```
File created:   24-May-2024 20:54:49  {MEDLEY}<SOURCES>CMLEXEC.;3

     edit by:   mth

  changes to:   (FUNCTIONS PRINT-ALL-DOCUMENTATION)
                (VARS CMLEXECCOMS)

previous date:  20-May-2024 21:28:00  {MEDLEY}<SOURCES>CMLEXEC.;2

  Read Table:   INTERLISP

     Package:   INTERLISP

      Format:    XCCS
```

```
(RPAQQ CMLEXECCOMS
       [(FILES CMLUNDO PROFILE)
        (XCL:PROFILES "EXEC")
        (STRUCTURES COMMAND-ENTRY EXEC-EVENT-ID EXEC-EVENT HISTORY)
                                                                    ; These are public except for command-entry.
        (FUNCTIONS XCL::EXEC-CLOSEFN XCL::EXEC-SHRINKFN XCL::SETUP-EXEC-WINDOW XCL::EXEC-TITLE-FUNCTION FIX-FORM
               XCL::GET-PROCESS-PROFILE XCL::SAVE-CURRENT-EXEC-PROFILE XCL::SETF-GET-PROCESS-PROFILE
               XCL:SET-EXEC-TYPE XCL:SET-DEFAULT-EXEC-TYPE XCL::ENTER-EXEC-FUNCTION)
        (SETFS XCL::GET-PROCESS-PROFILE)
        (FUNCTIONS DO-EVENT EXEC EXEC-EVAL PRINT-ALL-DOCUMENTATION PRINT-DOCUMENTATION VALUE-OF ADD-EXEC
               EXEC-READ-LINE EXEC-EVENT-ID-PROMPT FIND-EXEC-COMMAND)
        (FUNCTIONS CIRCLAR-COPYER)
        (FNS COPY-CIRCLE)
                                                                    ; CIRCLAR-COPYER and COPY-CIRCLE are the solution for
                                                                    ; AR#11172
        (FNS EXEC-READ DIR)
        (VARIABLES *PER-EXEC-VARIABLES* CL:* CL:** CL:*** + CL:++ CL:+++ - / CL:// CL:/// *CURRENT-EVENT*
               *EXEC-ID* XCL:*EXEC-PROMPT* XCL:*EVAL-FUNCTION* *NOT-YET-EVALUATED* *THIS-EXEC-COMMANDS*
               *EXEC-COMMAND-TABLE* *DEBUGGER-COMMAND-TABLE* *CURRENT-EXEC-TYPE* *EXEC-MAKE-UNDOABLE-P*)
        (VARIABLES *EDIT-INPUT-WITH-TTYIN*)
        (FNS DO-APPLY-EVENT DO-HISTORY-SEARCH EVAL-INPUT EVENTS-INPUT EXEC-PRIN1 EXEC-VALUE-OF
             GET-NEXT-HISTORY-EVENT HISTORY-ADD-TO-SPELLING-LISTS HISTORY-NTH PRINT-HISTORY FIND-HISTORY-EVENTS
             PRINT-EVENT PRINT-EVENT-PROMPT PROCESS-EXEC-ID SEARCH-FOR-EVENT-NUMBER \PICK.EVALQT LISPXREPRINT)
        (DECLARE%: DONTEVAL@LOAD DOCOPY (P (MOVD? 'READ 'TTYINREAD)
                                           (MOVD '\PICK.EVALQT '\PROC.REPEATEDLYEVALQT)
                                           (SETQ BackgroundMenu)))
        (FUNCTIONS CASE-EQUALP EXEC-EVENT-PROPS EXEC-PRINT EXEC-FORMAT)
        (ALISTS (BackgroundMenuCommands EXEC))
        (ALISTS (SYSTEMINITVARS LISPXHISTORY GREETHIST))

       ;; Exec Commands

        (DEFINE-TYPES COMMANDS)
        (FUNCTIONS DEFCOMMAND)
        (COMMANDS "?" "??" "CONN" "DA" "DIR" "DO-EVENTS" "FIX" "FORGET" "NAME" "NDIR" "PL" "REDO" "REMEMBER"
               "SHH" "UNDO" "USE" "PP")

       ;; Arrange to use the correct compiler

        (PROP FILETYPE CMLEXEC)
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
               (ADDVARS (NLAMA DIR)
                      (NLAML)
                      (LAMA PROCESS-EXEC-ID PRINT-EVENT PRINT-HISTORY EXEC-PRIN1 EVENTS-INPUT EVAL-INPUT
                            EXEC-READ])

(FILESLOAD CMLUNDO PROFILE)


(XCL:DEFPROFILE "EXEC" (XCL:*DEBUGGER-PROMPT* "")
                       (XCL:*EXEC-PROMPT* "")
                       (*READTABLE* "XCL")
                       (*PACKAGE* "XCL")
                       (XCL:*EVAL-FUNCTION* 'CL:EVAL))


(CL:DEFSTRUCT (COMMAND-ENTRY (:TYPE LIST))
   ARGUMENTS
   FUNCTION
   MODE)


(CL:DEFSTRUCT (EXEC-EVENT-ID (:TYPE LIST))
   NUMBER
   NAME
   dummy)


(CL:DEFSTRUCT (EXEC-EVENT (:TYPE LIST))
   INPUT
   ID
   (VALUE *NOT-YET-EVALUATED*)
   dummy)


(CL:DEFSTRUCT (HISTORY (:TYPE LIST))
   (EVENTS NIL)
   (INDEX 0)
   (SIZE 100)
   (MOD 100))
```

;; These are public except for command-entry.

```
(CL:DEFUN XCL::EXEC-CLOSEFN (XCL::WINDOW)
   [LET [(XCL::PROCESS (WINDOWPROP XCL::WINDOW 'PROCESS]
        (COND
           ((EQ (THIS.PROCESS)
                XCL::PROCESS)
            [ADD.PROCESS `(CLOSEW ',XCL::WINDOW]
            'DON'T)
           ((PROCESSP XCL::PROCESS)
            (CL:IF (TTY.PROCESSP XCL::PROCESS)
                   (TTY.PROCESS T))
            (DEL.PROCESS XCL::PROCESS])


(CL:DEFUN XCL::EXEC-SHRINKFN (XCL::WINDOW)
   (LET [(XCL::PROCESS (WINDOWPROP XCL::WINDOW 'PROCESS]
        (COND
           ((EQ (THIS.PROCESS)
                XCL::PROCESS)
            [ADD.PROCESS `(SHRINKW ',XCL::WINDOW]
            'DON'T)
           ((TTY.PROCESSP XCL::PROCESS)
            (TTY.PROCESS T)
            NIL))))


(CL:DEFUN XCL::SETUP-EXEC-WINDOW (XCL::WINDOW)
   "Add (non-title) properties to a new exec window."
   (WINDOWADDPROP XCL::WINDOW 'CLOSEFN 'XCL::EXEC-CLOSEFN)
   (WINDOWADDPROP XCL::WINDOW 'SHRINKFN 'XCL::EXEC-SHRINKFN)
   XCL::WINDOW)


(CL:DEFUN XCL::EXEC-TITLE-FUNCTION (XCL::WINDOW EXEC-ID)
   [WINDOWPROP XCL::WINDOW 'TITLE (CL:FORMAT NIL "Exec ~A (~A)" EXEC-ID (READTABLEPROP *READTABLE* 'NAME])


(CL:DEFUN FIX-FORM (INPUT &OPTIONAL (CIRCLE-FLAG NIL))
```

;;; Edits a form, in the current window if it is shorter than ttyinfixlimit, or if longer in the display editor using edite. Returns the newly edited form.

```
                                                                      ; Edited by Tomoru Teruuchi
   [COND
      ((OR (NOT *EDIT-INPUT-WITH-TTYIN*)
           (NOT (IMAGESTREAMP (TTYDISPLAYSTREAM)))
           (AND (NOT CIRCLE-FLAG)
                (EQUAL 0 (COUNTDOWN INPUT TTYINFIXLIMIT)))            ; (IGEQ (COUNT INPUT) TTYINFIXLIMIT)  is Original Code. But
                                                                      ; This Codecan't accept circler. Edited by TT (31-May-1990)

           )
       (EDITE (CL:IF (AND (EQ 1 (LENGTH INPUT))
                          (CL:CONSP (CAR INPUT)))
                  (CAR INPUT)
                  INPUT)
              NIL NIL T NIL :CLOSE-ON-COMPLETION)
       INPUT)
      (T (PRINT-EVENT-PROMPT *CURRENT-EVENT*)
         (DSPFONT INPUTFONT T)
         (CURSOR T)                                                   ; make sure can edit (in case cursor smashed somehow?)
         (CL:WHEN NIL                                                 ; Old expression
             (TTYIN "" NIL NIL 'LISPXREAD NIL NIL BUFFER-EXPR-FROM-BELOW *READTABLE*))
         (EXEC-READ-LINE (LET ((%#RPARS NIL)
                               (FONTCHANGEFLG NIL)
                               (*PRINT-ESCAPE* T)
                               (*PRINT-RADIX* (NOT (= *READ-BASE* 10)))
                               (*PRINT-BASE* *READ-BASE*)
                               (*PRINT-LEVEL* NIL)
                               (*PRINT-LENGTH* NIL)
                               (*PRINT-GENSYM* ':REREAD)
                               (*PRINT-ARRAY* T)
                               (*PRINT-STRUCTURE* T))
                              (DECLARE (CL:SPECIAL %#RPARS FONTCHANGEFLG)
                                                                      ; others are already globally special
                                     )
                              (CL:WITH-OUTPUT-TO-STRING (STR)
                                     (FOR X ON INPUT DO (IF CIRCLE-FLAG
                                                            THEN
                                                                ; Edited by TT (31-May-1990) CL:PRIN1 can print circlar
                                                                   (CL:PRIN1 (CAR X)
                                                                          STR)
                                                            ELSEIF (LISTP (CAR X))
                                                            THEN (PRINTDEF (CAR X)
                                                                          (POSITION STR)
                                                                          NIL NIL NIL STR)
                                                            ELSE (PRIN2 (CAR X)
                                                                        STR))
                                                          (AND (CDR X)
                                                               (PRIN1 " " STR])


(CL:DEFUN XCL::GET-PROCESS-PROFILE (&OPTIONAL (XCL::PROCESS (THIS.PROCESS)))
   (PROCESSPROP XCL::PROCESS 'PROFILE))


(CL:DEFUN XCL::SAVE-CURRENT-EXEC-PROFILE ()
   "Resave the profiled bindings of the exec process into their cache."
   (LET [(XCL::PROFILE (XCL::GET-PROCESS-PROFILE (THIS.PROCESS]
        (CL:IF (XCL:PROFILE-P XCL::PROFILE)
               (XCL:SAVE-PROFILE XCL::PROFILE))))


(CL:DEFUN XCL::SETF-GET-PROCESS-PROFILE (&OPTIONAL (XCL::PROCESS (THIS.PROCESS))
                                                   (XCL::PROFILE XCL:*PROFILE*))
   (CL:SETQ XCL::PROFILE (XCL::PROFILIZE XCL::PROFILE))
   (PROCESSPROP XCL::PROCESS 'PROFILE XCL::PROFILE)
   XCL::PROFILE)


(CL:DEFUN XCL:SET-EXEC-TYPE (TYPE)
   "Set the current Exec's type to TYPE"

   ;; The EXECA-FRAME bit is a gross hack to make this function work inside init files.  The problem is that you want to affect the EXEC, regardless of
   ;; who has bound the per-exec variables between here an the EXEC frame.  Yech.

   [LET [(XCL::EXECA-FRAME (STKPOS 'XCL::EXECA0001]
        (COND
           (XCL::EXECA-FRAME (ENVEVAL `(XCL:RESTORE-PROFILE ',TYPE)
                                    XCL::EXECA-FRAME XCL::EXECA-FRAME))
           (T (XCL:RESTORE-PROFILE TYPE])


(CL:DEFUN XCL:SET-DEFAULT-EXEC-TYPE (TYPE)
   (SETTOPVAL 'XCL:*PROFILE* TYPE))


(CL:DEFUN XCL::ENTER-EXEC-FUNCTION (XCL::EXEC-FUNCTION XCL::PROFILE XCL::ID)
   "Start up an exec function in the proper profile, setting the default window title properly."
   (XCL:WITH-PROFILE (XCL:COPY-PROFILE XCL::PROFILE)
          (XCL::EXEC-TITLE-FUNCTION T (PROCESS-EXEC-ID (THIS.PROCESS)
                                            XCL::ID))
          (CL:FUNCALL XCL::EXEC-FUNCTION)))


(CL:DEFSETF XCL::GET-PROCESS-PROFILE XCL::SETF-GET-PROCESS-PROFILE)


(CL:DEFUN DO-EVENT (ORIGINAL-INPUT ENVIRONMENT &OPTIONAL (FUNCTION (FUNCTION EVAL-INPUT)))
                                                                ; Edited by Tomoru Teruuchi
   (PROG (TODO INPUT VALUES COM (ADD-TO-SPELLING-LIST ADDSPELLFLG)
               STR
               (RETRYFLAG NIL)                                   ; A really gross hack for RETRY to always break.  It exists
                                                                ; because: users can setq HELPFLAG anywhere (can't bind it in
                                                                ; DO-EVENTand set it in RETRY), RETRY operates on
                                                                ; commands (can't wrap the form with a binding of HELPFLAG).

               )
         (DECLARE (CL:SPECIAL RETRYFLAG))                        ; RETRY command sets this variable if it wants to be sure to
                                                                ; break.

         (DSPFONT PRINTOUTFONT T)
         (SETQ INPUT ORIGINAL-INPUT)
     RETRY
         (SETQ TODO (COPY-CIRCLE INPUT))                         ; Break EQ link between input and evaluated form (todo), so that
                                                                ; in-place mods don't affect history.

         [COND
            [[AND (OR (STRINGP (CAR INPUT))
                      (CL:SYMBOLP (CAR INPUT)))
                  (PROGN (SETQ STR (STRING (CAR INPUT)))
                         (SOME *THIS-EXEC-COMMANDS* (FUNCTION (LAMBDA (TABLE)
                                                                (SETQ COM (GETHASH STR TABLE]

             ;; Handle exec commands.

             (CL:ECASE (COMMAND-ENTRY-MODE COM)
                 (:QUIET
                 [MAPC (SETQ VALUES (CL:MULTIPLE-VALUE-LIST (CL:FUNCALL (COMMAND-ENTRY-FUNCTION COM)
                                                                        INPUT ENVIRONMENT)))
                      (FUNCTION (LAMBDA (X)
                                  (EXEC-PRINT X]
                 (SETQ IT (CAR VALUES))                              ; just do it and return
                 (RETURN))
             ((:HISTORY :INPUT)                                     ;  create new input. If an error occurs while handling the
                                                                    ; command, the INPUT will be left as the original input.
                 (CL:WHEN *CURRENT-EVENT*
                     (CL:SETF (EXEC-EVENT-INPUT *CURRENT-EVENT*)
                            INPUT))
                 (SETQ INPUT (CL:FUNCALL (COMMAND-ENTRY-FUNCTION COM)
                                    INPUT ENVIRONMENT))
                 (CL:WHEN *CURRENT-EVENT*
                     (CL:SETF (EXEC-EVENT-INPUT *CURRENT-EVENT*)
                            INPUT)                                   ;  Overwrite the original input with the newly generated one.
                     (CL:SETF (EXEC-EVENT-PROPS *CURRENT-EVENT*)
                            (LIST* '*HISTORY* ORIGINAL-INPUT (EXEC-EVENT-PROPS *CURRENT-EVENT*))))
                 (GO RETRY)                                          ;  could have generated a command

                 )
             ((NIL :EVAL)                                           ;  normal kind of command, just apply
                 [SETQ TODO `((CL:FUNCALL ',(COMMAND-ENTRY-FUNCTION COM)
                                    ',INPUT
                                    ',ENVIRONMENT]
                 (SETQ ADD-TO-SPELLING-LIST NIL)
                 (CL:WHEN *CURRENT-EVENT*
                     (CL:SETF (EXEC-EVENT-INPUT *CURRENT-EVENT*)
                            INPUT))))]
          (T
             ;; Handle non-exec commands (fns, functions, macros, etc.).

             (CL:WHEN *CURRENT-EVENT*
                 (CL:SETF (EXEC-EVENT-INPUT *CURRENT-EVENT*)
                        INPUT))
             (CL:WHEN *EXEC-MAKE-UNDOABLE-P*
                 [if (CDR TODO)
                     then (SETQ TODO (CONS (OR (CDR (ASSOC (CAR TODO)
                                                            LISPXFNS))
                                                (CAR TODO))
                                            (CDR TODO)))
                    else (SETQ TODO (LIST (XCL::MAKE-UNDOABLE (CAR TODO)
                                                  NIL])]
        (AND ADD-TO-SPELLING-LIST (HISTORY-ADD-TO-SPELLING-LISTS TODO))
        (SETQ LISPXHIST *CURRENT-EVENT*)
        (DSPFONT PRINTOUTFONT T)
        (RETURN (LET ((HELPCLOCK (CLOCK 2))
                      VALUES)
                     (DECLARE (CL:SPECIAL HELPCLOCK))
                     (CL:SETQ CL:+++ CL:++ CL:++ + + - - (CAR INPUT))

                     ;; the book doesn't define what - and friends should be when input is in APPLY format.  Here it says it is just the function
                     ;; name.

                     [SETQ VALUES (CL:MULTIPLE-VALUE-LIST (CL:IF RETRYFLAG
                                                                (LET ((HELPFLAG 'BREAK!))
                                                                     (DECLARE (CL:SPECIAL HELPFLAG))
                                                                     (CL:FUNCALL FUNCTION TODO ENVIRONMENT))
                                                                (CL:FUNCALL FUNCTION TODO ENVIRONMENT))]
                     (CL:SETQ CL:/// CL:// CL:// / / VALUES)
                     (CL:UNLESS (EQ 'NOBIND (CAR VALUES))        ; Be a bit careful about NOBIND.
                         (CL:SETQ CL:*** CL:** CL:** CL:* CL:* (SETQ IT (CAR VALUES))))
                     (CL:WHEN *CURRENT-EVENT*
                         (CL:SETF (EXEC-EVENT-VALUE *CURRENT-EVENT*)
                                (CAR VALUES))
                         (CL:SETF (EXEC-EVENT-PROPS *CURRENT-EVENT*)
                                (LIST* 'LISPXVALUES VALUES (EXEC-EVENT-PROPS *CURRENT-EVENT*))))
                     (DSPFONT VALUEFONT T)
                     (for X in VALUES do (EXEC-PRINT X))
                     VALUES))))


(CL:DEFUN EXEC (&KEY XCL::TOP-LEVEL-P                              ; True of top level execs.  Used for event number restarting and
                                                                    ; profile caching.
                     (XCL::WINDOW (WFROMDS (TTYDISPLAYSTREAM)))
                                                                    ; Window for this exec, if any.
                     (XCL::TITLE NIL XCL::TITLE-SUPPLIED)           ; If given, specific title for this window.
                     ((:COMMAND-TABLES *THIS-EXEC-COMMANDS*)
                      (LIST *EXEC-COMMAND-TABLE*))                  ; List of hash tables to look up commands in.
                     XCL::ENVIRONMENT                               ; Lexical environment to evaluate things in, default NIL.
                     XCL::PROMPT                                    ; Special prompt to use (optional).
                     ((:FUNCTION XCL::FN)
                      'EVAL-INPUT)                                  ; Function for processing input.
                     XCL::PROFILE                                   ; Optional profile, sets the exec's bindings.
                     XCL::ID                                        ; A handle on the exec.
                     &ALLOW-OTHER-KEYS                              ; To catch obsolete calls
                     &AUX
                     (*EXEC-ID* (PROCESS-EXEC-ID (THIS.PROCESS)
                                       XCL::ID))
                      (XCL::PROFILE-CACHE (XCL::GET-PROCESS-PROFILE (THIS.PROCESS)))
                                                              ; The exec's cached profile (if entering from a hardreset).

                      )
   [CL:PROGV (MAPCAR *PER-EXEC-VARIABLES* (FUNCTION CAR))
          [MAPCAR *PER-EXEC-VARIABLES* (FUNCTION (LAMBDA (XCL::X)
                                                   (EVAL (CADR XCL::X]
          (CL:WHEN (OR (NULL XCL::TOP-LEVEL-P)
                       (NULL XCL::PROFILE-CACHE))             ; If not hardresetting...
              (CL:WHEN XCL::PROFILE                           ; then initialize the profile vars.
                  (XCL:RESTORE-PROFILE XCL::PROFILE))
              (CL:WHEN XCL::PROMPT                            ; If a special prompt was provided (as from the debugger)...
                  (CL:SETQ XCL:*EXEC-PROMPT* XCL::PROMPT)     ; ...use it.

                  ))
          (CL:WHEN XCL::TOP-LEVEL-P
              (CL:IF (NULL XCL::PROFILE-CACHE)                ; This was a new entry into top level exec.
                  (CL:SETF (XCL::GET-PROCESS-PROFILE (THIS.PROCESS))
                         (XCL:SAVE-PROFILE (XCL:COPY-PROFILE "EXEC")))
                                                              ; ...make a fresh cache and save bindings into it.
                  (XCL:RESTORE-PROFILE XCL::PROFILE-CACHE)    ; ...otherwise it was a HARDRESET.

                  ))
          (CL:WHEN XCL::WINDOW
              (COND
                 ((NOT XCL::TITLE-SUPPLIED)                   ; If no title was supplied, set it to the default.
                  (XCL::EXEC-TITLE-FUNCTION XCL::WINDOW *EXEC-ID*))
                 (XCL::TITLE                                  ; If a non-nil title was supplied, set the title to it.
                        (WINDOWPROP XCL::WINDOW 'TITLE XCL::TITLE)))
              (TTYDISPLAYSTREAM (DECODE/WINDOW/OR/DISPLAYSTREAM XCL::WINDOW)))
          (LET [(*CURRENT-EVENT* NIL)                         ; the event being processed. Used by some commands
                (XCL::OLD-DS (CL:IF XCL::WINDOW
                                 (TTYDISPLAYSTREAM (DECODE/WINDOW/OR/DISPLAYSTREAM XCL::WINDOW)))]
               (CL:LOOP (CL:FORMAT T "~&~%%")                 ; newlines to notice that this is a new instance of the exec
                      (PROG1 [ERSETQ (CL:LOOP                 ; loop until errors out
                                         (CL:SETQ *CURRENT-EVENT* (GET-NEXT-HISTORY-EVENT LISPXHISTORY
                                                                         *EXEC-ID* XCL:*EXEC-PROMPT*
                                                                         (NOT XCL::TOP-LEVEL-P)))
                                                              ; This optimization keeps HARDRESET from generating all new
                                                              ; event numbers for all execs that are open.
                                         (PRINT-EVENT-PROMPT *CURRENT-EVENT*)
                                         (DSPFONT INPUTFONT T)
                                         (LET ((XCL::ORIGINAL-INPUT (EXEC-READ-LINE))
                                               (LISPXHIST LISPXHIST)
                                               (HELPCLOCK 0))
                                              (DECLARE (CL:SPECIAL LISPXHIST HELPCLOCK))
                                              (CL:UNLESS (CL:EQUAL XCL::ORIGINAL-INPUT '(NIL))
                                                  (DO-EVENT XCL::ORIGINAL-INPUT XCL::ENVIRONMENT XCL::FN)
                                                  (CL:WHEN XCL::TOP-LEVEL-P
                                                              ; Used to determine whether to cache the settings of the profile
                                                              ; back into the process (for retrieval in case of hardreset).
                                                         (XCL::SAVE-CURRENT-EXEC-PROFILE))))]
                          (CL:WHEN XCL::WINDOW (TTYDISPLAYSTREAM XCL::OLD-DS)))])


(CL:DEFUN EXEC-EVAL (FORM &OPTIONAL ENVIRONMENT &KEY (PROMPT ">")
                                 (ID "eval/")
                                 ((:TYPE *CURRENT-EXEC-TYPE*)
                                  'COMMON-LISP))              ; Edited by JDS 16-Aug-90 12:55.
   (LET ((*CURRENT-EVENT* (GET-NEXT-HISTORY-EVENT LISPXHISTORY ID PROMPT T))
         (LISPXHIST LISPXHIST)
         (HELPCLOCK 0)
         VALUES)
        (DECLARE (CL:SPECIAL *CURRENT-EVENT* LISPXHIST HELPCLOCK))
        (SETQ VALUES (CL:MULTIPLE-VALUE-LIST (EVAL-INPUT (CL:SETF (EXEC-EVENT-INPUT *CURRENT-EVENT*)
                                                                  (LIST FORM))
                                                    ENVIRONMENT)))
        (SETQ IT (CAR VALUES))
        (COND
           (*CURRENT-EVENT*

                  ;; Only update the current event if it's not NIL.  This might happen, e.g., if LISPXHIST has been set to NIL by the user.

                  (CL:SETF (EXEC-EVENT-PROPS *CURRENT-EVENT*)
                         (LIST* 'LISPXVALUES VALUES (EXEC-EVENT-PROPS *CURRENT-EVENT*)))
                  (CL:SETF (EXEC-EVENT-VALUE *CURRENT-EVENT*)
                         IT)))
        (CL:VALUES-LIST VALUES)))


(CL:DEFUN PRINT-ALL-DOCUMENTATION (NAME)                      ; Edited 24-May-2024 20:52 by mth
   "Print all documentation strings for NAME (as symbol and string)."
   (LET ((FOUND NIL))
        (CL:DOLIST (TYPE FILEPKGTYPES)
            (CL:WHEN (AND (CL:SYMBOLP TYPE)
                          (OR (GET TYPE :DEFINED-BY)
                              (GET TYPE 'DEFINED-BY))
                          (HASH-TABLE-FOR-DOC-TYPE TYPE))
                (SETQ FOUND (OR (PRINT-DOCUMENTATION NAME TYPE)
                                FOUND))
                (CL:WHEN (CL:SYMBOLP NAME)
                    (SETQ FOUND (OR (PRINT-DOCUMENTATION (STRING NAME)
                                           TYPE)
                                    FOUND)))))
        (CL:UNLESS FOUND (CL:FORMAT *TERMINAL-IO* "No documentation found.~%%"))))


(CL:DEFUN PRINT-DOCUMENTATION (NAME TYPE)
   "If it exists, print documentation for NAME as TYPE.  Returns T if doc was found, else NIL."
   [LET ((DOC (CL:DOCUMENTATION NAME TYPE)))
        (AND DOC (TRUE (CL:FORMAT *TERMINAL-IO* "~&~A (~A)" DOC (OR (CL:DOCUMENTATION NAME 'DEFINE-TYPES)
                                                                    TYPE])


(DEFMACRO VALUE-OF (&REST EVENT-SPEC)
   `(EXEC-VALUE-OF ',EVENT-SPEC))


(CL:DEFUN ADD-EXEC (&KEY (XCL::PROFILE XCL:*PROFILE*)
                          XCL::REGION XCL::TTY (EXEC 'EXEC)
                          XCL::ID &ALLOW-OTHER-KEYS)
   (LET* [(XCL::WINDOW (XCL::SETUP-EXEC-WINDOW (CREATEW XCL::REGION "Exec")))
          (XCL::HANDLE (ADD.PROCESS `[PROGN (TTYDISPLAYSTREAM ',XCL::WINDOW)
                                            (PROCESSPROP (THIS.PROCESS)
                                                   'WINDOW
                                                   ',XCL::WINDOW)
                                            ,(CASE EXEC
                                                 (EXEC `(EXEC :TOP-LEVEL-P T :PROFILE ',XCL::PROFILE :ID
                                                              ',XCL::ID))
                                                 (T `(XCL::ENTER-EXEC-FUNCTION ',EXEC ',XCL::PROFILE
                                                            ',XCL::ID)))]
                              'NAME
                              'EXEC
                              'RESTARTABLE T 'INTERRUPTS (LISPINTERRUPTS]
         (AND XCL::TTY (TTY.PROCESS XCL::HANDLE))
         XCL::HANDLE))


(CL:DEFUN EXEC-READ-LINE (&OPTIONAL BUFFER-STRING)
   ;; Code stolen from READLINE, and not cleaned up.
   [PROG (LINE SPACEFLG CHRCODE (*IN-THE-DEBUGGER* NIL))
         (COND
            ((AND (READP T)
                  (SYNTAXP (PEEKCCODE T T)
                         'EOL))                           ; Avoid picking up end of line as a NIL.
             (READC T)))
         (SETQ LINE (LIST (EXEC-READ BUFFER-STRING)))
     TOP (COND
            ((LISTP (CAR LINE))                           ; If we got a list, return right away--it's a standard EVAL form of
                                                          ; input
             (GO OUT)))
     LP  (SETQ SPACEFLG NIL)                              ; to distinguish between
                                                          ; FOO (A B)
                                                          ; FOO(A B)
                                                          ; the latter has no space and returns right away
     LP1 (COND
            ((NOT (READP T))                              ; nothing more in line buffer, so must have consumed last thing
                                                          ; on the line
             (GO OUT))
            ((NULL (SETQ CHRCODE (PEEKCCODE T T)))        ; PEEKCCODE can return NIL when stream is at EOF.
                                                          ; However, we already checked for READP before getting here.
             (GO OUT))
            ((SYNTAXP CHRCODE 'EOL)
             (READC T)
             (GO OUT))
            ((OR (SYNTAXP CHRCODE 'RIGHTPAREN *READTABLE*)
                 (SYNTAXP CHRCODE 'RIGHTBRACKET *READTABLE*))
             (AND (READ T *READTABLE*)
                  (SHOULDNT))
             (AND (NULL (CDR LINE))
                  (SETQ LINE (NCONC1 LINE NIL)))          ;  A ")" is treated as NIL if it is the second thing on the line when
                                                          ; EXEC-READ-LINE is called
             (GO OUT))
            ((EQ CHRCODE (CHARCODE SPACE))
             (SETQ SPACEFLG T)
             (READC T)
             (GO LP1)))
         (SETQ LINE (NCONC1 LINE (EXEC-READ)))
         (COND
            ((NULL (OR (SYNTAXP (SETQ CHRCODE (CHCON1 (LASTC T)))
                              'RIGHTPAREN *READTABLE*)
                       (SYNTAXP CHRCODE 'RIGHTBRACKET *READTABLE*)))
             (GO LP))
            ((NOT SPACEFLG)                               ; A list terminates the line if it is the second element on the line,
                                                          ; not preceded by a space.
                ;; [JDS 1/12/88:  This used to test (AND (NOT SPACEFLG) (READP T)), and loop if there were more input pending.  This seems
                ;; wrong, because when you type it should throw the carriage at once, and not depend on how fast you're typing.  Further, when there's
                ;; type-ahead, it's often followed by a SPACE, to prevent output pausing.  With the old test here, that would hang up a final eval-quote
                ;; form without executing it.]
                (GO OUT))
               (T (GO LP)))
            (GO LP)
      OUT (RETURN (COND
                     ((AND (LISTP LINE)
                           CTRLUFLG)                                   ; Edit interrupt during reading--forces structure editor use.
                      (SETQ CTRLUFLG NIL)
                      (LET ((*EDIT-INPUT-WITH-TTYIN* NIL))
                           (FIX-FORM LINE)))
                     (T LINE])


(DEFMACRO EXEC-EVENT-ID-PROMPT (EVENT-ID)
   `(CDDR ,EVENT-ID))


(CL:DEFUN FIND-EXEC-COMMAND (NAME TABLE)
   "Find an exec command based on its name (either a string or a symbol).  Returns the command entry or NIL if
   not found."
   (CL:WHEN (OR (CL:STRINGP NAME)
                (CL:SYMBOLP NAME))
       (LET ((STR (CL:IF (CL:SYMBOLP NAME)
                         (CL:SYMBOL-NAME NAME)
                         NAME)))
            (CL:SOME #'(CL:LAMBDA (TABLE)
                               (SETQ COM (GETHASH STR TABLE)))
                   TABLE))))


(CL:DEFUN CIRCLAR-COPYER (INPUT)                                      ; Edited by TT 31-May-1990
   (PROG (SCANBUF REST VAL NEW BODY ID AUX (CIRCLAR-FLAG NIL))
         (COND
            ((NLISTP INPUT)
             (RETURN INPUT))
            (T [push SCANBUF (CONS INPUT (SETQ VAL (CONS NIL NIL]
               (push REST VAL)
               (RPLACA VAL (CAR INPUT))
               (RPLACD VAL (CDR INPUT))

;;; (COND ((EQ X (CAR X)) (RPLACA VAL VAL)) (T (RPLACA VAL (CAR X)))) (COND ((EQ X (CDR X)) (RPLACD VAL VAL)) (T (RPLACD VAL (CDR
;;; X))))

               ))                                                     ; Initialization is over
      LP   (SETQ BODY (pop REST))
      LP0 [COND
            ((NULL BODY)
             (RETURN (CL:VALUES VAL CIRCLAR-FLAG)))
            ((NLISTP BODY)
             (GO LP))
            (T (SETQ NEW BODY)
               (COND
                  ((NLISTP (CDR NEW)))
                  ((SETQ ID (FASSOC (CDR NEW)
                                    SCANBUF))
                   (SETQ CIRCLAR-FLAG T)
                   (RPLACD NEW (CDR ID)))
                  (T [push REST (SETQ AUX (CONS (CADR NEW)
                                                (CDDR NEW]
                     (push SCANBUF (CONS (CDR NEW)
                                         AUX))
                     (RPLACD NEW AUX)))
               (COND
                  ((NLISTP (CAR NEW)))
                  ((SETQ ID (FASSOC (CAR NEW)
                                    SCANBUF))
                   (SETQ CIRCLAR-FLAG T)
                   (RPLACA NEW (CDR ID)))
                  (T [push REST (SETQ AUX (CONS (CAAR NEW)
                                                (CDAR NEW]
                     (push SCANBUF (CONS (CAR NEW)
                                         AUX))
                     (RPLACA NEW AUX]
         (GO LP)))

(DEFINEQ

(COPY-CIRCLE
  [LAMBDA (X)                                                         ; Edited 23-May-90 15:02 by Tomtom
    (PROG (SCANBUF REST VAL NEW BODY ID AUX)
          (COND
             ((NLISTP X)
              (RETURN X))
             (T [push SCANBUF (CONS X (SETQ VAL (CONS NIL NIL]
                (push REST VAL)
                (RPLACA VAL (CAR X))
                (RPLACD VAL (CDR X))
```

;;; (COND ((EQ X (CAR X)) (RPLACA VAL VAL)) (T (RPLACA VAL (CAR X)))) (COND ((EQ X (CDR X)) (RPLACD VAL VAL)) (T (RPLACD VAL (CDR
;;; X))))

```
                ))                                                   ; Initialization is over
       LP  (SETQ BODY (pop REST))
       LP0 [COND
             ((NULL BODY)
              (RETURN VAL))
             ((NLISTP BODY)
              (GO LP))
             (T (SETQ NEW BODY)
                (COND
                   ((NLISTP (CDR NEW)))
                   ((SETQ ID (FASSOC (CDR NEW)
                                     SCANBUF))
                    (RPLACD NEW (CDR ID)))
                   (T [push REST (SETQ AUX (CONS (CADR NEW)
                                                 (CDDR NEW]
                      (push SCANBUF (CONS (CDR NEW)
                                          AUX))
                      (RPLACD NEW AUX)))
                (COND
                   ((NLISTP (CAR NEW)))
                   ((SETQ ID (FASSOC (CAR NEW)
                                     SCANBUF))
                    (RPLACA NEW (CDR ID)))
                   (T [push REST (SETQ AUX (CONS (CAAR NEW)
                                                 (CDAR NEW]
                      (push SCANBUF (CONS (CAR NEW)
                                          AUX))
                      (RPLACA NEW AUX]
          (GO LP])

)
```

;; CIRCLAR-COPYER and COPY-CIRCLE are the solution for AR#11172

```
(DEFINEQ
```

(EXEC-READ

```
  [CL:LAMBDA (&OPTIONAL BUFFER-STRING)                               ; Edited  4-Feb-88 18:22 by amd
```

;;; Reads structure from the user (in the exec), taking care to handle read errors so that they will be edited and fixed.

```
         (HANDLER-BIND [[XCL:SYMBOL-COLON-ERROR #'(LAMBDA (CONDITION)
                                                    (DECLARE (CL:SPECIAL CTRLUFLG))
                                                    (CL:FORMAT *TERMINAL-IO* "~a~%%" CONDITION)
                                                    (SETQ CTRLUFLG T)
                                                    (XCL:ESCAPE-COLONS-PROCEED)
                                                    (SHOULDNT "Didn't find XCL::ESCAPE-COLONS-PROCEED"]
                        [XCL:MISSING-EXTERNAL-SYMBOL #'(LAMBDA (CONDITION)
                                                         (DECLARE (CL:SPECIAL CTRLUFLG))
                                                         (CL:FORMAT *TERMINAL-IO* "~a~%%" CONDITION)
                                                         (SETQ CTRLUFLG T)
                                                         (XCL:MAKE-INTERNAL-PROCEED)
                                                         (SHOULDNT "Didn't find XCL:MAKE-INTERNAL-PROCEED"]
                        (XCL:MISSING-PACKAGE #'(LAMBDA (CONDITION)
                                                 (DECLARE (CL:SPECIAL CTRLUFLG))
                                                 (CL:FORMAT *TERMINAL-IO* "~a~%%" CONDITION)
                                                 (SETQ CTRLUFLG T)
                                                 (XCL:UGLY-SYMBOL-PROCEED)
                                                 (SHOULDNT "Didn't find XCL:UGLY-SYMBOL-PROCEED"]
                (COND
                   ([OR (NOT (GETD 'TTYIN))
                        (NOT *EDIT-INPUT-WITH-TTYIN*)
                        (NOT (DISPLAYSTREAMP (GETSTREAM T 'OUTPUT]
                                                                     ; If debugging and TTYIN breaks, don't want to die
                    (CL:READ T))
                   (T (LET (X)
                           (COND
                              ((OR (LINEBUFFER-SKIPSEPRS T *READTABLE*)
                                   (until (SETQ X (TTYIN "" NIL NIL '(EVALQT FILLBUFFER NOPROMPT)
                                                        NIL NIL BUFFER-STRING *READTABLE*))
                                      do ;; Until he types something at all, keep printing the event-number prompt.
                                         (PRINT-EVENT-PROMPT *CURRENT-EVENT*)
                                         (DSPFONT INPUTFONT T))
                                   (EQ X T))
                               (CL:READ-PRESERVING-WHITESPACE T))
                              (T (CAR X])
(DIR
  [NLAMBDA ARGS                                                     ; Edited 12-Mar-87 16:08 by raf
    (DODIR ARGS])

)


(CL:DEFPARAMETER *PER-EXEC-VARIABLES*
   '((CL:* CL:*)
     (CL:** CL:**)
     (CL:*** CL:***)
     (+ +)
     (CL:++ CL:++)
     (CL:+++ CL:+++)
     (- -)
     (/ /)
     (CL:// CL://)
     (CL:/// CL:///)
     (HELPFLAG T)
     (*EVALHOOK* NIL)
     (*APPLYHOOK* NIL)
     (*ERROR-OUTPUT* *TERMINAL-IO*)
     (*READTABLE* *READTABLE*)
     (*PACKAGE* *PACKAGE*)
     (XCL:*EVAL-FUNCTION* XCL:*EVAL-FUNCTION*)
     (XCL:*EXEC-PROMPT* XCL:*EXEC-PROMPT*)
     (XCL:*DEBUGGER-PROMPT* XCL:*DEBUGGER-PROMPT*))
   "List of (non-profile) variables rebound for each Exec")

(CL:DEFVAR CL:* NIL)

(CL:DEFVAR CL:** NIL)

(CL:DEFVAR CL:*** NIL)

(CL:DEFVAR + NIL)

(CL:DEFVAR CL:++ NIL)

(CL:DEFVAR CL:+++ NIL)

(CL:DEFVAR - NIL)

(CL:DEFVAR / NIL
   "Holds a list of all the values returned by the most recent top-level EVAL.")

(CL:DEFVAR CL:// NIL
   "Gets the previous value of / when a new value is computed.")

(CL:DEFVAR CL:/// NIL
   "Gets the previous value of // when a new value is computed.")

(CL:DEFVAR *CURRENT-EVENT* NIL
   "contains the current event being processed. Used for communicating between Exec and commands")

(CL:DEFVAR *EXEC-ID* NIL
   "A unique per-exec-process ID so that commands that search the history list can find this Exec's events")

(CL:DEFVAR XCL:*EXEC-PROMPT* "> "
   "Default prompt used by exec")

(CL:DEFPARAMETER XCL:*EVAL-FUNCTION* 'CL:EVAL "The evaluator to use in the exec")

(CL:DEFVAR *NOT-YET-EVALUATED* "<not yet evaluated>")

(CL:DEFVAR *THIS-EXEC-COMMANDS* NIL
   "List of command hash-tables for the current executive")

(DEFGLOBALVAR *EXEC-COMMAND-TABLE* (HASHARRAY 30 NIL 'STRING-EQUAL-HASHBITS 'STRING-EQUAL)
                                          "hash-table for top level exec commands")
```

```
(DEFGLOBALVAR *DEBUGGER-COMMAND-TABLE* (HASHARRAY 20 NIL 'STRING-EQUAL-HASHBITS 'STRING-EQUAL)
                                       "string-equal hash-table for debugger commands")

(CL:DEFVAR *CURRENT-EXEC-TYPE* NIL
   "Rebound under Exec; if NIL, means use default")

(CL:DEFPARAMETER *EXEC-MAKE-UNDOABLE-P* T
   "global parameter controls whether the exec makes input undoable")

(CL:DEFVAR *EDIT-INPUT-WITH-TTYIN* T)

(DEFINEQ

(DO-APPLY-EVENT
  [LAMBDA (TODO)                                               (* lmm "31-Jul-86 03:22")
    (CL:IF (CL:MACRO-FUNCTION (CAR TODO))
        (CL:IF (EQ (ARGTYPE (CAR TODO))
                   3)
            (CL:FUNCALL (CAR TODO)
                   (CL:IF (CDDR TODO)
                       (CDR TODO)
                       (CADR TODO)))
            (CL:EVAL TODO))
        (CL:APPLY (CAR TODO)
               (CADR TODO)))])

(DO-HISTORY-SEARCH
  [LAMBDA (SPEC PRED-P VALUE-P)                                ; Edited 10-Mar-87 18:53 by raf

    ;; SEARCHES HISTORY LIST, LOOKING FOR SPEC AND RESETTING *EVENTS* TO THE CORRESPONDING TAIL.

    (PROG (PAT1 PAT2 TEM PRED)
          (DECLARE (CL:SPECIAL *EVENTS*))                      ; Setup by FIND-HISTORY-EVENTS
          [COND
             ((NOT PRED-P)
              (SETQ PAT2 (EDITFPAT SPEC T]
      LP  [COND
             ((EQ (CAR *EVENTS*)
                  *CURRENT-EVENT*)
              (SETQ *EVENTS* (CDR *EVENTS*]
          [COND
             ((COND
                 (PRED-P (APPLY* SPEC (CAR *EVENTS*)))
                 [PAT1 (EDIT4E PAT1 (CAR (EXEC-EVENT-INPUT (CAR *EVENTS*]
                 (T (EDITFINDP [COND
                                  (VALUE-P (CL:GETF (EXEC-EVENT-PROPS (CAR *EVENTS*))
                                                  'LISPXVALUES))
                                  (T (EXEC-EVENT-INPUT (CAR *EVENTS*]
                           PAT2 T)))
              (RETURN *EVENTS*))
             (T (SETQ *EVENTS* (CDR *EVENTS*]
      LP1 (COND
             ((NULL *EVENTS*)
              (RETURN NIL)))
          (GO LP])

(EVAL-INPUT
  [CL:LAMBDA (TODO ENV)                                        ; Edited 23-Nov-87 13:07 by raf
         (CASE XCL:*EVAL-FUNCTION*
             (EVAL                                             ; Interlisp EVAL
                [COND
                   [(CDR TODO)                                 ; this is the 'apply' case

                    ;; we first check for input of things like macros in apply format or Interlisp NLAMBDA functions (which have a
                    ;; MACRO-FUNCTION)

                    (if [OR (CDDR TODO)
                            (AND (CADR TODO)
                                 (NLISTP (CADR TODO]
                        then (if (FMEMB (ARGTYPE (CAR TODO))
                                        '(1 3))
                                 then                          ; this is an Interlisp NLAMBDA function (1 = spread, 3 =
                                                               ; nospread).
                                      [if (AND (EQ (ARGTYPE (CAR TODO))
                                                   3)
                                               (CDDR TODO))
                                          then (APPLY (CAR TODO)
                                                      (CDR TODO))
                                        else (if (CDDR TODO)
                                                 then (PRIN1 "... = ")
                                                      (PRINT TODO)
                                                      (APPLY (CAR TODO)
                                                      (CDR TODO))
                                          else (APPLY (CAR TODO)
                                                      (CADR TODO]
                              else ;; evaluate the entire input list as if it were typed in with parens around it, e.g.  a 'FOR I FROM 1 TO 10
                                   ;; DO ...' possibly bogus 'DWIM' case
                                   (EVAL TODO))
                       else                                               ; a normal apply case
                           (if (CDDR TODO)
                              then (PRIN1 "... = ")
                                   (PRINT TODO)
                                   (APPLY (CAR TODO)
                                          (MAPCAR (CDR TODO)
                                                  (FUNCTION EVAL)))
                             else (APPLY (CAR TODO)
                                         (CADR TODO]
                   (T                                                     ; a normal eval case
                      (EVAL (CAR TODO])
              (T                                                          ; Common Lisp EVAL

                 ;; maybe should have used ECASE and checked for Common-Lisp explicitly, but could get recursive errors if *current-exec-type*
                 ;; was rebound

                 (COND
                    [(CDR TODO)                                           ; this is the 'apply' case

                     ;; we first check for input of things like macros in apply format or Interlisp NLAMBDA functions (which have a
                     ;; MACRO-FUNCTION)

                     (COND
                        [(CL:MACRO-FUNCTION (CAR TODO))
                         (COND
                            [(FMEMB (ARGTYPE (CAR TODO))
                                    '(1 3))                               ; this is an Interlisp NLAMBDA function (1 = spread, 3 =
                                                                          ; nospread).
                             (COND
                                ((AND (EQ (ARGTYPE (CAR TODO))
                                          3)
                                      (CDDR TODO))
                                 (APPLY (CAR TODO)
                                        (CDR TODO)))
                                (T (COND
                                      ((CDDR TODO)
                                       (PRIN1 "... = ")
                                       (PRINT TODO)
                                       (APPLY (CAR TODO)
                                              (CDR TODO)))
                                      (T (APPLY (CAR TODO)
                                                (CADR TODO]
                            (T ;; evaluate the entire input list as if it were typed in with parens around it, e.g.  a 'FOR I FROM 1 TO 10 DO ...'
                               ;; possibly bogus 'DWIM' case
                               (CL:EVAL TODO ENV]
                        (T                                                ; a normal apply case
                           (COND
                              [(CDDR TODO)
                               (PRIN1 "... = ")
                               (PRINT TODO)
                               (CL:APPLY (CAR TODO)
                                      (CL:MAPCAR #'(CL:LAMBDA (A)
                                                              (CL:EVAL A ENV))
                                             (CDR TODO]
                              (T (CL:APPLY (CAR TODO)
                                        (CADR TODO]
                   (T                                                     ; a normal eval case
                      (CL:EVAL (CAR TODO)
                             ENV))))))])
```

(EVENTS-INPUT

```
  [CL:LAMBDA (EVENTS)                                                     ; Edited 26-Nov-86 11:16 by lmm
                                                                          ; takes a list of events and returns the input concatenated into a
                                                                          ; single event, as appropriate
         (IF (CDR EVENTS)
             THEN [CONS 'DO-EVENTS (FOR EVENT IN EVENTS COLLECT (IF (CDR (EXEC-EVENT-INPUT EVENT))
                                                                    THEN (CONS 'EVENT (EXEC-EVENT-INPUT EVENT)
                                                                               )
                                                                  ELSE (CAR (EXEC-EVENT-INPUT EVENT]
           ELSE (LET* ((INPUT (EXEC-EVENT-INPUT (CAR EVENTS)))
                       (TAIL (FMEMB HISTSTR0 INPUT)))
                      (IF TAIL
                          THEN (LDIFF INPUT TAIL)
                        ELSE INPUT])
```

(EXEC-PRIN1

```
  (CL:LAMBDA (VALUE)                                                      ; Edited 23-Feb-87 18:15 by raf
         (WRITE VALUE :STREAM *TERMINAL-IO* :ESCAPE T)))
```

```
(EXEC-VALUE-OF
  [LAMBDA (EVENT-SPEC)                                                    (* lmm "11-Sep-86 17:28")
    (CL:VALUES-LIST (LISTGET (EXEC-EVENT-PROPS (CAR (FIND-HISTORY-EVENTS EVENT-SPEC LISPXHISTORY)))
                           'LISPXVALUES])

(GET-NEXT-HISTORY-EVENT
  [LAMBDA (HISTORY ID PROMPT FIRST-ONLY)                                  ; Edited  2-Mar-87 15:34 by raf
    (for EVENT in (HISTORY-EVENTS HISTORY) do (CL:WHEN (EQ (CADR (LISTP (EXEC-EVENT-ID EVENT)))
                                                           ID)
                                                  (CL:IF (AND (NULL (EXEC-EVENT-INPUT EVENT))
                                                              (NULL (EXEC-EVENT-PROPS EVENT)))
                                                      (PROGN (CL:SETF (CDDR (EXEC-EVENT-ID EVENT))
                                                                    PROMPT)
                                                             (RETURN EVENT))
                                                      (GO $$OUT)))
                                              (if FIRST-ONLY
                                                  then                    ; only do this for the first event
                                                       (GO $$OUT))
       finally (COND
                  (HISTORY                                                ; Watch out for NIL LISPXHISTORY
                         (SETQ EVENT (MAKE-EXEC-EVENT :ID (LIST* (CL:INCF (HISTORY-INDEX HISTORY))
                                                                 ID PROMPT)))
                         (CL:PUSH EVENT (HISTORY-EVENTS HISTORY))
                         (CL:SETF (CDR (CL:NTHCDR (CL:1- (HISTORY-SIZE HISTORY))
                                                (HISTORY-EVENTS HISTORY)))
                                NIL)
                         (RETURN EVENT])

(HISTORY-ADD-TO-SPELLING-LISTS
  [LAMBDA (INPUT)                                                         (* lmm "31-Jul-86 02:22")
    (COND
       ((CDR INPUT)                                                       ; Add to the spelling list if it has a definition
        (AND (LITATOM (CAR INPUT))
             (FGETD (CAR INPUT))
             (ADDSPELL (CAR INPUT)
                    2)))
       ([AND (CL:CONSP (CAR INPUT))
             (LITATOM (CAR (CAR INPUT]                                    ; looks like a valid function
        (AND [OR (CL:FBOUNDP (CAR (CAR INPUT)))
                 (CL:SPECIAL-FORM-P (CAR (CAR INPUT]
             (ADDSPELL (CAR (CAR INPUT))
                    2)))
       ((AND (CL:SYMBOLP (CAR INPUT))
             (BOUNDP (CAR INPUT)))
        (ADDSPELL (CAR INPUT)
               3])

(HISTORY-NTH
  [LAMBDA (LST N ID)                                                      (* lmm " 6-Nov-86 01:40")
    (bind EVENT while LST do (if (<= N 0)
                                 then (RETURN))
                             (SETQ EVENT (CAR LST))
                             (CL:IF (AND (EXEC-EVENT-INPUT EVENT)
                                         (NEQ EVENT *CURRENT-EVENT*)
                                         (OR (NOT (STRINGP ID))
                                             (EQ (CADR (LISTP (EXEC-EVENT-ID EVENT)))
                                                 ID)))
                                 (if (<= (CL:DECF N)
                                         0)
                                     then (RETURN LST)))
                             (pop LST])

(PRINT-HISTORY
  [CL:LAMBDA (HISTORY EVENT-SPECS &OPTIONAL NOVALUES)                    (* lmm " 5-Nov-86 23:29")
         (PROG [HELPCLOCK (EVENTS (CL:IF EVENT-SPECS
                                         (FIND-HISTORY-EVENTS EVENT-SPECS HISTORY)
                                         (HISTORY-EVENTS HISTORY))]
               (TERPRI T)
               (for X in EVENTS do (PRINT-EVENT X NOVALUES)
                                   (FRESHLINE T)
                                   (TERPRI T))
               (TERPRI T)
               (RETURN (CL:VALUES])

(FIND-HISTORY-EVENTS
  [LAMBDA (EVENT-SPEC HISTORY)                                            ; Edited  6-Nov-87 15:22 by raf
    (PROG [(*EVENTS* (HISTORY-EVENTS HISTORY))
           (ORIGINAL-EVENT-SPEC EVENT-SPEC)
           SPEC TEM VALUE-P VAL PRED-P ALL-P (AND-SPEC (CL:MEMBER "AND" EVENT-SPEC :TEST 'STRING.EQUAL]
          (DECLARE (CL:SPECIAL *EVENTS*))                                 ; Used by DO-HISTORY-SEARCH
          [if AND-SPEC
```

```
              then (RETURN (APPEND (SETQ *EVENTS* (FIND-HISTORY-EVENTS (LDIFF EVENT-SPEC AND-SPEC)
                                                                           HISTORY))
                                   (for X in (FIND-HISTORY-EVENTS (CDR AND-SPEC)
                                                                  HISTORY)
                                      when (NOT (FMEMB X *EVENTS*)) collect X]
      LP  (CL:WHEN (EQ (CAR *EVENTS*)
                       *CURRENT-EVENT*)
              (SETQ *EVENTS* (CDR *EVENTS*)))
          [CASE-EQUALP (SETQ SPEC (CAR EVENT-SPEC))
                (ALL (SETQ ALL-P T)
                     (pop EVENT-SPEC)
                     (GO LP))
                (F [COND
                      ((SETQ TEM (CDR EVENT-SPEC))                ; Otherwise, F is not a special symbol, e.g.  user types REDO F,
                                                                  ; meaning search for F itself.
                       (SETQ EVENT-SPEC (CDR EVENT-SPEC))
                       (SETQ SPEC (CAR EVENT-SPEC]
                   (DO-HISTORY-SEARCH SPEC PRED-P VALUE-P))
                [FROM (LET ((EVENTS (FIND-HISTORY-EVENTS (CDR EVENT-SPEC)
                                                         HISTORY)))
                           (CL:WHEN (CDR EVENTS)
                                  (ERROR "from?"))
                           (RETURN (REVERSE (LDIFF *EVENTS* (CDR (CL:MEMBER (CAR EVENTS)
                                                                            *EVENTS*]
                (SUCHTHAT

                      ;; What follows SUCHTHAT is a function to be applied to the entire event;  and if true, approves that event.

                       (SETQ PRED-P T)
                       (SETQ EVENT-SPEC (CDR EVENT-SPEC))
                       (SETQ SPEC (CAR EVENT-SPEC))
                       (DO-HISTORY-SEARCH SPEC PRED-P VALUE-P))
                (= (SETQ VALUE-P T)
                   (GO LP))
                (T (COND
                      ((NOT (CL:INTEGERP SPEC))
                       (DO-HISTORY-SEARCH SPEC PRED-P VALUE-P) ; Does searching.

                       )
                      [(< SPEC 0)                                 ; count backward
                       (SETQ *EVENTS* (HISTORY-NTH *EVENTS* (- SPEC)
                                             (AND (NOT ALL-P)
                                                  *EXEC-ID*]
                      (T                                          ; absolute event number
                         (SETQ *EVENTS* (SEARCH-FOR-EVENT-NUMBER *EVENTS* HISTORY SPEC]
          [COND
             ((NULL *EVENTS*)
              (COND
                 (ALL-P (RETURN VAL)))
              (ERROR SPEC '" ?" T))
             ((NULL (SETQ EVENT-SPEC (CDR EVENT-SPEC)))
              (COND
                 [(NULL ALL-P)
                  (RETURN (LIST (CAR *EVENTS*]
                 (T (SETQ VAL (NCONC1 VAL (CAR *EVENTS*)))
                    (SETQ EVENT-SPEC ORIGINAL-EVENT-SPEC]
          (SETQ *EVENTS* (CDR *EVENTS*))
          (CL:WHEN (EQ (CAR *EVENTS*)
                       *CURRENT-EVENT*)
              (SETQ *EVENTS* (CDR *EVENTS*)))
          (SETQ VALUE-P NIL)
          (SETQ PRED-P NIL)
          (GO LP])

(PRINT-EVENT
  [CL:LAMBDA (EVENT &OPTIONAL NOVALUES)                           ; Edited  9-Mar-87 11:02 by raf
         (PROG ((INPUT (EXEC-EVENT-INPUT EVENT))
                (FILE (\GETSTREAM T 'OUTPUT))
                (POSITION (STRINGWIDTH "99/9999>" T))
                Y TEM EVENT#)
               (FRESHLINE FILE)
               (if (SETQ TEM (LISTGET (EXEC-EVENT-PROPS EVENT)
                                    '*HISTORY*))
                   then (DSPXPOSITION POSITION FILE)
                        (CL:FORMAT FILE "~{~S ~}~&" TEM))
               (PRINT-EVENT-PROMPT EVENT)
               (DSPXPOSITION (MAX POSITION (DSPXPOSITION NIL FILE))
                      T)
               (DSPFONT INPUTFONT FILE)
          LP   [COND
                  ((SETQ Y (FMEMB HISTSTR0 (LISTP INPUT)))
                   (SETQ INPUT (LDIFF INPUT Y]
               [COND
                  [(NLISTP INPUT)
                   (COND
                      ((NULL INPUT)
                       (if (EXEC-EVENT-PROPS EVENT)
                          then                                        ; don't do anything
                        else (PRIN1 "<in progress>" FILE)))
                     (T                                               ; shouldn't happen??
                        (EXEC-PRIN1 INPUT]
                 [(CDDR INPUT)                                        ; a command, just print out all elements
                  (CASE (CAR INPUT)
                      (DO-EVENTS                                      ;  special generated combination event
                         (DSPFONT DEFAULTFONT FILE)
                         (CL:FORMAT FILE "~A" (CAR INPUT))
                         (DSPFONT INPUTFONT FILE)
                         (for X in (CDR INPUT) do (FRESHLINE FILE)
                                                  (DSPXPOSITION POSITION FILE)
                                                  (CL:FORMAT FILE "   ~S" X)))
                      (T (CL:FORMAT FILE "~{~S ~}~&" INPUT)))]
                 [(CDR INPUT)                                         ; APPLY format
                  (EXEC-PRIN1 (CAR INPUT))
                  (COND
                     ((NULL (SETQ TEM (CADR INPUT)))
                      (PRIN1 ")" FILE))
                     (T (COND
                           ((NLISTP TEM)
                            (SPACES 1 FILE)))
                        (EXEC-PRIN1 TEM]
                 (T                                                   ; EVAL input
                    (EXEC-PRIN1 (CAR INPUT]
              (COND
                 (Y (SETQ INPUT (CDR Y))
                    (TERPRI FILE)
                    (DSPXPOSITION POSITION FILE)
                    (GO LP)))
         LP1 [LET [(RNT (CL:GETF (EXEC-EVENT-PROPS EVENT)
                                 '*LISPXPRINT*]
                   (if RNT
                       then (DSPFONT PRINTOUTFONT FILE)
                            (FRESHLINE FILE)
                            (MAPC RNT (FUNCTION (LAMBDA (X)
                                                   (LISPXREPRINT X FILE]
              (COND
                 ((NOT NOVALUES)
                  (DSPFONT VALUEFONT FILE)
                  (for X in (LISTGET (CDDDR EVENT)
                                     'LISPXVALUES)
                     do (FRESHLINE FILE)
                        (DSPXPOSITION POSITION FILE)
                        (EXEC-PRIN1 X])
```

(PRINT-EVENT-PROMPT

```
  [LAMBDA (EVENT)                                                     ; Edited  2-Mar-87 16:47 by raf
    (LET [(TERM (\GETSTREAM T 'OUTPUT]                                ; Crock because format interprets T to mean primary output, not
                                                                      ; terminal
         (FRESHLINE TERM)
         (if (CL:CONSP (EXEC-EVENT-ID EVENT))
             then (DSPFONT PROMPTFONT TERM)
                  (DESTRUCTURING-BIND (INDEX ID . PROMPT)
                         (EXEC-EVENT-ID EVENT)
                         (IF (CL:EQUAL ID "")
                             THEN (CL:FORMAT TERM "~D~A" INDEX PROMPT)
                           ELSE (CL:FORMAT TERM "~A/~D~A" ID INDEX PROMPT)))
           elseif LISPXHISTORY
             then (CL:FORMAT TERM "~D~A" (ENTRY# LISPXHISTORY EVENT)
                         (EXEC-EVENT-ID EVENT))
           else                                                       ; No prompt availible, use the default.
                (CL:FORMAT TERM "~A" XCL:*EXEC-PROMPT*])
```

(PROCESS-EXEC-ID

```
  (CL:LAMBDA (PROCESS &OPTIONAL ID)                                   ; Edited  5-Mar-87 17:29 by raf
         (OR (PROCESSPROP PROCESS 'ID)
             (LET ((NAME (PROCESS.NAME PROCESS)))
                  [PROCESSPROP PROCESS 'ID (OR ID (SETQ ID (COND
                                                              ((STRPOS "EXEC" NAME 1 NIL T)
                                                               (OR (SUBSTRING NAME 6 -1)
                                                                   ""))
                                                              (T
                                                                 ; under some other process
                                                                 (STRING NAME]
                  ID))))
```

(SEARCH-FOR-EVENT-NUMBER

```
  [LAMBDA (EVENTS HISTORY SPEC)                                       (* lmm "11-Sep-86 10:53")
    (while EVENTS do (if [LET [(ID (EXEC-EVENT-ID (CAR EVENTS]
                              (COND
                                 ((LISTP ID)
                                  (EQL (CAR ID)
                                       SPEC))
```

```
                                (T (EQL SPEC (ENTRY# HISTORY (CAR EVENTS]
                          then (RETURN EVENTS)
                        else (pop EVENTS])
```

(**\PICK.EVALQT**

```
  [LAMBDA NIL                                                       ; Edited 27-Feb-87 17:40 by raf

;;; Replacement for \PROC.REPEATEDLYEVALQT.  Activated by the HARDRESET at the end of LOADUP.LISP

    (INPUT T)
    (OUTPUT T)
    (TTYDISPLAYSTREAM \TopLevelTtyWindow)
    (\RESETSYSTEMSTATE)
    (EXEC :TOP-LEVEL-P T :PROFILE XCL:*PROFILE* :WINDOW (XCL::SETUP-EXEC-WINDOW \TopLevelTtyWindow])
```

(**LISPXREPRINT**

```
  [LAMBDA (X FILE)                                                  ; Edited 19-Jan-87 16:03 by bvm:
                                                                    ; takes an element from a *LISPXPRINT* property and prints it
                                                                    ; properly.
    [OR FILE (SETQ FILE (\GETSTREAM T 'OUTPUT]
    (COND
       ((STRINGP X)
        (PRIN1 X FILE))
       ((NLISTP X)
        (PRIN2 X FILE))
       ((CL:STRINGP (CAR X))
        (CL:APPLY (FUNCTION CL:FORMAT)
               FILE X))
       (T (SELECTQ (CAR X)
              ((PRINT PRIN1 PRIN2 SPACES)
                   (APPLY* (CAR X)
                          (CADR X)
                          FILE
                          (CADDDR X)))
              (TAB (TAB (CADR X)
                        (CADDR X)
                        FILE))
              (TERPRI (TERPRI FILE))
              (LISPXPRINTDEF0
                   [APPLY (CAR X)
                          (CONS (CADR X)
                                (CONS FILE (CDDDR X])
              (APPLY (CAR X)
                     (CONS (CADR X)
                           (CONS FILE (CDDDR X])
)

(DECLARE%: DONTEVAL@LOAD DOCOPY

(MOVD? 'READ 'TTYINREAD)

(MOVD '\PICK.EVALQT '\PROC.REPEATEDLYEVALQT)

(SETQ BackgroundMenu)
)
```

(DEFMACRO **CASE-EQUALP** (SELECTOR &REST CASES)

```
   [LET*
    [(KV (CL:IF (CL:SYMBOLP SELECTOR)
             SELECTOR
             (GENSYM)))
     (CLAUSES
      (for STRING-CASE in CASES
         collect (COND
                    [(FMEMB (CAR STRING-CASE)
                            '(T CL:OTHERWISE))
                     `(T ,@(CDR STRING-CASE]
                    [(NOT (CL:CONSP (CAR STRING-CASE)))
                     `([STRING.EQUAL ,KV ',(CAR STRING-CASE]
                       ,@(CDR STRING-CASE]
                    (T `([OR ,@(CL:DO ((X (CAR STRING-CASE)
                                          (CDR X))
                                       (Y NIL))
                                      ((CL:ATOM X)
                                       (REVERSE Y))
                                   (CL:PUSH `[STRING.EQUAL ,KV ',(CAR X]
                                          Y))]
                         ,@(CDR STRING-CASE]
    (CL:IF (EQ KV SELECTOR)
        `(COND
            ,@CLAUSES)
        `(LET ((,KV ,SELECTOR))
              (COND
                ,@CLAUSES)))])


(DEFMACRO EXEC-EVENT-PROPS (X)
   `(CDDDR ,X))


(CL:DEFUN EXEC-PRINT (VALUE)
   (FRESHLINE T)
   (WRITE VALUE :STREAM *TERMINAL-IO* :ESCAPE T))


(CL:DEFUN EXEC-FORMAT (FORMAT-STRING &REST ARGS)
   (AND (CL:STRINGP FORMAT-STRING)
        (LISPXPUT '*LISPXPRINT* (LIST (CONS FORMAT-STRING ARGS))
               T *CURRENT-EVENT*))
   (CL:APPLY 'CL:FORMAT (\GETSTREAM T 'OUTPUT)
          FORMAT-STRING ARGS))

(ADDTOVAR BackgroundMenuCommands
          [EXEC '(ADD-EXEC :TTY T)
                "Start a new Exec"
                (SUBITEMS ("Xerox Common Lisp" '(ADD-EXEC :PROFILE "XCL" :TTY T))
                       ("Common Lisp" '(ADD-EXEC :PROFILE "LISP" :TTY T))
                       ("Interlisp" '(ADD-EXEC :PROFILE "INTERLISP" :TTY T)
                              "Start an Interlisp Exec"
                              (SUBITEMS ("Old-Interlisp" '(ADD-EXEC :PROFILE "OLD-INTERLISP-T" :EXEC
                                                                  'EVALQT :TTY T)
                                               "Start an old-style LISPX window"])

(ADDTOVAR SYSTEMINITVARS (LISPXHISTORY NIL 0 100 100)
                         (GREETHIST))
```

;; Exec Commands

```
(DEF-DEFINE-TYPE COMMANDS "Exec Commands")

(DEFDEFINER (DEFCOMMAND [:NAME (CL:LAMBDA (WHOLE)
                                          (LET ((NAME (CL:SECOND WHOLE)))
                                               (CL:IF (CL:CONSP NAME)
                                                   (CAR NAME)
                                                   NAME)])
    COMMANDS (NAME ARGUMENTS &ENVIRONMENT ENV &BODY BODY)
   [LET ((COMMAND-LEVEL '*EXEC-COMMAND-TABLE*)
         (COMMAND-TYPE :EVAL)
         (PREFIX "exec-"))
        [if (LISTP NAME)
            then (SETQ NAME (PROG1 (CAR NAME)
                                   [for X in (CDR NAME) do (CL:ECASE X
                                                             ((:QUIET :HISTORY :INPUT :EVAL :MACRO) (SETQ
                                                                                                     COMMAND-TYPE X
                                                                                                     ))
                                                             ((:DEBUGGER :BREAK)
                                                                (SETQ COMMAND-LEVEL '*DEBUGGER-COMMAND-TABLE*)
                                                                (SETQ PREFIX "break-")))])]
        (LET* ((CMACRONAME (PACK* PREFIX NAME))
               (STRINGNAME (STRING NAME)))
              (CL:MULTIPLE-VALUE-BIND (PARSED-BODY PARSED-DECLARATIONS PARSED-DOCSTRING)
                  (PARSE-DEFMACRO ARGUMENTS '$$MACRO-FORM BODY NAME ENV :ENVIRONMENT '$$MACRO-ENV)
                `(PROGN [CL:SETF (CL:SYMBOL-FUNCTION ',CMACRONAME)
                               (FUNCTION (CL:LAMBDA ($$MACRO-FORM $$MACRO-ENV)
                                                ,@PARSED-DECLARATIONS
                                                ,PARSED-BODY]
                        (CL:SETF (CL:DOCUMENTATION ,STRINGNAME 'COMMANDS)
                               ,PARSED-DOCSTRING)
                        (PUTHASH ,STRINGNAME ',(MAKE-COMMAND-ENTRY :FUNCTION CMACRONAME :MODE COMMAND-TYPE
                                                      :ARGUMENTS (\SIMPLIFY.CL.ARGLIST ARGUMENTS))
                               ,COMMAND-LEVEL)))])


(DEFCOMMAND ("?" :QUIET) (&OPTIONAL (NAME NIL NAMEP))
   "Show forms of valid input. ? <name> shows name's documentation."
   (CL:IF NAMEP
       (PRINT-ALL-DOCUMENTATION NAME)
       [PROGN (CL:FORMAT T "~&You are typing at the Exec. Enter~&")
              (DSPFONT INPUTFONT T)
              (CL:FORMAT T "<expression>")
              (DSPFONT DEFAULTFONT T)
              (CL:FORMAT T " ~20Tto evaluate an expression~&")
              (DSPFONT INPUTFONT T)
              (CL:FORMAT T "function(arg1 arg2 ...)")
              (DSPFONT DEFAULTFONT T)
              (CL:FORMAT T " ~20Tto apply function to the arguments given~&~%%or one of:")
              (FOR X ON (REVERSE *THIS-EXEC-COMMANDS*)
                 DO (LET (COMS)
                         [MAPHASH (CAR X)
```

```
                                  #'(CL:LAMBDA (VAL KEY)
                                          (AND [NOT (SOME (CDR X)
                                                          #'(CL:LAMBDA (TAB)
                                                                    (GETHASH KEY TAB]
                                               (PUSH COMS (LIST KEY VAL]
                             (CL:MAPC #'[CL:LAMBDA (COM)
                                               (CL:FORMAT T "~&")
                                               (DSPFONT INPUTFONT T)
                                               (CL:FORMAT T "~A " (CAR COM))
                                               (DSPFONT COMMENTFONT T)
                                               (PRINT-ARGLIST (COMMAND-ENTRY-ARGUMENTS (CADR COM)))
                                               (DSPFONT DEFAULTFONT T)
                                               (LET [(DOC (CL:DOCUMENTATION (CAR COM)
                                                                 'COMMANDS]
                                                    (CL:WHEN DOC
                                                        (TAB 20 1 T)
                                                        (CL:FORMAT T "~A" DOC))]
                                  (CL:SORT COMS #'CL:STRING< :KEY #'CAR])
   (CL:VALUES))


(DEFCOMMAND ("??" :QUIET) (&REST EVENT-SPECS) "Show events specified EVENT-SPECS (or all events)"
   (IF (AND EVENT-SPECS (EQ (CAR EVENT-SPECS)
                            ':INPUT))
       THEN (PRINT-HISTORY LISPXHISTORY (CDR EVENT-SPECS)
                   T)
     ELSE (PRINT-HISTORY LISPXHISTORY EVENT-SPECS))
   (CL:VALUES))


(DEFCOMMAND ("CONN" :EVAL) (&OPTIONAL DIRECTORY) "Change default pathname to DIRECTORY"
   (/CNDIR DIRECTORY))


(DEFCOMMAND "DA" NIL "Returns current time & date"
   (DATE))


(DEFCOMMAND ("DIR" :EVAL) (&OPTIONAL PATHNAME &REST KEYWORDS) "Show directory listing for PATHNAME"
   [DODIR (CONS PATHNAME (MAPCAR KEYWORDS (FUNCTION (LAMBDA (CL:KEYWORD)
                                                       (IF (CL:SYMBOLP CL:KEYWORD)
                                                           THEN (CL:INTERN (CL:SYMBOL-NAME CL:KEYWORD)
                                                                       "INTERLISP")
                                                         ELSE CL:KEYWORD])


(DEFCOMMAND "DO-EVENTS" (&REST INPUTS &ENVIRONMENT ENV)
   "Execute the multiple events in INPUTS, using the environment ENV for all evaluations."
   [LET ((OUTER-EVENT (AND *CURRENT-EVENT* (COPY-EXEC-EVENT *CURRENT-EVENT*)))
                                                                    ; DO-EVENT smashes *CURRENT-EVENT*, so we copy and
                                                                    ; save it.

         )
        (CL:WHEN OUTER-EVENT
            (CL:SETF (EXEC-EVENT-INPUT OUTER-EVENT)
                   (CONS 'DO-EVENTS INPUTS))                        ; Each of these is fixed up below.

            )
        (ERSETQ (CL:MAPL #'[CL:LAMBDA (INPUT)
                                  (LET ([TODO (CL:IF (EQ (CAR (LISTP (CAR INPUT)))
                                                         'EVENT)
                                                     (CDR (CAR INPUT))
                                                     (LIST (CAR INPUT)))]
                                        VALUES)
                                       (CL:WHEN ADDSPELLFLG (HISTORY-ADD-TO-SPELLING-LISTS TODO))
                                       (SETQ VALUES (DO-EVENT TODO ENV))
                                                                    ; If it exists, *CURRENT-EVENT* gets smashed here.
                                       (CL:WHEN OUTER-EVENT         ; If there is an outer event...

                                           ;; Fix the outer event's list of inputs with the expanded input.

                                           (RPLACA INPUT (CAR (EXEC-EVENT-INPUT *CURRENT-EVENT*)))
                                           (CL:WHEN VALUES          ; If the last sub-event generated some values...

                                               ;; Add the new values to the outer event's values.

                                               [LET [(OLD-VALUES (CL:GETF (EXEC-EVENT-PROPS OUTER-EVENT)
                                                                        'LISPXVALUES]
                                                    (CL:IF OLD-VALUES
                                                        (NCONC OLD-VALUES VALUES)
                                                        (CL:SETF (EXEC-EVENT-PROPS OUTER-EVENT)
                                                               (LIST* 'LISPXVALUES VALUES (EXEC-EVENT-PROPS
                                                                                           OUTER-EVENT))))])))]
                         INPUTS))
        (CL:WHEN *CURRENT-EVENT*                                    ; If there was a current event...
                                                                    ; Smash saved values back from OUTER-EVENT.

            (CL:SETF (EXEC-EVENT-INPUT *CURRENT-EVENT*)
                   (EXEC-EVENT-INPUT OUTER-EVENT))
            (CL:SETF (EXEC-EVENT-ID *CURRENT-EVENT*)
                (EXEC-EVENT-ID OUTER-EVENT))
            (CL:SETF (EXEC-EVENT-VALUE *CURRENT-EVENT*)
                (EXEC-EVENT-VALUE OUTER-EVENT))
            (CL:SETF (EXEC-EVENT-PROPS *CURRENT-EVENT*)
                (EXEC-EVENT-PROPS OUTER-EVENT)))]
   (SETQ *CURRENT-EVENT* NIL)                    ; Keeps the DO-EVENT which is evaluating us from setting the
                                                 ; event's results to (the result of evaluating) the NIL we return.
                                                 ; This is alright since *CURRENT-EVENT* is already pointed to
                                                 ; by the history list.
   (CL:VALUES)                                   ; We've evaluated all the subforms directly with DO-EVENT so
                                                 ; we don't return a form to EVAL.

   )


(DEFCOMMAND ("FIX" :HISTORY) (&REST EVENT-SPEC) "Edit input for specified events"
   [APPLY 'FIX-FORM (CL:MULTIPLE-VALUE-LIST (CIRCLAR-COPYER (EVENTS-INPUT (FIND-HISTORY-EVENTS
                                                                          (OR EVENT-SPEC '(-1))
                                                                          LISPXHISTORY])


(DEFCOMMAND "FORGET" (&REST EVENT-SPEC) "Erase UNDO information (for specified events)."
   (FOR EVENT IN (FIND-HISTORY-EVENTS (OR EVENT-SPEC '(-1))
                        LISPXHISTORY)
      DO (UNDOLISPX2 EVENT T) FINALLY (CL:FORMAT T "Forgotten.~&"))
   (CL:VALUES))


(DEFCOMMAND "NAME" (COMMAND-NAME &OPTIONAL ARGUMENT-LIST &REST EVENT-SPEC)
   "NAME command-name [argument-list] [event-spec] defines new command containing the event."
   (CL:UNLESS (LISTP ARGUMENT-LIST)
       (CL:PUSH ARGUMENT-LIST EVENT-SPEC)
       (SETQ ARGUMENT-LIST NIL))
   [LET [(EVENTS (FIND-HISTORY-EVENTS EVENT-SPEC LISPXHISTORY))
         (ARGNAMES (FOR I FROM 1 AS X IN ARGUMENT-LIST COLLECT (PACK* 'ARG I]
        (CL:EVAL '(DEFCOMMAND (,COMMAND-NAME :HISTORY) ,ARGNAMES
                     [SUBPAIR ',ARGNAMES (LIST ,@ARGNAMES)
                            ',(SUBPAIR ARGUMENT-LIST ARGNAMES (EVENTS-INPUT EVENTS)
                                     T])])


(DEFCOMMAND ("NDIR" :EVAL) (&OPTIONAL PATHNAME &REST KEYWORDS)
   "Show directory listing for PATHNAME in abbreviated format"
   (DODIR (CONS PATHNAME KEYWORDS)
          '(P COLUMNS 20)
          '* ""))


(DEFCOMMAND "PL" (CL:SYMBOL) "Show property list of SYMBOL"
   (PRINTPROPS CL:SYMBOL)
   (CL:VALUES))


(DEFCOMMAND ("REDO" :HISTORY) (&REST EVENT-SPEC) "Re-execute specified event(s)"
   (EVENTS-INPUT (FIND-HISTORY-EVENTS (OR EVENT-SPEC '(-1))
                        LISPXHISTORY)))


(DEFCOMMAND ("REMEMBER" :EVAL) (&REST EVENT-SPEC) "Tell Manager to remember type-in from specified event(s)"
   (MARKASCHANGED (GETEXPRESSIONFROMEVENTSPEC EVENT-SPEC)
          'EXPRESSIONS))


(DEFCOMMAND ("SHH" :QUIET) (&REST LINE) "Execute LINE without history processing"
   (EVAL-INPUT LINE))


(DEFCOMMAND "UNDO" (&REST EVENT-SPEC)
   "Undo side effects associated with the specified event (or last undoable one)"
   [FOR EVENT IN (FIND-HISTORY-EVENTS (OR EVENT-SPEC '(-1))
                        LISPXHISTORY)
      DO (LET ((INPUT (CAR (EXEC-EVENT-INPUT EVENT)))
               (RESULT (UNDOLISPX2 EVENT)))
              (CL:IF (LISTP INPUT)
                  (SETQ INPUT (CAR INPUT)))
              (COND
                 ((NULL RESULT)
                  (CL:FORMAT T "No undo info saved for ~A.~&" INPUT))
                 ((EQ RESULT 'already)
                  (CL:FORMAT T "~A already undone.~&" INPUT))
                 (T (CL:FORMAT T "~A undone.~&" INPUT]
   (CL:VALUES))


(DEFCOMMAND ("USE" :HISTORY) (&REST LINE) "USE <new> [FOR <old>] [IN <event-spec>]"
   ;; this code stolen from LISPXUSE in HIST and edited. The structure is still pretty incomprehensible
   [PROG (EVENT-SPECS EXPR ARGS VARS (STATE 'VARS)
                LST TEM USE-ARGS GENLST)
     LP   [COND
             ([OR (NULL LST)
                  (NULL (CDR LINE))
                  (NULL (CASE-EQUALP (CAR LINE)                          ; look for one of the special keywords
                                 (FOR (COND
                                          ((EQ STATE 'VARS)
                                           (SETQ VARS (NCONC1 VARS LST))
                                           (SETQ TEM (APPEND LST TEM))
                                           (SETQ STATE 'ARGS)
                                           (SETQ LST NIL)
                                           T)))
                                 (AND (COND
                                         ((EQ STATE 'EXPR)
                                          NIL)
                                         (T [COND
                                               ((EQ STATE 'ARGS)
                                                (SETQ ARGS (NCONC1 ARGS LST)))
                                               ((EQ STATE 'VARS)    ; E.g. user types USE A AND B following previous USE
                                                                    ; command.
                                                (SETQ VARS (NCONC1 VARS LST]
                                            (SETQ STATE 'VARS)
                                            (SETQ LST NIL)
                                            T)))
                                 (IN (COND
                                        ((AND (EQ STATE 'VARS)
                                              (NULL ARGS))
                                         (SETQ VARS (NCONC1 VARS LST))
                                         (SETQ TEM (APPEND LST TEM))
                                         (SETQ STATE 'EXPR)
                                         (SETQ LST NIL)
                                         T)
                                        ((EQ STATE 'ARGS)
                                         (SETQ ARGS (NCONC1 ARGS LST))
                                         (SETQ STATE 'EXPR)
                                         (SETQ LST NIL)
                                         T]
              (SETQ LST (NCONC1 LST (COND
                                         (NIL (MEMBER (CAR LINE)
                                                      TEM)

                                              ;; This enables USE A B FOR B A, USE A FOR B AND B FOR A, or USE A FOR B AND B C
                                              ;; FOR A

                                              (LET ((TEMP (CONCAT "temp string")))
                                                   (CL:PUSH (CONS (CAR LINE)
                                                                  TEMP)
                                                          GENLST)
                                                   TEMP))
                                         (T (CAR LINE]
          (COND
             ((SETQ LINE (CDR LINE))
              (GO LP)))
          (CL:ECASE STATE
              (VARS (SETQ VARS (NCONC1 VARS LST)))
              (ARGS (SETQ ARGS (NCONC1 ARGS LST)))
              (EXPR (SETQ EXPR LST)))
          (CL:WHEN (NULL EXPR)
              (CL:IF ARGS
                  (SETQ EXPR (LIST 'F (CAAR ARGS)))
                  (SETQ EXPR '(-1))))

    ;; EXPR specifies expressions to be substituted into, e.g. USE FOO FOR FIE IN FUM or USE FOO FOR FIE. In the latter case, searches for FIE.
    ;; The F is added to avoid confusion with event numbers, etc.

    ;;

          (SETQ EXPR (MAPCAR (FIND-HISTORY-EVENTS EXPR LISPXHISTORY)
                             (FUNCTION EXEC-EVENT-INPUT)))          ; EXPR is now a list of event inputs

    ;; at this point, VARS is a list of list of old things, the extra list corresponding to the clauses of an AND, e.g.

    ;; USE A B FOR C AND D E FOR F would have

    ;; ((A B) (D E)) for VARS and

    ;; ((C) (F)) for ARGS.

          (IF (NULL ARGS)
              THEN [SETQ EXPR (FOR X IN EXPR JOIN (FOR VAR IN VARS
                                                     COLLECT (IF (CL:CONSP (CAR X))
                                                                 THEN (CONS (CONS (CAR VAR)
                                                                                  (CDAR X))
                                                                            (CDR X))
                                                               ELSE (CONS (CAR VAR)
                                                                          (CDR X]
            ELSE (WHILE ARGS DO (SETQ EXPR (LISPXUSE1 (POP VARS)
                                                      (POP ARGS)
                                                      EXPR))
                    FINALLY (COND
                               (VARS (ERROR '"use what??" "" T)))
```

```
                              [MAPC GENLST (FUNCTION (LAMBDA (X)
                                                     (LISPXSUBST (CAR X)
                                                            (CDR X)
                                                            EXPR T]
                              ;; samples:
                              ;;  USE A B C D FOR X Y means substitute A for X and B for Y and then do it again with C for X and D for Y
                              ;;  Equivalent to USE A C FOR X AND B D FOR Y
                              ;;  USE A B C FOR D AND X Y Z FOR W means 3 operations:
                              ;;   A for D and X for W in the first
                              ;;   B for D and Y for W in the second
                              ;;  C for D and Z for W in the third
                              ;; USE A B C FOR D AND X FOR Y means 3 operations:
                              ;;  A for D and X for Y in first
                              ;;  B for D and X for Y in second, etc.
                              ;; USE A B C FOR D AND X Y FOR Z causes error
                              ;;
                              ;;  USE A B FOR B A will work correctly, but USE A FOR B AND B FOR A will result in all B's being changed to A's.
                              ;;
                              ;; The general rule is substitution proceeds from left to right with each %'AND' handled separately. Whenever the
                              ;; number of variables exceeds the number of expressions available, the expressions multiply.

                              ))
          (RETURN (COND
                     [(CDR EXPR)
                      (CONS 'DO-EVENTS (for X in EXPR collect (COND
                                                                 ((CDR X)
                                                                  (CONS 'EVENT X))
                                                                 (T (CAR X]
                     (T (CAR EXPR])


(DEFCOMMAND "PP" (&OPTIONAL (NAME LASTWORD)
                        &REST TYPES) "Show TYPES (or any) definition for NAME"
   (CL:BLOCK NIL

       ;; returned from if no definitions found

       (for TYPE in [OR TYPES [TYPESOF NAME NIL NIL '? (FUNCTION (LAMBDA (TYPE)
                                                                   (NEQ (GET TYPE 'EDITDEF)
                                                                        'NILL]
                        (TYPESOF [SETQ NAME (OR (FIXSPELL NAME NIL USERWORDS NIL NIL
                                                       [FUNCTION (LAMBDA (WORD)
                                                                   (TYPESOF WORD NIL '(FIELDS FILES)
                                                                          'CURRENT]
                                                       NIL NIL NIL 'MUSTAPPROVE)
                                                (PROGN (CL:FORMAT *TERMINAL-IO* "No definitions found for ~S."
                                                              NAME)
                                                       (RETURN NIL]
                               NIL NIL '? (FUNCTION (LAMBDA (TYPE)
                                                      (NEQ (GET TYPE 'EDITDEF)
                                                           'NILL]
          do (CL:FORMAT *TERMINAL-IO* "~A definition for ~S:~%%" TYPE NAME)
             (SHOWDEF NAME TYPE)))
   (CL:VALUES))

;; Arrange to use the correct compiler

(PUTPROPS CMLEXEC FILETYPE :FAKE-COMPILE-FILE)

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA DIR)

(ADDTOVAR NLAML )

(ADDTOVAR LAMA PROCESS-EXEC-ID PRINT-EVENT PRINT-HISTORY EXEC-PRIN1 EVENTS-INPUT EVAL-INPUT EXEC-READ)
)

(PUTPROPS CMLEXEC COPYRIGHT ("Venue & Xerox Corporation" 1985 1986 1987 1988 1990 1991 1993 2021 2024))
```

{MEDLEY}<sources>CMLEXEC.;1 9-Oct-2024 02:37:44
-- Listed on 9-Oct-2024 02:43:09 --

FUNCTION INDEX

VARIABLE INDEX

COMMAND INDEX

MACRO INDEX

STRUCTURE INDEX

PROPERTY INDEX

DEFINER INDEX

SETF INDEX

DEFINE-TYPE INDEX

PROFILE INDEX