File created: 4-Jul-90 01:48:00 **{ERINYES}<MEDLEY>1.2>INTERNAL>LIBRARY>RELEASETOOLS.;1**

changes to: (FNS LIBTOOL.MAKE.FLOPPIES LIBTOOL.MAKE.FLOPPIES.AUX LIBTOOL.BREAK.DEPENDENCY LIBTOOL.FIND.FREE.FLOPPY LIBTOOL.WRITE.FILES LIBTOOL.CONFIRM.BREAK LIBTOOL.INITIALIZE.FLOPPY) (VARS RELEASETOOLSCOMS)

previous date: 8-Nov-88 19:17:17 {ERINYES}<LISP>MEDLEY>INTERNAL>LIBRARY>RELEASETOOLS.;1

Read Table: XCL

Package: INTERLISP

Format: XCCS


```
(RPAQQ RELEASETOOLSCOMS
       ((COMS ;; Making hardcopy-able directories and indexes;
              (FNS COMPDIR FLOPPYDIR FLOPPYDIRECTORY FLOPPYINDEX FLOPPYINDEXAUX))
        (COMS ;; These next functions all combine to make a tool for writing files on floppies. Call LIBTOOL.MAKE.FLOPPIES with a list of files (any
              ;; dependent files in parens), e.g. ( (TEDIT.LCOM TEDITFILE.LCOM) (BROWSER.LCOM) . . .)
              (FNS LIBTOOL.MAKE.FLOPPIES LIBTOOL.REQUEST.FLOPPY LIBTOOL.MAKE.FLOPPIES.AUX
                   LIBTOOL.BREAK.DEPENDENCY LIBTOOL.FIND.FREE.FLOPPY LIBTOOL.WRITE.FILES LIBTOOL.CONFIRM.BREAK
                   LIBTOOL.INITIALIZE.FLOPPY)
              (RECORDS FLOPPY))
        (COMS ;;  These next functions are used to compare the creation dates between two directories within a certain tolerance  (DATECOMP) is
              ;; the only one called from the exec the others are supporting functions
              (FNS DATECOMP COMPCRDA BOTHHAVE PRINT2LISTS)
              ;;  WHATVER  creates a list of the version numbers for the source and Lcoms of files in a directory that make up a composite (usually
              ;; a sysout) file
              (FNS WHATVER)
              ;; Given a list of files, return only those that are newer than a base directory's set.
              (FNS SELECT-NEWER-FILES)
              ;; Check a directory to see if any files have both DFASL and LCOM files:
              (FUNCTIONS LCOM-VS-DFASL))
        (COMS ;; Verifying a group of floppies for validity against a specified set of release directories.
              (FNS VERIFY-FLOPPIES))
        (COMS ;; Gather a unified list of where files are across several directories
              (FNS CONSOLIDATED-DIRECTORIES CONSOLIDATED-DIRECTORY-LISTING)
              ;; And based on the results of CONSOLIDATED-DIRECTORIES, move files to a single directory:
              (FNS MOVE-TESTS))
        (COMS ;; Record success and failure in AR Test-Case runs
              (COMMANDS "Pass" "Fail")
              (FUNCTIONS \\RECORD-AR-TEST-CASE-SUCCESS)
              (VARS (*AR-TEST-CASE-LOG-FILE* "{ERIS}<Test>ARs>AR-TEST-CASE.Auto-log")
                    (*AR-TEST-CASE-SUMMARY-TEMPLATE-FILE* "{ERIS}<Test>ARs>AR-Test-Case-Summary-Template.TEdit")
                    )
              ;; Report generation functions
              (FUNCTIONS AR-TEST-CASE-SUMMARY AR-TEST-CASE-READ AR-FAILING-TEST-CASES))
        (COMS ;; Patch-file creation support.
              ;; See {Eris}<Lispcore>Internal>Doc>Making-a-Patch.TEdit for details.
              (COMMANDS "PATCH" "LIBPATCH" "LOGPATCH")
              (FNS \\MAKE-PATCH-FILE \\LOG-A-PATCH))
        (PROP FILETYPE RELEASETOOLS)
        (DECLARE\: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA)
                                                                              (NLAML)
                                                                              (LAMA)))))

;; Making hardcopy-able directories and indexes;

(DEFINEQ

(COMPDIR
  (LAMBDA (DIR1 DIR2 DAYS PRINTFILENM ALLFILES)                ; Edited 13-Apr-87 10:50 by lal
                                                               ; This function  determines the differences between the two
                                                               ; specified directories.  It will check to see if all files exist on both
                                                               ; directories, and if the creation date differences are within
                                                               ; tolerances.
    (LET ((PRINTFILE (OPENSTREAM PRINTFILENM 'OUTPUT 'NEW))
          (DIR1LIST NIL)
          (DIR2LIST 'NIL)
          (ANS NIL)
          (FILEN NIL))
         (PRINTOUT PRINTFILE "Discrepencies between  " DIR1 " and " DIR2 " (run on " (DATE)
                ")." T T)                                      ; First check the creation date differences (which automatically
```

```
                                                                ; checks that the files on DIR1 are on DIR2)
        (PRINTOUT PRINTFILE "File" 20 "Author" T T)
        (|for| FILE |in| (COND
                        (ALLFILES (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR1 'NAME "*.*;")))
                        (T (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR1 'NAME "*.;"))))
           |do| (COND
                   ((EQUAL (UNPACKFILENAME.STRING FILE 'NAME)
                           "")
                    NIL)
                   (T (SETQ FILEN (COND
                                      ((EQUAL (CL:LENGTH (UNPACKFILENAME.STRING FILE 'DIRECTORY))
                                              (CL:LENGTH (UNPACKFILENAME.STRING DIR1 'DIRECTORY)))
                                       (UNPACKFILENAME.STRING FILE 'NAME))
                                      (T (CONCAT (SUBSTRING (UNPACKFILENAME.STRING FILE 'DIRECTORY)
                                                            (+ (CL:LENGTH (UNPACKFILENAME.STRING DIR1 'DIRECTORY))
                                                               2))
                                                 ">"
                                                 (UNPACKFILENAME.STRING FILE 'NAME)))))
                      (SETQ DIR2LIST (CADDR (COMPCRDA DIR1 DIR2 (COND
                                                                    (ALLFILES
                                                                     (COND
                                                                        ((EQUAL (UNPACKFILENAME.STRING
                                                                                 FILE
                                                                                 'EXTENSION)
                                                                                "")
                                                                         FILEN)
                                                                        (T (CONCAT FILEN "."
                                                                                   (UNPACKFILENAME.STRING
                                                                                    FILE
                                                                                    'EXTENSION)))))
                                                                    (T FILEN))
                                                 DAYS PRINTFILE DIR1LIST DIR2LIST))))))
                                                                ; Then we check that the files on DIR2 are on DIR1
        (|for| FILE |in| (COND
                        (ALLFILES (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR2 'NAME "*.*;")))
                        (T (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR2 'NAME "*.;"))))
           |do| (COND
                   ((EQUAL (UNPACKFILENAME.STRING FILE 'NAME)
                           "")
                    NIL)
                   (T (SETQ FILEN (COND
                                      ((EQUAL (CL:LENGTH (UNPACKFILENAME.STRING FILE 'DIRECTORY))
                                              (CL:LENGTH (UNPACKFILENAME.STRING DIR2 'DIRECTORY)))
                                       (UNPACKFILENAME.STRING FILE 'NAME))
                                      (T (CONCAT (SUBSTRING (UNPACKFILENAME.STRING FILE 'DIRECTORY)
                                                            (+ (CL:LENGTH (UNPACKFILENAME.STRING DIR2 'DIRECTORY))
                                                               2))
                                                 ">"
                                                 (UNPACKFILENAME.STRING FILE 'NAME)))))
                      (COND
                         ((CADR (BOTHHAVE DIR1 DIR2 (COND
                                                        (ALLFILES (COND
                                                                      ((EQUAL (UNPACKFILENAME.STRING FILE
                                                                                     'EXTENSION)
                                                                              "")
                                                                       FILEN)
                                                                      (T (CONCAT FILEN "." (UNPACKFILENAME.STRING
                                                                                            FILE
                                                                                            'EXTENSION)))))
                                                        (T FILEN))
                                    DIR1LIST DIR2LIST))
                          (SETQ DIR1LIST (CADR (BOTHHAVE DIR1 DIR2 (COND
                                                                       (ALLFILES
                                                                        (COND
                                                                           ((EQUAL (UNPACKFILENAME.STRING
                                                                                    FILE
                                                                                    'EXTENSION)
                                                                                   "")
                                                                            FILEN)
                                                                           (T (CONCAT FILEN "."
                                                                                      (UNPACKFILENAME.STRING
                                                                                       FILE
                                                                                       'EXTENSION)))))
                                                                       (T FILEN))
                                               DIR1LIST DIR2LIST)))))))
        (COND
           ((OR (GREATERP (LENGTH DIR1LIST)
                          0)
                (GREATERP (LENGTH DIR2LIST)
                          0))
            (PRINTOUT PRINTFILE T T "Files not in" 30 "Files not in" T DIR1 30 DIR2 T T)))
        (PRINT2LISTS DIR1LIST DIR2LIST PRINTFILE)
        (CLOSEF PRINTFILE))))

(FLOPPYDIR
  (LAMBDA (DIR)                                                 ; Edited 19-Feb-87 19:32 by jds
```

```
;; Creates a TEdit window containing a DIRECTORY listing of DIR, or by default the FLOPPY.
(LET ((LISTINGFILE (OPENSTREAM '{NODIRCORE} 'BOTH)))
     (DIRECTORY (OR DIR '{FLOPPY})
            `(OUT ,LISTINGFILE P DA))
     (COND
        ((NOT (OPENP LISTINGFILE))
         (SETQ LISTINGFILE (OPENSTREAM LISTINGFILE 'INPUT))))
     (SETFILEPTR LISTINGFILE 0)
     (TEDIT LISTINGFILE))))
```

(FLOPPYDIRECTORY

```
(LAMBDA (|ListFile|)                                    ; Edited  4-Mar-87 09:32 by shw:
    ;; Makes a TEdit file (listfile) that contains a formatted list of what is on the floppy;
    (PROG (STRM NAMES F.NAME)
          (|if| (NOT |ListFile|)
              |then| (|printout| T "PLEASE GIVE OUTPUT FILE NAME" T)
                     (RETURN NIL)
            |else| (SETQ STRM (OPENTEXTSTREAM)))
          (|until| (MOUSECONFIRM "Please insert floppy.") |do| (BLOCK))
          (SETQ NAMES (FILDIR '{FLOPPY}))
          (SETQ F.NAME (FLOPPY.NAME))
          (TEDIT.INSERT STRM (CONCAT (CHARACTER 13)
                                     "Contents of "
                                     (IF (EQ (MACHINETYPE)
                                             'DOVE)
                                         THEN "1186"
                                       ELSE "1108")
                                     " Floppy: " F.NAME (CHARACTER 13)
                                     (CHARACTER 13))
                NIL
                (FONTCREATE 'TIMESROMAN 12 'BOLD))
          (|for| X |in| NAMES |do| (TEDIT.INSERT STRM (CONCAT (PACKFILENAME 'HOST NIL 'BODY X)
                                                               (CHARACTER 13))
                                                   NIL
                                                   (FONTCREATE 'GACHA 10)))
          (TEDIT.PARALOOKS (TEXTOBJ STRM)
                '(QUAD CENTERED)
                1 12)
          (TEDIT.LOOKS (TEXTOBJ STRM)
                '(UNDERLINE ON)
                2
                (NCHARS (CONCAT "Contents of Floppy: " F.NAME)))
          (CLOSEF STRM)
          (TEDIT.PUT STRM |ListFile|)
          (RETURN |ListFile|))))
```

(FLOPPYINDEX

```
(LAMBDA (OUTFILE)                                       ; Edited 14-Jan-87 14:03 by shw:

;;; Given a series of library-package floppies, creates a directory showing where each library file resides, sorted alphabetically by library filename.  Prints
;;; the list on OUTFILE, and returns a list of ((LIBNAME FLOPPY) (LIBNAME2 FLOPPY2) ...)

    (PROG (FLOPPYCONTENTS PACKAGELOCS WASOPEN)
     ;;  Ask the person to load each floppy in turn.  Collect the DIRECTORY off each floppy.
          (SETQ FLOPPYCONTENTS (|collect| (|until| (MOUSECONFIRM (CONCAT "Load data for " (FLOPPY.NAME)
                                                                         "?"))
                                                   |do| (BLOCK))
                                          (CONS (FLOPPY.NAME)
                                                (DIRECTORY '{FLOPPY}))
                                    |repeatwhile| (MOUSECONFIRM "Click LEFT once you have loaded another floppy,
                                                                 RIGHT if you want to end."))))
          (SETQ FLOPPYCONTENTS (INTERSECTION FLOPPYCONTENTS FLOPPYCONTENTS))
                                                        ; Now make sure we have no duplicates, in case he loaded the
                                                        ; same floppy twice.
          (SETQ PACKAGELOCS (|for| FLOPPY |in| FLOPPYCONTENTS |join|
                                                        ;; Gather the root filenames for the files on eadh floppy
                                   (|bind| (FLOPPYNAME _ (CAR FLOPPY)) |for| PACKAGE
                                      |in| (CDR FLOPPY)
                                      |collect|
                                                        ;; collect the root file name for each file on the floppy together
                                                        ;; with the floppy's name.
                                           (LIST (PACKFILENAME 'VERSION NIL
                                                               'HOST NIL 'DIRECTORY
                                                               'NIL
                                                               'BODY PACKAGE)
                                                 FLOPPYNAME))))
          (|FLOPPYINDEXAUX| OUTFILE PACKAGELOCS)        ; Print the directory
          (RETURN OUTFILE))))
```

(FLOPPYINDEXAUX

```
(LAMBDA (OUTFILE PACKAGELOCS)                                   ; Edited 12-Feb-87 14:29 by shw:
    ;; Given a list of the form ((pkgName floppyName) (pkgName floppyName) etc.), print an alphabetized listing with package names on the left, and
    ;; floppy names on the right.  Prints the listing on OUTFILE, which is, if need be, opened and closed.
    (PROG (FLOPPYCONTENTS WASOPEN)
          (COND
             ((OPENP OUTFILE)
              (SETQ WASOPEN T))
             (T (SETQ OUTFILE (OPENSTREAM OUTFILE 'OUTPUT 'NEW))))
          (SETQ PACKAGELOCS (INTERSECTION PACKAGELOCS PACKAGELOCS))
          (SETQ PACKAGELOCS (SORT PACKAGELOCS '(LAMBDA (PKG1 PKG2)
                                                  (UALPHORDER (CAR PKG1)
                                                         (CAR PKG2)))))
          (|for| PKG |in| PACKAGELOCS |do| (|pushnew| FLOPPYCONTENTS (CADR PKG)))
          (|for| PKG |in| PACKAGELOCS |first| (PRINTOUT OUTFILE T (|if| (EQ (MACHINETYPE)
                                                                            'DOVE)
                                                                       |then| "1186"
                                                                      |else| "1108")
                                                  " >>NAME<< " "Module name to floppy directory" T T "Created from
                                                  floppies named:  ")
                                               (|for| FLOPPY |in| (SORT FLOPPYCONTENTS) |do| (PRINTOUT OUTFILE FLOPPY \,)
                                                  |finally| (PRINTOUT OUTFILE T T))
             |do| (PRINTOUT OUTFILE (PACKFILENAME 'VERSION NIL 'BODY (CAR PKG))
                            "       "
                            (CADR PKG)
                            T))
          (COND
             ((NOT WASOPEN)
              (CLOSEF OUTFILE))))))

)

;; These next functions all combine to make a tool for writing files on floppies. Call LIBTOOL.MAKE.FLOPPIES with a list of files (any dependent files in
;; parens), e.g. ( (TEDIT.LCOM TEDITFILE.LCOM) (BROWSER.LCOM) . . .)

(DEFINEQ

(LIBTOOL.MAKE.FLOPPIES
  (LAMBDA (FILELIST FROMDIR FLOPPYSIZE FLOPPY-NAME-BASE)          (* |edited:| "24-Oct-86 16:21")
    ;; Request a floppy.  Initialize the first floppy Call the aux function which will do the rest of the work on that group.
    (LET ((FLOPSIZE (IF (OR (EQUAL FLOPPYSIZE 8)
                            (EQUAL FLOPPYSIZE 1108))
                        THEN 2250
                      ELSE 690)))
         (SETQ FLOPPYLIST NIL)                                    ; Start with no known floppies.
         ;; Ask for the first floppy, and format it, if needed.
         (LIBTOOL.INITIALIZE.FLOPPY FLOPPYSIZE FLOPPY-NAME-BASE)
         (|for| GROUP |in| FILELIST |do| (LIBTOOL.MAKE.FLOPPIES.AUX GROUP FROMDIR FLOPSIZE FLOPPY-NAME-BASE)))))

(LIBTOOL.REQUEST.FLOPPY
  (LAMBDA (FLOPPYNUM)                                             (* |edited:| "27-Oct-86 14:15")
    (RINGBELLS)
    (PRINTOUT T "Please insert " FLOPPYNUM T "Is the requested floppy ready? ")
    (LET ((ANSWER (READ)))
         (|if| (NOT (OR (EQUAL ANSWER 'Y)
                        (EQUAL ANSWER '\y)))
             |then| (LIBTOOL.REQUEST.FLOPPY FLOPPYNUM)
            |else| (PRINTOUT T "Floppy accepted." T)))))

(LIBTOOL.MAKE.FLOPPIES.AUX
  (LAMBDA (GROUP FROMDIR FLOPSIZE FLOPPY-NAME-BASE)               ; Edited 30-Dec-86 10:09 by Wessling
    ;; Given a list of files to be written together on one floppy (or a single symbol that's a single file), find space and write the files.  If there's not room
    ;; for all the files on a single floppy, split the list (and let the user confirm that he wants it busted as we say).
    (SETQ GROUP (MKLIST GROUP))                                   ; Single file -> List of file(s).
    (LET ((FILELENGTH (|for| FILE |in| GROUP |sum| (GETFILEINFO (PACK* FROMDIR FILE)
                                                          'SIZE))))
         ;; If the file length is greater than 687, then the group of files is going to have to be broken.  Call LIBTOOL.BREAK.DEPENDENCY to split
         ;; them up.
         (|if| (GREATERP (PLUS FILELENGTH (TIMES 2 (LENGTH GROUP)))
                         FLOPSIZE)
             |then| (LIBTOOL.BREAK.DEPENDENCY (REMOVE (CAR (LAST GROUP))
                                                      GROUP)
                                              (LAST GROUP)
                                              (GETFILEINFO (PACK* FROMDIR (CAR (LAST GROUP)))
                                                           'SIZE)
                                              FROMDIR FLOPSIZE FLOPPY-NAME-BASE)
            |else| (|if| (LESSP FILELENGTH (|fetch| FREEPAGES |of| CURRENTFLOPPY))
                       |then| (LIBTOOL.WRITE.FILES (|if| (LISTP GROUP)
                                                       |then| GROUP
                                                      |else| (LIST GROUP))
                                                   FROMDIR FLOPSIZE)
```

```
                |else| (LIBTOOL.FIND.FREE.FLOPPY FILELENGTH GROUP FLOPSIZE FLOPPY-NAME-BASE)
                     (LIBTOOL.WRITE.FILES (|if| (LISTP GROUP)
                                                |then| GROUP
                                               |else| (LIST GROUP))
                                          FROMDIR FLOPSIZE))))))
```

(LIBTOOL.BREAK.DEPENDENCY

```
(LAMBDA (GROUP LASTFILE LENGTHLASTFILE FROMDIR FLOPSIZE FLOPPY-NAME-BASE)
                                                                    (* |edited:| "24-Oct-86 17:15")
```

;; Find a combo of files that will fit on one floppy. Return 2 lists: that combo and the rest.

```
(|if| (GREATERP (PLUS LENGTHLASTFILE (GETFILEINFO (PACK* FROMDIR (CAR (LAST GROUP)))
                                                 'SIZE))
          FLOPSIZE)
     |then| (LIBTOOL.CONFIRM.BREAK GROUP LASTFILE FROMDIR FLOPSIZE FLOPPY-NAME-BASE)
    |else| (LIBTOOL.BREAK.DEPENDENCY (REMOVE (CAR (LAST GROUP))
                                            GROUP)
                (APPEND (LAST GROUP)
                        LASTFILE)
                (PLUS LENGTHLASTFILE (GETFILEINFO (PACK* FROMDIR (CAR (LAST GROUP)))
                                                  'SIZE))
                FROMDIR FLOPSIZE FLOPPY-NAME-BASE))))
```

(LIBTOOL.FIND.FREE.FLOPPY

```
(LAMBDA (PAGESNEEDED GROUP FLOPSIZE FLOPPY-NAME-BASE)          (* |edited:| "27-Oct-86 15:24")
```

;; Hunt for a floppy that has enough space for the files we want to write (PAGESNEEDED).

;; If the floppylist is empty, create a new floppy record and request a new floppy.

```
(LET ((NEWFLOP (|for| FLOPPY |in| FLOPPYLIST |thereis| (LESSP PAGESNEEDED (|fetch| FREEPAGES |of| FLOPPY)))))
     (|if| (NULL NEWFLOP)
         |then| (LIBTOOL.INITIALIZE.FLOPPY FLOPSIZE FLOPPY-NAME-BASE)
       |else| (LIBTOOL.REQUEST.FLOPPY (|fetch| NAME |of| NEWFLOP))
            (SETQ CURRENTFLOPPY NEWFLOP)))                      ; If it finds a floppy that has enough room, it should return back
                                                                ; with that floppy it and resume.
(FOR X INFILES "{FLOPPY}*" COLLECT X)))
```

(LIBTOOL.WRITE.FILES

```
(LAMBDA (GROUP FROMDIR FLOPSIZE)                               (* |edited:| "24-Oct-86 16:34")
```

;; Write the files onto the floppy.

```
(|for| FILE |in| GROUP |do| (PRINTOUT T "Copying " FILE " to " (|fetch| NAME |of| CURRENTFLOPPY)
                                   "..." T)
                           (COPYFILE (PACK* FROMDIR FILE)
                                     (PACK* '{FLOPPY} FILE)))
(PRINTOUT T " ...done." T)
```

;; Update the prop list of that floppy to know that there are fewer pages:

```
(|replace| FREEPAGES |of| CURRENTFLOPPY |with| (FLOPPY.FREE.PAGES))))
```

(LIBTOOL.CONFIRM.BREAK

```
(LAMBDA (GROUP1 GROUP2 FROMDIR FLOPSIZE FLOPPY-NAME-BASE)      (* |edited:| "27-Oct-86 15:40")
```

;; Let the user confirm a suggested breakdown of a group of files that won't fit on a single floppy.

```
(PRINTOUT T "Group must be broken. Breakup will be:" T T GROUP1 T T GROUP2 T T)
(PRINTOUT T "Please confirm: ")
(LET ((ANSWER (READ)))
     (|if| (OR (EQUAL ANSWER 'Y)
               (EQUAL ANSWER '\y))
         |then| (|for| GROUP |in| (LIST GROUP1 GROUP2) |do| (LIBTOOL.MAKE.FLOPPIES.AUX GROUP FROMDIR FLOPSIZE
                                                                FLOPPY-NAME-BASE))
       |else| (PRINTOUT T "Please enter your split, typing the two groups as lists:" T)
              (PRINTOUT T "Please type first group: ")
              (LET ((SPLIT1 (READ)))
                   (PRINTOUT T "Please type second group: ")
                   (LET ((SPLIT2 (READ)))
                        (|for| GROUP |in| (LIST SPLIT1 SPLIT2) |do| (LIBTOOL.MAKE.FLOPPIES.AUX GROUP FROMDIR
                                                                       FLOPSIZE FLOPPY-NAME-BASE))))))))
```

(LIBTOOL.INITIALIZE.FLOPPY

```
(LAMBDA (FLOPSIZE FLOPPY-NAME-BASE)                            (* |edited:| "24-Oct-86 17:12")
```

;; Ask the user to insert a NEW floppy (not one that we've written on before). Assign it the next number n the series, and format it, if he specified a ;; name.

```
(LET ((NEWFLOPPY (CONCAT (OR FLOPPY-NAME-BASE 'FLOPPY)
                         " #"
                         (ADD1 (LENGTH FLOPPYLIST)))))
```

;; Get the new floppy inserted:

```
(LIBTOOL.REQUEST.FLOPPY NEWFLOPPY)
```

;; Now format it, if he gave us a name (if not, assume he formatted the floppies himself):

```
(AND FLOPPY-NAME-BASE (FLOPPY.FORMAT NEWFLOPPY NIL T))
```

```
          ;; Create the description for this new floppy, and add it to the list of known floppies:
          (SETQ CURRENTFLOPPY (|create| FLOPPY
                                        NAME _ NEWFLOPPY
                                        FREEPAGES _ FLOPSIZE))
          (SETQ FLOPPYLIST (NCONC1 FLOPPYLIST CURRENTFLOPPY)))))

)

(DECLARE\: EVAL@COMPILE

(RECORD FLOPPY (NAME FREEPAGES))
)

;;  These next functions are used to compare the creation dates between two directories within a certain tolerance  (DATECOMP) is the only one called
;; from the exec the others are supporting functions

(DEFINEQ
```

(DATECOMP

```
  (LAMBDA (DIR1 DIR2 DAYS PRINTFILENM ALLFILES)                    ; Edited 13-Apr-87 10:50 by lal
                                                                   ; This function  determines the differences between the two
                                                                   ; specified directories.  It will check to see if all files exist on both
                                                                   ; directories, and if the creation date differences are within
                                                                   ; tolerances.
    (LET
     ((PRINTFILE (OPENSTREAM PRINTFILENM 'OUTPUT 'NEW))
      (DIR1LIST NIL)
      (DIR2LIST 'NIL)
      (ANS NIL)
      (FILEN NIL))
     (PRINTOUT PRINTFILE "Discrepencies between  " DIR1 " and " DIR2 " (run on " (DATE)
           ")." T T)                                               ; First check the creation date differences (which automatically
                                                                   ; checks that the files on DIR1 are on DIR2)
     (PRINTOUT PRINTFILE "File" 20 "Author" T T)
     (|for| FILE |in| (|if| ALLFILES
                         |then| (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR1 'NAME "*.*;"))
                        |else| (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR1 'NAME "*.;")))
        |do| (|if| (EQUAL (UNPACKFILENAME.STRING FILE 'NAME)
                          "")
                   |then| NIL
                  |else| (SETQ FILEN (|if| (EQUAL (CL:LENGTH (UNPACKFILENAME.STRING FILE 'DIRECTORY))
                                                  (CL:LENGTH (UNPACKFILENAME.STRING DIR1 'DIRECTORY)))
                                           |then| (UNPACKFILENAME.STRING FILE 'NAME)
                                          |else| (CONCAT (SUBSTRING (UNPACKFILENAME.STRING FILE 'DIRECTORY)
                                                                   (+ (CL:LENGTH (UNPACKFILENAME.STRING DIR1 'DIRECTORY))
                                                                      2))
                                                         ">"
                                                         (UNPACKFILENAME.STRING FILE 'NAME))))
                        (SETQ DIR2LIST (CADDR (COMPCRDA DIR1 DIR2
                                                        (|if| ALLFILES
                                                            |then| (|if| (EQUAL (UNPACKFILENAME.STRING FILE 'EXTENSION)
                                                                                "")
                                                                       |then| FILEN
                                                                      |else| (CONCAT FILEN "." (UNPACKFILENAME.STRING
                                                                                                  FILE
                                                                                                  'EXTENSION)))
                                                           |else| FILEN)
                                                        DAYS PRINTFILE DIR1LIST DIR2LIST)))))
                                                                   ; Then we check that the files on DIR2 are on DIR1
     (|for| FILE |in| (|if| ALLFILES
                         |then| (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR2 'NAME "*.*;"))
                        |else| (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR2 'NAME "*.;")))
        |do| (|if| (EQUAL (UNPACKFILENAME.STRING FILE 'NAME)
                          "")
                   |then| NIL
                  |else| (SETQ FILEN (|if| (EQUAL (CL:LENGTH (UNPACKFILENAME.STRING FILE 'DIRECTORY))
                                                  (CL:LENGTH (UNPACKFILENAME.STRING DIR2 'DIRECTORY)))
                                           |then| (UNPACKFILENAME.STRING FILE 'NAME)
                                          |else| (CONCAT (SUBSTRING (UNPACKFILENAME.STRING FILE 'DIRECTORY)
                                                                   (+ (CL:LENGTH (UNPACKFILENAME.STRING DIR2 'DIRECTORY))
                                                                      2))
                                                         ">"
                                                         (UNPACKFILENAME.STRING FILE 'NAME))))
                        (|if| (CADR (BOTHHAVE DIR1 DIR2 (|if| ALLFILES
                                                             |then| (|if| (EQUAL (UNPACKFILENAME.STRING FILE 'EXTENSION)
                                                                                 "")
                                                                        |then| FILEN
                                                                       |else| (CONCAT FILEN "." (UNPACKFILENAME.STRING
                                                                                                   FILE
                                                                                                   'EXTENSION)))
                                                            |else| FILEN)
                                              DIR1LIST DIR2LIST))
                              |then| (SETQ DIR1LIST (CADR (BOTHHAVE
                                                          DIR1 DIR2
                                                          (|if| ALLFILES
                                                              |then| (|if| (EQUAL (UNPACKFILENAME.STRING FILE 'EXTENSION)
```

```
                                                                    "")
                                                              |then| FILEN
                                                             |else| (CONCAT FILEN "." (UNPACKFILENAME.STRING
                                                                                      FILE
                                                                                      'EXTENSION)))
                                                       |else| FILEN)
                                                    DIR1LIST DIR2LIST))))))
     (|if| (OR (GREATERP (LENGTH DIR1LIST)
                         0)
               (GREATERP (LENGTH DIR2LIST)
                         0))
         |then| (PRINTOUT PRINTFILE T T "Files not in" 30 "Files not in" T DIR1 30 DIR2 T T))
     (PRINT2LISTS DIR1LIST DIR2LIST PRINTFILE)
     (CLOSEF PRINTFILE))))
```

(COMPCRDA

```
  (LAMBDA (DIR1 DIR2 DCFILE DAYS PRINTFILE DIR1LIST DIR2LIST)    ; Edited  5-Feb-87 17:39 by lal
                                                                 ; This function finds the difference between the creation dates of
                                                                 ; the given file in both directories
    (COND
       ((CAR (BOTHHAVE DIR1 DIR2 DCFILE DIR1LIST DIR2LIST))
        (COND
           ((IGREATERP (ABS (DIFFERENCE (GETFILEINFO (PACKFILENAME.STRING 'DIRECTORY DIR1 'NAME DCFILE)
                                                     'ICREATIONDATE)
                                        (GETFILEINFO (PACKFILENAME.STRING 'DIRECTORY DIR2 'NAME DCFILE)
                                                     'ICREATIONDATE)))
                       (TIMES DAYS 24 60 60))
            (PRINTOUT PRINTFILE DCFILE 20 (GETFILEINFO (PACKFILENAME.STRING 'DIRECTORY DIR2 'NAME DCFILE)
                                                       'AUTHOR)
                      T)                                         ; Testing to see if the difference is greater than the specified
                                                                 ; number of days
            (LIST NIL DIR1LIST DIR2LIST))
           (T (LIST NIL DIR1LIST DIR2LIST))))
       (T (BOTHHAVE DIR1 DIR2 DCFILE DIR1LIST DIR2LIST)))))
```

(BOTHHAVE

```
  (LAMBDA (DIR1 DIR2 FILENM DIR1LIST DIR2LIST)                   ; Edited 10-Apr-87 11:51 by lal
                                                                 ; This function checks to see if both directories have the
                                                                 ; specified file on them and return them in an appropriate list.
    (COND
       ((FINDFILE FILENM T (LIST DIR1))
        (COND
           ((FINDFILE FILENM T (LIST DIR2))
            (LIST T NIL NIL))
           (T (LIST NIL NIL (APPEND DIR2LIST (LIST FILENM))))))
       (T (LIST NIL (APPEND DIR1LIST (LIST FILENM)))))))
```

(PRINT2LISTS

```
  (LAMBDA (FIRST SECOND PRINTFILE)                               ; Edited  5-Feb-87 11:11 by lal
                                                                 ; This function will print the elements of two lists in tow columns.
    (LET ((NUM (MAX (LENGTH FIRST)
                    (LENGTH SECOND))))
         (|for| X |from| 1 |to| NUM |do| (|if| (CAR FIRST)
                                             |then| (PRINTOUT PRINTFILE (CAR FIRST))
                                                                 ; print an element for the 1st column
                                                    (SETQ FIRST (CDR FIRST)))
                                          (|if| (CAR SECOND)
                                             |then| (PRINTOUT PRINTFILE 30 (CAR SECOND)
                                                              T)  ;  print an element for the 2nd column
                                                    (SETQ SECOND (CDR SECOND))
                                              |else| (PRINTOUT PRINTFILE T))))))

)

;;  WHATVER  creates a list of the version numbers for the source and Lcoms of files in a directory that make up a composite (usually a sysout) file

(DEFINEQ
```

(WHATVER

```
  (LAMBDA (DIR1 COMPOUND PRINTFILENM)                            ; Edited 10-Apr-87 09:56 by lal
                                                                 ; This function exists to record the versions of all the source and
                                                                 ; compiled  files that make up a compound file for later
                                                                 ; comparison
    (LET ((OLDFILE NIL)
          (PRINTFILE (OPENSTREAM PRINTFILENM 'OUTPUT 'NEW))
          (COMPILE-EXTEN '("LCOM" "DFASL")))
         (PRINTOUT PRINTFILE "This program will find the version numbers for files that make up a particular
                sysout or patch. (run on " (DATE)
                ")." T T)                                        ; printing out headings
         (PRINTOUT PRINTFILE "The compound file is " (CAR (DIRECTORY COMPOUND))
                " created on "
                (GETFILEINFO COMPOUND 'CREATIONDATE)
                T)
         (PRINTOUT PRINTFILE T "FILE" 27 "SOURCE" 36 "COMPILED" T 27 "VERSION" 37 "VERSION")
```

```
                                                                    ; print each unique filename
        (|for| FILE |in| (SORT (DIRECTORY (PACKFILENAME.STRING 'DIRECTORY DIR1 'NAME "*.*;")))
           |do|                                                     ; only process the files that are either source or Lcoms
                (|if| (OR (EQUAL (UNPACKFILENAME.STRING FILE 'EXTENSION)
                                 "")
                          (MEMBER (UNPACKFILENAME.STRING FILE 'EXTENSION)
                                  COMPILE-EXTEN))
                      |then|                                        ; Is this a different file ?
                           (|if| (OR (NOT (EQUAL (UNPACKFILENAME.STRING FILE 'NAME)
                                                 (UNPACKFILENAME.STRING OLDFILE 'NAME)))
                                     (NULL OLDFILE))
                                 |then| (|if| (AND OLDFILE (NOT (MEMBER (UNPACKFILENAME.STRING OLDFILE 'EXTENSION)
                                                                        COMPILE-EXTEN)))
                                              |then|                ; if we found the next file but there was no Lcom for the previous
                                                                    ; one, print out a dash in the Lcom column for the last file
                                                    (PRINTOUT PRINTFILE 40 "-"))
                                                                    ;  print out next file name
                                            (PRINTOUT PRINTFILE T (UNPACKFILENAME.STRING FILE 'NAME)))
                           (|if| (EQUAL (UNPACKFILENAME.STRING FILE 'EXTENSION)
                                        "")
                                 |then|                             ; if it is a source file print its version number in the proper column
                                      (PRINTOUT PRINTFILE 30 (UNPACKFILENAME.STRING FILE 'VERSION)))
                           (|if| (MEMBER (UNPACKFILENAME.STRING FILE 'EXTENSION)
                                         COMPILE-EXTEN)
                                 |then|                             ; if we found the Lcom but there was no source, print a dash in
                                                                    ; the source column
                                      (|if| (NOT (EQUAL (UNPACKFILENAME.STRING OLDFILE 'EXTENSION)
                                                        ""))
                                            |then| (PRINTOUT PRINTFILE 30 "-"))
                                                                    ; print the compiled file version number in its proper column
                                      (PRINTOUT PRINTFILE 40 (UNPACKFILENAME.STRING FILE 'VERSION)))
                           (SETQ OLDFILE FILE)))
        (CLOSEF PRINTFILE))))

)
```

;; Given a list of files, return only those that are newer than a base directory's set.

```
(DEFINEQ

(SELECT-NEWER-FILES
  (LAMBDA (FILE-LIST BASE-DIRECTORY)                                ; Edited 21-Jan-88 09:46 by jds
    (PRINTOUT T T "- - - - Gathering newer files - - - -" T T)
    (|bind| OTHERFILE |for| FILE |in| FILE-LIST |when| (COND
                                                          ((NOT (INFILEP (SETQ OTHERFILE (PACKFILENAME 'VERSION NIL
                                                                                               'DIRECTORY
                                                                                               BASE-DIRECTORY
                                                                                               'BODY FILE))))
                                                                    ; New file.
                                                           (PRINTOUT T "   NEW file:  " FILE T)
                                                           FILE)
                                                          ((> (GETFILEINFO FILE 'ICREATIONDATE)
                                                              (GETFILEINFO OTHERFILE 'ICREATIONDATE))
                                                                    ; This file is newer.
                                                           (PRINTOUT T "   Newer file collected:  " FILE T)
                                                           FILE)
                                                          (T (PRINTOUT T "   Not collected:  " FILE T)
                                                             NIL))
      |collect| (LIST FILE (GETFILEINFO FILE 'CREATIONDATE)))))

)
```

;; Check a directory to see if any files have both DFASL and LCOM files:

```
(CL:DEFUN LCOM-VS-DFASL (DIRECTORY &OPTIONAL DRIBBLE?)
   (LET ((FILES (DIRECTORY DIRECTORY))
         LCOM DFASL)
        (SETQ FILES (|for| FILE |in| FILES |collect| (PACKFILENAME 'EXTENSION NIL 'VERSION NIL 'BODY FILE)))
        (SETQ FILES (INTERSECTION FILES FILES))
        (COND
           (DRIBBLE? (DRIBBLE '{CORE}LCOM-VS-DFASL-CHECKOUT)))
        (COND
           (FILES (PRINTOUT T T T "- - - Checking " DIRECTORY " for DFASL/LCOM conflict - - -" T T))
           (T (PRINTOUT T T T "- - - " DIRECTORY " has no DFASL/LCOM conflicts - - -" T T)))
        (|for| FILE |in| FILES |when| (AND (INFILEP (SETQ LCOM (PACKFILENAME 'EXTENSION "LCOM" 'BODY FILE)))
                                           (INFILEP (SETQ DFASL (PACKFILENAME 'EXTENSION "DFASL" 'BODY FILE))))
           |do| (PRINTOUT T "   " FILE " [" (GETFILEINFO FILE 'AUTHOR)
                      "] has both LCOM & DFASL ")
                (COND
                   ((>= (GETFILEINFO LCOM 'ICREATIONDATE)
                        (GETFILEINFO DFASL 'ICREATIONDATE))
                    (PRINTOUT T "(the LCOM is newer)." T))
                   (T (PRINTOUT T "(the DFASL is newer)." T))))
        (COND
           (DRIBBLE? (DRIBBLE NIL)
```

```
                  (SEND.FILE.TO.PRINTER '{CORE}LCOM-VS-DFASL-CHECKOUT)
                  (DELFILE '{CORE}LCOM-VS-DFASL-CHECKOUT)))))
```

;; Verifying a group of floppies for validity against a specified set of release directories.

```
(DEFINEQ
```

(VERIFY-FLOPPIES

```
  (LAMBDA (RELEASE-DIRECTORY-LIST)                          ; Edited 19-Feb-88 14:35 by jds
```

;; This function will repeatedly ask for floppy disks, and for each will

;; -- Verify that it can read the directory

;; -- Pick a file at random and compare it with the corresponding file in the release directories.

;; RELEASE-DIRECTORY-LIST is a single directory name or a list of directories where the originals of the files on each floppy reside. These directories are searched (in order) to find equivalent files for comparison.

```
    (LET ((DIR-LIST (OR (LISTP RELEASE-DIRECTORY-LIST)
                        (LIST RELEASE-DIRECTORY-LIST)))
          FLOPPY-DIR TEST-FILE EQUIVALENT-FILE)
         (DRIBBLE '{LPT}FLOPPY-VERIFICATION-RESULTS)
         (|for| FLOPPY# |from| 1 |while| (MOUSECONFIRM "Click LEFT when next floppy is in drive.")
            |do| (PRINTOUT T T T "- - - - - - - - - - - Verifying Floppy #" FLOPPY# " - - - - - - - - - - -" T T)
                 (SETQ FLOPPY-DIR (DIRECTORY "{FLOPPY}"))
                 (PRINTOUT T "     * Directory read successfully" T)
                 (SETQ TEST-FILE (CL:NTH (RAND 0 (CL:1- (LENGTH FLOPPY-DIR)))
                                         FLOPPY-DIR))
                 (SETQ EQUIVALENT-FILE (FINDFILE (PACKFILENAME.STRING 'VERSION NIL 'HOST NIL 'DIRECTORY NIL
                                                                      'BODY TEST-FILE)
                                                 NIL RELEASE-DIRECTORY-LIST))
                 (COND
                    (EQUIVALENT-FILE (PRINTOUT T "     * Comparing " TEST-FILE " vs. " EQUIVALENT-FILE " ... ")
                           (COMPAREFILES TEST-FILE EQUIVALENT-FILE)
                           (PRINTOUT T "done." T))
                    (T (PRINTOUT T "     ****NO EQUIVALENT FOUND FOR " TEST-FILE T)))))))
```

)

;; Gather a unified list of where files are across several directories

```
(DEFINEQ
```

(CONSOLIDATED-DIRECTORIES

```
  (LAMBDA (DIR-LIST)                                        ; Edited 12-Feb-88 13:34 by jds
```

;; Create a consolidated listing of all occurrances of a given file across several directories.

```
    (LET ((FULL-DIR-NAMES (|for| DIR |in| DIR-LIST |collect| (COND
                                                               ((OR (UNPACKFILENAME DIR 'NAME)
                                                                    (UNPACKFILENAME DIR 'EXTENSION)
                                                                    (UNPACKFILENAME DIR 'VERSION))
                                                                 ; A name of some sort got specified.  Use his pattern.
                                                                DIR)
                                                               (T ; Default the pattern to *.*;.
                                                                  (PACKFILENAME.STRING 'DIRECTORY DIR 'NAME "*.*;")))
                                                             ))
          FILE-NAME FILE-LIST FILE-NAMES FILE-PLIST FILE-INFO)
         (|for| DIR |in| FULL-DIR-NAMES |do| (|for| FILE |in| (DIRECTORY DIR)
                                                |do| (SETQ FILE-NAME (PACKFILENAME 'HOST NIL 'DIRECTORY NIL
                                                                                  'VERSION NIL 'BODY FILE))
                                                     (CL:PUSH (LIST FILE (GETFILEINFO FILE 'CREATIONDATE))
                                                            (CL:GETF FILE-LIST FILE-NAME))
                                                     (CL:PUSHNEW FILE-NAME FILE-NAMES)))
         (SETQ FILE-NAMES (SORT FILE-NAMES))
         (|for| NAME |in| FILE-NAMES |collect| (CONS NAME (CL:GETF FILE-LIST NAME))))))
```

(CONSOLIDATED-DIRECTORY-LISTING

```
  (LAMBDA (DIR-LIST PRINT-FILE)                             ; Edited 11-Feb-88 17:19 by jds
    (LET ((SORTED-LIST (CONSOLIDATED-DIRECTORIES DIR-LIST))
          (BOLDFONT (FONTCREATE 'OPTIMA 10 'BOLD NIL 'INTERPRESS))
          (SMALLFONT (FONTCREATE 'OPTIMA 7 NIL NIL 'INTERPRESS)))
         (CL:WITH-OPEN-STREAM (PRINT (OPENIMAGESTREAM (OR PRINT-FILE "{LPT}")
                                                      'INTERPRESS))
                (FOR FILE-INFO IN SORTED-LIST DO (DSPFONT BOLDFONT PRINT)
                                                 (PRINTOUT PRINT T (CAR FILE-INFO)
                                                        T)
                                                 (DSPFONT SMALLFONT PRINT)
                                                 (FOR FILE IN (CDR FILE-INFO) DO (PRINTOUT PRINT 10
                                                                                           (CAR FILE)
                                                                                           " ("
                                                                                           (CADR FILE)
                                                                                           ")" T)))))))
```

)

;; And based on the results of CONSOLIDATED-DIRECTORIES, move files to a single directory:

```
(DEFINEQ

(MOVE-TESTS
  (LAMBDA (TEST-LIST TEST-DIR)                                    ; Edited 15-Feb-88 14:18 by jds
    ;; Move tests onto TEST-DIR.
    (LET (FILE-NAME FILE-INSTANCES)
         (|for| FILE |in| TEST-LIST |do| (DESTRUCTURING-BIND (FILE-NAME . FILE-INSTANCES)
                                                FILE
                                                (COND
                                                   ((> (LENGTH FILE-INSTANCES 1))
                                                    ;; More than one.  Sort them downward by creation date.
                                                    (SETQ FILE-INSTANCES (CL:SORT FILE-INSTANCES #'> :KEY
                                                                                 '(LAMBDA (FILE-INSTANCE)
                                                                                    (IDATE (CADR FILE-INSTANCE)))))
                                                    (CL:PUSH FILE-INSTANCES MULTIPLE-INSTANCES)))
                                                (DESTRUCTURING-BIND (NAME CRDATE)
                                                       (CAR FILE-INSTANCES)
                                                       (PRINTOUT T "Copying " NAME " to " (PACKFILENAME 'DIRECTORY
                                                                                                 TEST-DIR
                                                                                                 'BODY FILE-NAME)
                                                             "...")
                                                       (COPYFILE NAME (PACKFILENAME 'DIRECTORY TEST-DIR 'BODY
                                                                                    FILE-NAME))
                                                       (PRINTOUT T "Done." T)))))))

)

;; Record success and failure in AR Test-Case runs


(DEFCOMMAND "Pass" (&REST AR-NUMBERS) "Log successful AR Test-case runs for the AR numbers given."
   (FOR AR IN AR-NUMBERS DO (\\RECORD-AR-TEST-CASE-SUCCESS AR T)))


(DEFCOMMAND "Fail" (&REST AR-NUMBERS) "Log UN-successful AR Test-case runs for the AR numbers given."
   (FOR AR IN AR-NUMBERS DO (\\RECORD-AR-TEST-CASE-SUCCESS AR NIL)))


(CL:DEFUN \\RECORD-AR-TEST-CASE-SUCCESS (AR-NUMBER PASSED?)
   ;; Record whether the Test case for AR-NUMBER ran OK or not.  If PASSED? is non-NIL, the test case ran OK.
   ;; --Used by the "Pass" and "Fail" commands--
   (COND
      ((NUMBERP AR-NUMBER)
       ;; It was a number. so log it.
       (LET ((*READTABLE* FILERDTBL))
            (CL:WITH-OPEN-STREAM (LOG (OPENSTREAM *AR-TEST-CASE-LOG-FILE* 'APPEND 'OLD))
                   (PRINT (LIST AR-NUMBER (COND
                                             (PASSED? (CL:FORMAT T "Recording success for AR ~d." AR-NUMBER)
                                                    :PASS)
                                             (T (CL:FORMAT T "Recording failure for AR ~d." AR-NUMBER)
                                                :FAIL))
                                (USERNAME)
                                (DATE))
                          LOG))))
      (T ;; That wasn't an AR number.  Complain about it.
         (CL:WARN "Not an AR Number:  ~A~%" AR-NUMBER))))

(RPAQ *AR-TEST-CASE-LOG-FILE* "{ERIS}<Test>ARs>AR-TEST-CASE.Auto-log")

(RPAQ *AR-TEST-CASE-SUMMARY-TEMPLATE-FILE* "{ERIS}<Test>ARs>AR-Test-Case-Summary-Template.TEdit")

;; Report generation functions


(CL:DEFUN AR-TEST-CASE-SUMMARY (&OPTIONAL PRINT-FILE)
   ;; Creates a summary showing AR Test-Case status by AR# for all AR test cases ever run.
   (LET (LOWEST-SEEN-AR HIGHEST-SEEN-AR AR-TABLE LOG-ENTRY REPORT-FORM)
        (CL:MULTIPLE-VALUE-SETQ (LOWEST-SEEN-AR HIGHEST-SEEN-AR AR-TABLE)
              (AR-TEST-CASE-READ 5))
        (CL:WITH-OPEN-STREAM (REPORT (OPENSTREAM '{NODIRCORE} 'BOTH 'NEW))
               (FOR AR# FROM LOWEST-SEEN-AR TO HIGHEST-SEEN-AR
                  DO (SETQ LOG-ENTRY (CL:AREF AR-TABLE AR#))
                      (COND
                         (LOG-ENTRY                                ; The test got run sometime.
                                (DESTRUCTURING-BIND (AR# PASS/FAIL TESTER TIME-STAMP)
                                       LOG-ENTRY
                                       (PRINTOUT REPORT AR# "    " (SELECTQ PASS/FAIL
                                                                          (:PASS "OK")
                                                                          (:FAIL "-BAD-")
```

```
                                                                    "??")
                                        T)))
                              (T                                        ; The test never got run.
                                 (PRINTOUT REPORT AR# "       --" T))))
                        (FORCEOUTPUT REPORT T)
                        (SETQ REPORT-FORM (OPENTEXTSTREAM (MKATOM *AR-TEST-CASE-SUMMARY-TEMPLATE-FILE*)
                                                  NIL NIL NIL '(SEL 1)))
                        (TEDIT.NEXT REPORT-FORM)
                        (TEDIT.INSERT REPORT-FORM (DATE))
                        (TEDIT.NEXT REPORT-FORM)
                        (TEDIT.DELETE REPORT-FORM)
                        (TEDIT.RAW.INCLUDE REPORT-FORM REPORT)
                        (TEDIT.HARDCOPY REPORT-FORM PRINT-FILE)
                        (CLOSEF REPORT-FORM))))
```

(CL:DEFUN **AR-TEST-CASE-READ** (&OPTIONAL (MINIMUM-INTERESTING-AR-NUMBER 5))

;; Read the most recent results for every AR test case ever run into an array. Returns the 3 values, "lowest AR number seen", "highest AR number ;; seen", and the array of results.

```
(LET ((LOWEST-SEEN-AR 65535)
      (HIGHEST-SEEN-AR 0)
      (AR-TABLE (CL:MAKE-ARRAY 32000)))
     (CL:WITH-OPEN-STREAM (LOG (OPENSTREAM *AR-TEST-CASE-LOG-FILE* 'INPUT 'OLD))
            (WHILE (SETQ LOG-ENTRY (CL:READ LOG NIL NIL)) DO (DESTRUCTURING-BIND (AR# PASS/FAIL TESTER
                                                                                   TIME-STAMP)
                                                                    LOG-ENTRY
                                                                    (COND
                                                                       ((AND (NUMBERP AR#)
                                                                             (> AR#
                                                                                MINIMUM-INTERESTING-AR-NUMBER
                                                                                ))
                                                                        (SETQ HIGHEST-SEEN-AR
                                                                         (IMAX HIGHEST-SEEN-AR AR#))
                                                                        (SETQ LOWEST-SEEN-AR
                                                                         (IMIN LOWEST-SEEN-AR AR#))
                                                                        (ASET LOG-ENTRY AR-TABLE AR#))))))
     (CL:VALUES LOWEST-SEEN-AR HIGHEST-SEEN-AR AR-TABLE)))
```

(CL:DEFUN **AR-FAILING-TEST-CASES** (&OPTIONAL PRINT-FILE)

;; Collect a list of the AR test cases that have run but have failed.

```
(LET (LOWEST-SEEN-AR HIGHEST-SEEN-AR AR-TABLE LOG-ENTRY REPORT-FORM)
     (CL:MULTIPLE-VALUE-SETQ (LOWEST-SEEN-AR HIGHEST-SEEN-AR AR-TABLE)
            (AR-TEST-CASE-READ 5))
     (FOR AR# FROM LOWEST-SEEN-AR TO HIGHEST-SEEN-AR WHEN (SETQ LOG-ENTRY (DESTRUCTURING-BIND (AR#
                                                                                               PASS/FAIL
                                                                                                TESTER

                                                                                               TIME-STAMP
                                                                                                )
                                                                                    (CL:AREF AR-TABLE AR#)
                                                                                    (EQ PASS/FAIL :FAIL)))
       COLLECT AR#)))
```

;; Patch-file creation support.

;; See {Eris}<Lispcore>Internal>Doc>Making-a-Patch.TEdit for details.

```
(DEFCOMMAND "PATCH" (&REST ARS) (\\MAKE-PATCH-FILE ARS))
```

```
(DEFCOMMAND "LIBPATCH" (&REST ARS)
```

;; Given a list of ARs on the command line, ask for the Library file name(s) of the files that patch it; move those files to the patch directory; log the ;; patch.

```
  (LET (FILE-LINE FILES COMPILED-FILES FOUND-FILES)
       (SETQ FILE-LINE (PROMPTFORWORD "Library File(s) that make up the patch (spaces between names):  " NIL
                                 NIL NIL NIL NIL (CHARCODE (CR LF))))
       (SETQ FILES (CL:WITH-INPUT-FROM-STRING (LINE FILE-LINE)
                             (WHILE (NOT (EOFP LINE)) COLLECT (PROG1 (RATOM LINE)
                                                                     (OR (EOFP LINE)
                                                                         (SKIPSEPRS LINE))))))
       (SETQ FOUND-FILES (FOR FILE IN FILES COLLECT (OR (FINDFILE FILE NIL '({ERIS}<LISPCORE>LIBRARY>))
                                                        (HELP "File not found in the patch list."))))
       (SETQ COMPILED-FILES (FOR FILE IN FILES COLLECT (FINDFILE-WITH-EXTENSIONS FILE '(
                                                                                        {ERIS}<LISPCORE>LIBRARY>
                                                                                         )
                                                                                 '(DFASL LCOM))))
       (COPYFILES FOUND-FILES "{ERIS}<LISPCORE>PATCHES>LIBRARY>")
       (COPYFILES COMPILED-FILES "{ERIS}<LISPCORE>PATCHES>LIBRARY>")
       (\\LOG-A-PATCH ARS FILES)))
```

```
(DEFCOMMAND "LOGPATCH" (ARS FILES)
```

```
    ;; Given a list of ARs on the command line, ask for the Library file name(s) of the files that patch it; move those files to the patch directory; log the
    ;; patch.

    (LET NIL (COND
                ((AND (LISTP ARS)
                      (LISTP FILES))
                 (\\LOG-A-PATCH ARS FILES))
                (T (PRINTOUT T "ARs or FILEs weren't a list."))))))

(DEFINEQ

(\\MAKE-PATCH-FILE
  (LAMBDA (AR-LIST)                                          ; Edited  8-Nov-88 19:16 by jds
    (LET* ((MAIN-AR (CAR AR-LIST))
           (FILE-NAME (CL:FORMAT NIL "AR~{-~A~}-PATCH" AR-LIST))
           (COMSNAME (PACK* FILE-NAME 'COMS))
           (AR-FEATURE-LIST (|for| AR |in| AR-LIST |collect| (CL:INTERN (CONCAT "AR-" AR)
                                                                        "KEYWORD"))))
          (SET COMSNAME (COPYALL `(                              (* |;;| (\\\, (CL:FORMAT NIL "Patch file ~A.  Contains fixes for the
                                                                 AR(s) ~S." FILE-NAME AR-LIST)))
                                  (ADDVARS (*FEATURES* ,@AR-FEATURE-LIST))
                                  (FILES "Pre-requisite patches")
                                  (FNS)
                                  (VARS))))                      ; Build fileCOMS
          (ED (MKATOM FILE-NAME)
              :FILES)                                            ; Edit the file, so user can fill in functions, etc., to be saved in the
                                                                 ; patch.
          ;; Log the patch:
          (\\LOG-A-PATCH AR-LIST (LIST (MKATOM FILE-NAME))))))


(\\LOG-A-PATCH
  (LAMBDA (AR-LIST FILE-LIST)                                ; Edited 14-Oct-88 14:21 by jds
    ;; Write the log entries for a patch, setting for each AR patches the name of the patch file(s).  Send a message to Cheryl and John saying that it got
    ;; patched.

    (LET* ((PATCH-NOTIFY-LIST (SELECTQ (LAFITEMODE)
                                       (GV "James.envos, Sybalsky.envos")
                                       (NS "James:AISNorth, Sybalsky:AISNorth")
                                       (ERROR "LAFITEMODE not set.")))
           MSG)
          (LET ((*READTABLE* FILERDTBL))
               (CL:WITH-OPEN-STREAM (LOG (OPENSTREAM "{ERIS}<Lispcore>Patches>Patch-Directory" 'APPEND
                                                     'OLD))
                                    (|for| AR |in| AR-LIST |do| (PRINT (LIST AR FILE-LIST (USERNAME)
                                                                             (DATE))
                                                                       LOG))))
          (SETQ MSG (OPENTEXTSTREAM (CONCAT (CL:FORMAT NIL "Subject: Patches Created for AR~p~{ ~A~#[.~;,
                                                           and~:;,~]~}" (FLENGTH AR-LIST)
                                                           AR-LIST)
                                            "
                                            To: " PATCH-NOTIFY-LIST "
                                            cc: " (FULLUSERNAME)
                                            (CL:FORMAT NIL "~%~%Patch Files:~{~14T~A~%~}" FILE-LIST))
                                    NIL NIL NIL (LIST 'FONT LAFITEEDITORFONT)))
          (ADD.PROCESS `(\\SENDMESSAGE ',MSG 'NAME 'MESSAGESENDER)))))

)

(PUTPROPS RELEASETOOLS FILETYPE CL:COMPILE-FILE)

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

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML )

(ADDTOVAR LAMA )
)

(PUTPROPS RELEASETOOLS COPYRIGHT ("Venue & Xerox Corporation" 1987 1988 1990))
```

FUNCTION INDEX

COMMAND INDEX

VARIABLE INDEX

PROPERTY INDEX

RECORD INDEX