package and IL
error system.  The individual test files for each of the functions have been appended together in
this big file to share common code and gain diagnostic information by testing the functions in a
particular order.
;;
;; The source for the text file listing is the NoteCards database at
{Erinyes}<Test>Lisp>Lyric>Plans>Debugger.NoteFile.  Changes are made only to the NoteFile.  The
listings are
;; Filed As:  {eris}<lispcore>test>Debugger>Debugger.u and
{eris}<lispcore>test>Debugger>BreakWindow.u


(do-test "setup user interaction window"
      (unless (fboundp 'do-test-menu-setup)
            (load "{Eris}<test>tools>do-test-menu.dfasl" t))
      (setq dtmw (do-test-menu-setup "Debugger")))
(do-test-group
    ("il:break" :before (progn (unbreak il:ourfn)
                       (il:defineq (il:ourfn nil nil))) :after (unbreak il:ourfn))
    (do-test "il:break"
          (and (il:break il:ourfn)
              (il:memb 'il:ourfn il:brokenfns)
              (unbreak il:ourfn)))
    (do-test "il:break of il:broken fns update of il:brokenfns" (unbreak il:ourfn)
          (and (il:break il:ourfn)
              (il:break il:ourfn)
              (il:memb 'il:ourfn il:brokenfns)
              (unbreak il:ourfn)))
    (do-test "il:break of il:advisedfns updating il:brokenfns"
          (and (il:defineq (il:ourfn nil nil))
              (not (il:ourfn))
              (il:advise 'il:ourfn 'il:around nil t)
              (il:break il:ourfn)
              (il:memb 'il:ourfn il:brokenfns)
              (car (unbreak il:ourfn))))
    (do-test "AR 7618 BREAK :IN broken"
        (il:defineq (foo nil (print "foo"))
               (bar nil (foo) (print "bar")))
        (il:break (foo :in bar))
        (prog1
            (do-test-menu-message dtmw 'low
                (concatenate 'string "In the exec, enter (foo).

                "
                           "You should see a break window."))
            (unbreak (foo :in bar)))))
(do-test-group
    ("xcl:break-function" :before (progn (unbreak il:ourfn)
                               (il:defineq (il:ourfn nil nil))) :after
     (unbreak il:ourfn))
    (do-test "simple il:break-function" (unbreak)
          (and (break-function 'il:ourfn)
              (il:memb 'il:ourfn il:brokenfns)
              (unbreak il:ourfn)))
    (do-test "il:break-function :trace"
        (and (break-function 'floor :trace t)
            (floor 20 3)
            (and (do-test-menu-message dtmw 'low
                    (concatenate 'string
                        "You should see a *Trace-Output* window
                        "
                        "with a call to floor passed a 20 and a 3
                        "
                        "and returning a 6 and a 2."))
                (eq 'our-fn (car (untrace 'our-fn))))))
    (do-test "il:break-function :when nil (AR 8162)" (break-function 'floor :when nil)
        (prog1
            (do-test-menu-message dtmw 'low
                (concatenate 'string
                          "Enter

                          (floor 20 3)

                          to the exec.

                          "
                          "You should see the result

                          " "6
                          2
```

```
                        "
                        " and you should NOT see a break window
                        "))
        (untrace 'floor)))
(do-test "il:break-function :when t" (break-function 'floor :when t)
    (prog1
        (do-test-menu-message dtmw 'low
            (concatenate 'string
                        "Enter

                        (floor 20 3)

                        to the exec.

                        "
                        "You should see a break window "
                        "for Breakpoint at flo0r.

                        "
                        "Type OK to the break window."
                        "  The break window should go away and "
                        "the exec should show"
                        " floor returning a 6 and a 2."))
        (unbreak 'floor)))
(do-test "il:break-function :trace nil" (break-function 'floor :trace nil)
    (prog1
        (do-test-menu-message dtmw 'low
            (concatenate 'string
                        "Enter

                        (floor 20 3)

                        to the exec.

                        "
                        "You should see a break window "
                        "for Breakpoint at floor.

                        "
                        "Type OK to the break window."
                        "  The break window should go away and "
                        "the exec should show"
                        " floor returning a 6 and a 2."))
        (unbreak 'floor)))
(do-test "il:break-function :trace :when nil"
    (and (break-function 'floor :trace t :when nil)
        (prog1
            (do-test-menu-message dtmw 'low
                (concatenate 'string
                    "Enter

                    (floor 20 3)

                    to the exec.

                    "
                    "You should see the result

                    " "6
                    2

                    "
                    "You should not see a new call to floor"
                    " in the *Trace-Output* window
                    "))
            (untrace 'our-fn))))
(do-test "il:break-function :when exp (AR 8162)"
    (break-function 'floor :when (when nil t))
    (prog1
        (do-test-menu-message dtmw 'low
            (concatenate 'string
                        "Enter

                        (floor 20 3)

                        to the exec.

                        "
                        "You should see the result
```

```
                            " "6
                            2

                            "
                            " and you should NOT see a break window
                            "))
              (untrace 'floor))))
(do-test-group ("HELPFLAG" :after (progn (setq il:helpflag t)
                                         (setq il:helpdepth 7)))
    (do-test "*test-mode* :interactive switch on" (setq save.test.mode *test-mode*)
          (setq *test-mode* :interactive))
    (do-test "IL:HELPFLAG BREAK!" (setq il:helpflag 'break!)
        (proceed-case
            (error
                "Break test.  This is a test, just a test.
                Select Proceed from middle button menu.")
            (proceed nil :report "Select me!" t)))
    (do-test "IL:HELPFLAG NIL" (setq il:helpflag nil)
        (prog1
            (not  (ignore-errors
                    (proceed-case
                        (error
                            "Break test.  This is a test, just a test.
                            Select Proceed from middle button menu.")
                        (proceed nil :report "Select me!" nil))))
            (setq il:helpflag t)))
    (do-test "restore *test-mode*" (setq *test-mode* save.test.mode))
    (do-test "IL:HELPFLAG IL:SETTOPVAL (AR 7845)" (il:settopval il:helpflag nil)
          (and (not (il:gettopval il:helpflag))
              (il:settopval il:helpflag t)
              (eq t (il:gettopval il:helpflag)))))
(do-test-group ("IL:NLSETQ" :before (setq il:helpflag nil) :after (setq il:helpflag t))
    (do-test "IL:NLSETQ error" (not (il:nlsetq (error "just an error"))))
    (do-test "IL:NLSETQ signal error" (not (il:nlsetq (signal 'error))))
    (do-test "AR 7252 IL:NLSETQ SERIOUS-CONDITION"
        ;;nlsetq should not trap serious-conditions
        (expect-errors (serious-condition) (il:nlsetq (signal 'serious-condition)))))
(do-test-group
    ("unbreak" :before (defun our-fn (x) (values x (not x)))
     (defun super-fn nil (our-fn nil) t))
    (do-test "simple unbreak"
          (and (il:break our-fn) (unbreak) (not (member 'our-fn il:brokenfns))
              (not (unbreak))))
    (do-test "unbreak of (sub-fn il:in superfn)" (unbreak super-fn)
          (and (il:break (our-fn il:in super-fn))
              (unbreak (our-fn il:in super-fn))))
    (do-test "unbreak of (sub-fn in superfn)" (unbreak super-fn)
          (and (il:break (our-fn in super-fn))
              (unbreak (our-fn in super-fn))))
    ;;the following commented out due to  ignore-errors causing
    ;;do-test-file to abort.:  (do-test "unbreak of (sub-fn in
    ;;superfn)" (unbreak super-fn) (and (il:break (our-fn in
    ;;super-fn)) (unbreak (our-fn in super-fn)))) (do-test
    ;;"unbreak of '(sub-fn in superfn)" (unbreak super-fn) (and
    ;;(il:break (our-fn in super-fn)) (unbreak '(our-fn in
    ;;super-fn))))
    (do-test "unbreak of '(sub-fn il:in superfn)" (unbreak super-fn)
          (and (il:break (our-fn il:in super-fn))
              (unbreak '(our-fn il:in super-fn))))
    ;;(do-test "unbreak of '(sub-fn :in superfn)" (unbreak
    ;;super-fn) (and (il:break (our-fn :in super-fn)) (unbreak
    ;;'(our-fn :in super-fn)))) (do-test "unbreak0 of '(sub-fn il:in
    ;;superfn)" (unbreak super-fn) (and (il:break (our-fn il:in
    ;;super-fn)) (il:unbreak0 '(our-fn il:in super-fn))))
)
(do-test-group ("il:rebreak" :before (defun our-fn (x) (values x (not x)))
           (defun super-fn nil (our-fn nil) t))
           (do-test "simple il:rebreak"
                 (and (il:break our-fn)
                     (unbreak our-fn)
                     (il:rebreak our-fn))))
(do-test-group ("untrace" :before (defun our-fn (x) (values x (not x))) (untrace))
           (do-test "simple untrace"
                 (and (trace our-fn) (untrace) (not (untrace))
                     (not (member 'our-fn il:brokenfns))))
           (do-test "(untrace) with broken fns"
                 (and (il:break our-fn)
                     (not (untrace))
```

```
                    (member 'our-fn il:brokenfns)))
          (do-test "(untrace (sub-fn in super-fn))"
                (and (defun our-fn (x) (values x (not x)))
                    (defun super-fn nil (our-fn nil) t)
                    (trace (our-fn il:in super-fn))
                    (untrace (our-fn il:in super-fn)))))
(do-test-group ("trace" :before (defun our-fn (x) (values x (not x))))
    (do-test "trace il:brokenfns check" (trace our-fn)
          (prog1 (member 'our-fn il:brokenfns)
                (untrace 'our-fn)))
    (do-test "il:broken prop check" (trace our-fn)
          (prog1 (get 'our-fn 'il:broken)
                (untrace 'our-fn)))
    (do-test "simple interpreted trace" (trace our-fn)
        (our-fn t)
        (and (do-test-menu-message dtmw 'low
                (concatenate 'string "Do you see a *Trace-Output* window
                "
                          "with a call to our-fn passed a t
                          "
                          "and returning a t nil?"))
             (eq 'our-fn (car (untrace 'our-fn)))))
    (do-test "simple compiled trace" (trace floor)
        (floor 20 3)
        (prog1
            (do-test-menu-message dtmw 'low
                (concatenate 'string "Do you see a *Trace-Output* window
                "
                          "with a call to floor passed a 20 and a 3
                          "
                          "and returning a 6 and a 2?"))
            (untrace 'floor)))
    (do-test "trace of subfunction" (defun super-fn nil (our-fn nil) t)
        (and (trace (our-fn il:in super-fn)) (super-fn)
            (do-test-menu-message dtmw 'low
                (concatenate 'string "Do you see a *Trace-Output* window
                "
                          "with a call to our-fn passed a nil
                          "
                          "and returning a nil and a t?"))
            (untrace (our-fn il:in super-fn)))))
(do-test-group ("advise")
           (do-test "simple il:advise il:around of defun" (defun our-fun nil nil)
                 (il:advise 'our-fun 'il:around t)
                 (prog1 (our-fun) (il:unadvise our-fun)))
           (do-test "simple il:advise il:around of fn" (il:defineq (our-fn nil nil))
                 (il:advise 'our-fn 'il:around t)
                 (prog1 (our-fn) (il:unadvise our-fn)))
           (do-test "il:advise redefined broken defun" (defun our-fun nil nil)
                 (il:break our-fun)
                 (defun our-fun nil nil)
                 (il:advise 'our-fun 'il:around t)
                 (prog1 (our-fun) (unbreak our-fun)
                        (il:unadvise our-fun)))
           (do-test "il:advise redefined advised defun (AR 8172)"
                 (defun our-fun nil nil)
                 (il:advise 'our-fun 'il:around t)
                 (defun our-fun nil nil)
                 (il:advise 'our-fun 'il:around t)
                 (prog1 (our-fun) (il:unadvise our-fun)))
           (do-test "il:advise re-defined advised fn" (il:defineq (our-fn nil nil))
                 (il:advise 'our-fn 'il:around t)
                 (il:defineq (our-fn nil nil))
                 (il:advise 'our-fn 'il:around t)
                 (prog1 (our-fn) (il:unadvise our-fn))))
(do-test-group ("il:unadvise" :before (defun our-fun nil nil)
            (il:unadvise our-fun))
           (do-test "simple il:unadvise" (il:advise 'our-fun 'il:around t)
                 (and (our-fun) (eq 'our-fun (first (il:unadvise our-fun)))
                     (not (our-fun))))
           (do-test "il:unadvise t" (il:advise 'our-fun 'il:around t)
                 (and (our-fun) (eq 'our-fun (first (il:unadvise t)))
                     (not (our-fun)))))
(do-test "close user interaction window" (do-test-menu-cleanup dtmw) t)
STOP
(do-test "Error List condition correspondence"
    (dotimes (i 52)
           (ignore-errors (il:seterrorn i)))
    (print "Select Inspect from the menu that will be appearing.")
    (let ((iw (inspect (il:for i il:from 0 il:to 52
```

```
          il:|collect| (ignore-errors (il:seterrorn i))) nil
          (il:|create| il:position il:xcoord il:_ 10 il:ycoord il:_ 10)))
(result
    (do-test-menu-message dtmw 'low
        (concatenate 'string
                  "Does the inspect window have conditions
                  "
                  "correctly corresponding to error number + 1
                  "
                  "in the Lyric release notes section 14.10?"))))
(il:closew iw)
result))
```