File created: 8-Dec-91 15:16:14 **{DSK}<users>sybalsky>PUBS>IMTRAN.;2**

changes to: (FNS PARTITION.LIST)
(VARS IMTRANCOMS)

previous date: 11-Jul-86 12:39:31 {DSK}<users>sybalsky>PUBS>IMTRAN.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
(RPAQQ IMTRANCOMS
       [;; Functions to help converting IM format to TEdit.
        (FNS ADD.ANY.PARENT.PROP.IF ADD.MY.PROP.IF ANC.INDENT ANC.WIDTH ANY.PARENT.CORRECT.ARG ANY.PARENT.EVAL
             ANY.PARENT.TO.MATCH ARGS.REMAINING ARGS.REMAINING.AFTER CHANGE.INDENT CONCCONC CORRECT.ARG
             CORRECT.TO DECODE.TO.ARG.NAME.SYNONYM DECODE.TO.NAME.SYNONYM DEFINED.TO DESCRIBE.CURRENT.TO
             DESCRIBE.TO DUMP.ARG DUMP.FORMAT FLUSH.ARG GET.ANY.PARENT.PROP GET.ARG GET.ARG.SAV GET.ARG.TYPE
             GET.MY.PROP GET.TO.TYPE GOBBLE GOBBLE.DUMP GOBBLE.DUMP.UNDEFINED GOBBLE.FLUSH GOBBLE.SAVE
             GOBBLE.TRIVIAL IM.DUMP.CHARS IM.FILE.EOF IM.PRINT.MESSAGE IMTRAN INCLUDE.FILE LIST.ORDER
             LIST.TO.STRING MAKE.SAVE OPEN.IM.FILE PARSE.ATOM PARSE.INDEX.SPEC PARSE.LIST
             PARSE.NUMBER.OR.PERCENTAGE PARSE.STRING PARSE.TO.NAME PARTITION.LIST PRINT.INFILE.NOTES PUT.MY.PROP
             SAVE.ARG SAVE.ARGS SAVE.ARGS.EMPTY SAVE.INFILE.NOTE SCRUNCH.REFS SEND.IMPLICIT SKIP.WHITE.SPACE
             STANDARD.DUMMY.TO.PROG TAG.LIST.MATCH TO.MATCH TO.NAME.CHAR TRANSLATE.SPECIAL.TYPES TRIVIAL.ARG)
        (RECORDS TO)
        (VARS IM.INFILE.NOTE.TAGS)
        (INITVARS (IM.NOTE.FLG NIL)
               (IM.DRAFT.FLG NIL)
               (IM.INDEX.FILE.FLG NIL)
               (IM.REF.FLG NIL)
               (IM.SEND.IMPLICIT NIL)
               (IM.EVEN.FLG NIL)
               (IM.CHECK.DEFS NIL))
        (PROP INFO DUMPOUT)
        (MACROS IM.ERROR IM.WARNING)
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA SAVE.ARGS)
                                                                              (NLAML GET.ARG.SAV)
                                                                              (LAMA])
```

;; Functions to help converting IM format to TEdit.

```
(DEFINEQ

(ADD.ANY.PARENT.PROP.IF
  [LAMBDA (PROPNAME VAL)                                    (* mjs "15-APR-83 10:31")
    (ANY.PARENT.EVAL (LIST 'ADD.MY.PROP.IF (LIST 'QUOTE PROPNAME)
                           (LIST 'QUOTE VAL])

(ADD.MY.PROP.IF
  [LAMBDA (PROPNAME VAL)                                    (* mjs "15-APR-83 10:32")
    (if (GET.MY.PROP PROPNAME)
        then [PUT.MY.PROP PROPNAME (CONS VAL (LISTP (GET.MY.PROP PROPNAME]
             T
      else NIL])

(ANC.INDENT
  [LAMBDA NIL                                               (* mjs "14-APR-83 13:31")
    (GET.ANY.PARENT.PROP 'INDENT])

(ANC.WIDTH
  [LAMBDA NIL                                               (* mjs "14-APR-83 13:31")
    (GET.ANY.PARENT.PROP 'WIDTH])

(ANY.PARENT.CORRECT.ARG
  [LAMBDA (NAME)                                            (* mjs " 8-APR-83 12:32")
    (ANY.PARENT.EVAL (LIST 'CORRECT.ARG (LIST 'QUOTE NAME])

(ANY.PARENT.EVAL
  [LAMBDA (FORM)                                            (* mjs "11-APR-83 10:00")
    (PROG ((S (STKSCAN 'TO.NAME))
           VAL)
      loop
          (SETQ S (STKNTH 1 S S))
          (SETQ S (STKSCAN 'TO.NAME S S))
          (COND
             ((NULL S)
              (RETURN NIL))
             ((SETQ VAL (STKEVAL S FORM))
              (RETURN VAL))
             (T (GO loop])

(ANY.PARENT.TO.MATCH
  [LAMBDA (NAME BEGINFLG TAGLIST)                                   (* mjs " 8-APR-83 16:00")
    (ANY.PARENT.EVAL (LIST 'TO.MATCH (LIST 'QUOTE NAME)
                               (LIST 'QUOTE BEGINFLG)
                               (LIST 'QUOTE TAGLIST])

(ARGS.REMAINING
  [LAMBDA (NAME ARGS)                                               (* mjs "21-APR-83 13:28")

             (* returns T if NAME is a correct next-arg according to the TO.ARGS.REMAINING list ARGS.
             Specifically if the next non-list member of ARGS is EQ to NAME or NAME is a member of one of the list elements of ARGS
             before the next non-list member, then NAME is a correct next-arg.)

    (if (NLISTP (CAR ARGS))
        then (EQ (CAR ARGS)
                 NAME)
      else (OR (FMEMB NAME (CAR ARGS))
               (ARGS.REMAINING NAME (CDR ARGS])

(ARGS.REMAINING.AFTER
  [LAMBDA (NAME ARGS)                                               (* mjs "21-APR-83 13:28")

             (* returns the new TO.ARGS.REMAINING list after arg NAME has been processed.
             Assumes that NAME is a correct next-arg of ARGS, as according to ARGS.REMAINING NIL)

    (if (NULL ARGS)
        then NIL
      elseif (EQ NAME (CAR ARGS))
        then (CDR ARGS)
      elseif (AND (LISTP (CAR ARGS))
                  (FMEMB NAME (CAR ARGS)))
        then ARGS
      else (ARGS.REMAINING.AFTER NAME (CDR ARGS])

(CHANGE.INDENT
  [LAMBDA (X)                                                       (* mjs "10-Apr-85 10:07")
    (if (EQ X 'NONE)
        then (PRIN3 "\parshape 0 {}" IM.OUTFILE)
      else (printout IM.OUTFILE "\parshape 1 " .I4.10 X "pt " .I4.10 (IDIFFERENCE INITIAL.WIDTH X)
                  "pt " "{}"])

(CONCCONC
  [LAMBDA (X Y)                                                     (* mjs "13-APR-83 16:44")
                                                                    (* takes two TCONC-format lists, and concatonates them)
    (COND
       ((NLISTP X)
        Y)
       ((NULL (CAR Y))
        X)
       ((NULL (CAR X))
        (RPLNODE2 X Y))
       (T (RPLACD (CDR X)
                 (CAR Y))
          (RPLACD X (CDR Y])

(CORRECT.ARG
  [LAMBDA (NAME)                                                    (* mjs "21-APR-83 13:30")

             (* returns non-NIL if NAME is an arg that appears in the args.remaining list of the current TO)

    (ARGS.REMAINING NAME TO.ARGS.REMAINING])

(CORRECT.TO
  [LAMBDA (NAME)                                                    (* mjs " 9-NOV-83 11:56")
                                                                    (* returns T if NAME can occur as a TO below TO.NAME)
    (if (NULL TO.NAME)
        then                                                        (* anything can occur below top-level TO)
             T
      elseif (OR (EQ TO.NAME 'COMMENT)
                 (EQ TO.NAME 'NOTE))
        then                                                        (* anything can occur below a comment or a note)
             T
      elseif (SELECTQ (GET.ARG.TYPE TO.NAME)
                (SIMPLE                                             (* a COMPLEX TO cannot occur within a SIMPLE arg)
                        (EQ (GET.TO.TYPE NAME)
                          NIL))
                (CHARS                                                  (* no TO can occur within a CHARS arg)
                      T)
                NIL)
         then NIL
       elseif (SELECTQ NAME
                (CHAPTER                                                (* exception 1%: chapter can only occur in top-level TO)
                       T)
                ((SUBSEC FNDEF VARDEF PROPDEF MACDEF COMDEF)            (* exception 2%: subsec, ---def can only occur within subsec,
                                                                        chapter, or top-level TO)
                    [NOT (FMEMB TO.NAME '(CHAPTER SUBSEC])
                (FOOT                                                   (* exception 3%: foot cannot occur in foot)
                     [OR (EQ TO.NAME 'FOOT)
                         (ANY.PARENT.EVAL '(EQ TO.NAME 'FOOT])
                NIL)
         then NIL
       else T])
```

(DECODE.TO.ARG.NAME.SYNONYM

```
  [LAMBDA (NAME)                                                        (* mjs "14-APR-83 14:40")
                                                                        (* eventually should do arg name synonym search)
    (COND
       ((LISTGET (fetch TO.ARG.SYNONYMS of TO.NAME)
               NAME))
       (T NAME])
```

(DECODE.TO.NAME.SYNONYM

```
  [LAMBDA (NAME)                                                        (* mjs "13-APR-83 18:56")
    (COND
       ((CADR (ASSOC NAME TO.SYNONYM.LIST)))
       (T NAME])
```

(DEFINED.TO

```
  [LAMBDA (NAME)                                                        (* mjs " 8-APR-83 11:02")
    (fetch TO.PROG of NAME])
```

(DESCRIBE.CURRENT.TO

```
  [LAMBDA NIL                                                           (* mjs "13-APR-83 18:22")
    (DESCRIBE.TO (if TO.BEGIN.FLG
                     then 'BEGIN
                   else NIL)
           TO.ORIG.NAME TO.TAG.LIST TO.ORIG.ARG.NAME])
```

(DESCRIBE.TO

```
  [LAMBDA (BEGIN.END ORIG.NAME TAG.LIST ORIG.ARG.NAME)                  (* mjs " 3-MAY-83 14:42")
    (CONCAT "{" (COND
                   (BEGIN.END (CONCAT BEGIN.END " " ORIG.NAME " " (LIST.TO.STRING TAG.LIST)
                                     "}"))
                   (T ORIG.NAME))
           (COND
              (ORIG.ARG.NAME (CONCAT " (arg " ORIG.ARG.NAME ")"))
              (T ""])
```

(DUMP.ARG

```
  [LAMBDA (NO.SKIP.FLG)                                                 (* mjs "21-APR-83 14:14")
    (if (fetch TO.ARGS of TO.NAME)
        then (if TO.ARG.NAME
                 then (GOBBLE.DUMP NO.SKIP.FLG)
                      (SETQ TO.ARG.NAME NIL)
               else (ERROR "TO PROG BUG: attempting to dump arg not yet gotten"))
      else (if (NOT TO.DONE.FLG)
               then (GOBBLE.DUMP NO.SKIP.FLG)
                    (SETQ TO.DONE.FLG T)
             else (ERROR "TO PROG BUG: attempting to process unlabeled arg twice"])
```

(DUMP.FORMAT

```
  [LAMBDA (FORMAT.TYPE FORMAT)                                          (* mjs "13-APR-83 17:01")
    (COND
       ((BOUNDP 'GOBBLE.SAVE.CONC)
        (TCONC GOBBLE.SAVE.CONC (CONS FORMAT.TYPE FORMAT)))
       (T (DUMP (CONS FORMAT.TYPE FORMAT])
```

(FLUSH.ARG

```
  [LAMBDA NIL                                                           (* mjs "21-APR-83 14:13")
    (if (fetch TO.ARGS of TO.NAME)
        then (if TO.ARG.NAME
                 then (GOBBLE.FLUSH)
                      (SETQ TO.ARG.NAME NIL)
               else (ERROR "TO PROG BUG: attempting to dump arg not yet gotten"))
```

```
      else (if (NOT TO.DONE.FLG)
               then (GOBBLE.FLUSH)
                    (SETQ TO.DONE.FLG T)
             else (ERROR "TO PROG BUG: attempting to process unlabeled arg twice"])


(GET.ANY.PARENT.PROP
  [LAMBDA (PROPNAME)                                          (* mjs "11-APR-83 10:08")
    (ANY.PARENT.EVAL (LIST 'GET.MY.PROP (LIST 'QUOTE PROPNAME])


(GET.ARG
  [LAMBDA NIL                                                 (* mjs "12-Jul-85 12:32")
    (PROG (C SAVE.FILE.PTR ORIG.NAME BEGIN.END TAG.LIST NAME)
          (if (OR TO.ARG.NAME TO.DONE.FLG (NULL (fetch TO.ARGS of TO.NAME)))
              then (ERROR "TO PROG BUG: called GET.ARG at bad time"))
          (SKIP.WHITE.SPACE)
          (SETQ C (if (EOFP IM.INFILE)
                      then (IM.FILE.EOF)
                    else (BIN IM.INFILE)))
          (if (EQ C (CHARCODE }))
              then (if TO.BEGIN.FLG
                       then (IM.ERROR "} closes " (DESCRIBE.CURRENT.TO)))
                   (SETQ TO.DONE.FLG T)
                   (RETURN NIL)
            elseif (EQ C 'EOF)
              then (if (NEQ TO.NAME NIL)
                       then (IM.ERROR "EOF - closing bracket for " (DESCRIBE.CURRENT.TO)))
                   (SETQ TO.DONE.FLG T)
                   (RETURN NIL)
            elseif (NEQ C (CHARCODE {))
              then (IM.ERROR "Garbage chars found between labeled args of " (DESCRIBE.CURRENT.TO)
                          " --- auto-completed")
                   (SETQ SAVE.FILE.PTR (SUB1 (GETFILEPTR IM.INFILE)))
                   (SETQ TO.DONE.FLG T)
                   (RETURN NIL))
          (SETQ SAVE.FILE.PTR (SUB1 (GETFILEPTR IM.INFILE)))
          (SETQ ORIG.NAME (PARSE.TO.NAME))                    (* if PARSE.TO.NAME returns a list, it must be a BEGIN/END,
                                                              so unpack elements)
          (if (NLISTP ORIG.NAME)
              then (SETQ BEGIN.END NIL)
                   (SETQ TAG.LIST NIL)
            else (SETQ BEGIN.END (CAR ORIG.NAME))
                 (SETQ TAG.LIST (CADDR ORIG.NAME))
                 (SETQ ORIG.NAME (CADR ORIG.NAME)))
          (if (EQ BEGIN.END 'BEGIN)
              then (IM.ERROR (DESCRIBE.TO BEGIN.END ORIG.NAME TAG.LIST)
                          " found between labeled args of "
                          (DESCRIBE.CURRENT.TO)
                          " --- auto-completed")
                   (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                   (SETQ TO.DONE.FLG T)
                   (RETURN NIL)
            elseif (EQ BEGIN.END 'END)
              then (SETQ NAME (DECODE.TO.NAME.SYNONYM ORIG.NAME))
                   (if (TO.MATCH NAME T TAG.LIST)
                       then                                   (* normal {end return)
                            (SETQ TO.DONE.FLG T)
                            (RETURN NIL)
                     elseif (ANY.PARENT.TO.MATCH NAME T TAG.LIST)
                       then (IM.ERROR (DESCRIBE.TO 'END ORIG.NAME TAG.LIST)
                                   " found between labeled args of "
                                   (DESCRIBE.CURRENT.TO)
                                   " --- auto-completed")
                            (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                            (SETQ TO.DONE.FLG T)
                            (RETURN NIL)
                     else (IM.ERROR "Unmatched " (DESCRIBE.TO 'END ORIG.NAME TAG.LIST)
                                 " --- flushed")
                          (RETURN (GET.ARG)))
            elseif BEGIN.END
              then (ERROR "PARSE.TO.NAME returned something besides BEGIN or END"))
          (SETQ NAME (DECODE.TO.ARG.NAME.SYNONYM ORIG.NAME))
          (if (CORRECT.ARG NAME)
              then (SETQ TO.ORIG.ARG.NAME ORIG.NAME)
                   (SETQ TO.ARG.NAME NAME)
                   (SETQ TO.ARGS.REMAINING (ARGS.REMAINING.AFTER NAME TO.ARGS.REMAINING))
                   (RETURN NAME)
            else (IM.ERROR (DESCRIBE.TO NIL ORIG.NAME NIL)
                        " found between labeled args of "
                        (DESCRIBE.CURRENT.TO)
                        " --- auto-completed")
                 (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                 (SETQ TO.DONE.FLG T)
                 (RETURN NIL])
```

```
(GET.ARG.SAV
  [NLAMBDA (NAME)                                                   (* mjs "13-APR-83 10:23")
    (LISTGET (GET.MY.PROP 'SAVE.ARGS.PROPLIST)
           NAME])

(GET.ARG.TYPE
  [LAMBDA (NAME)                                                    (* mjs "22-APR-83 13:19")

             (* if NAME is a TO with labeled args, use the current arg name as index into the property list TO.ARG.TYPE.
             otherwise TO.ARG.TYPE is the arg type)

    (if (fetch TO.ARGS of NAME)
        then (LISTGET (fetch TO.ARG.TYPE of NAME)
                    TO.ARG.NAME)
      else (fetch TO.ARG.TYPE of NAME])

(GET.MY.PROP
  [LAMBDA (PROPNAME)                                                (* mjs "11-APR-83 10:10")
    (LISTGET TO.PROP.LIST PROPNAME])

(GET.TO.TYPE
  [LAMBDA (NAME)                                                    (* mjs "22-APR-83 13:20")

             (* note that undefined TOs are always SIMPLE, so they can appear anywhere except in CHARS TOs)

    (if (fetch TO.PROG of NAME)
        then (fetch TO.TYPE of NAME)
      else 'SIMPLE])

(GOBBLE
  [LAMBDA NIL                                                       (* mjs "12-Jul-85 12:39")

             (* * reads and dumps chars until closing } or {end)

    (PROG ((SAVE.DUMP.FLG (BOUNDP 'GOBBLE.SAVE.CONC))
           C SAVE.FILE.PTR CLIST ORIG.NAME NAME BEGIN.END TAG.LIST SAVE.CONC)
          (if SAVE.DUMP.FLG
              then (SETQ SAVE.CONC GOBBLE.SAVE.CONC))
      loop
          (SETQ C (if (EOFP IM.INFILE)
                      then (IM.FILE.EOF)
                    else (BIN IM.INFILE)))                          (* if C is just a normal char, just dump it and loop)
          (if (EQ C 'EOF)
              then (if (NEQ TO.NAME NIL)
                       then (IM.ERROR "EOF - closing bracket for " (DESCRIBE.CURRENT.TO)))
                   (RETURN NIL))
          (if (EQ C (CHARCODE }))
              then (if (AND TO.BEGIN.FLG (NOT TO.ARG.NAME))
                       then (IM.ERROR "} closes " (DESCRIBE.CURRENT.TO)))
                   (RETURN NIL))
          (if (NEQ C (CHARCODE {))
              then

          (* if it is anything else besides a left bracket, it is a character to dump)

                   (if SAVE.DUMP.FLG
                       then (TCONC SAVE.CONC C)
                     else (DUMP C))
                   (GO loop))                                       (* at this point, C must be a left bracket)

          (* * before getting name after {, save current file ptr so you can back it up if necessary.)

          (* * note%: complete TO name (or Begin spec) must be in one file, so that you can back up)

          (SETQ SAVE.FILE.PTR (SUB1 (GETFILEPTR IM.INFILE)))
          (SETQ ORIG.NAME (PARSE.TO.NAME))                          (* if PARSE.TO.NAME returns a list, it must be a BEGIN/END,
                                                                    so unpack elements)
          (if (NLISTP ORIG.NAME)
              then (SETQ BEGIN.END NIL)
                   (SETQ TAG.LIST NIL)
            else (SETQ BEGIN.END (CAR ORIG.NAME))
                 (SETQ TAG.LIST (CADDR ORIG.NAME))
                 (SETQ ORIG.NAME (CADR ORIG.NAME)))
          (SETQ NAME (DECODE.TO.NAME.SYNONYM ORIG.NAME))
          (COND
             [(EQ BEGIN.END 'END)
              (COND
                 ((AND (NOT TO.ARG.NAME)
                       (TO.MATCH NAME T TAG.LIST))                  (* normal {end return)
                  (RETURN NIL))
                 ((OR (TO.MATCH NAME T TAG.LIST)
                      (ANY.PARENT.TO.MATCH NAME T TAG.LIST))
                  (IM.ERROR (DESCRIBE.TO 'END ORIG.NAME TAG.LIST)
                         " occurred in "
                         (DESCRIBE.CURRENT.TO)
                         " --- auto-completed")
                  (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                  (RETURN NIL))
                 (T (IM.ERROR "Unmatched " (DESCRIBE.TO 'END ORIG.NAME TAG.LIST)
                           " --- flushed")
                     (GO loop]
             ((AND BEGIN.END (NEQ BEGIN.END 'BEGIN))
              (ERROR "PARSE.TO.NAME returned something besides BEGIN or END")))

          (* * now, we must be beginning a TO)

          (COND
             ((AND (DEFINED.TO NAME)
                   (NOT (CORRECT.TO NAME)))

          (* if name is defined but not correct, auto-complete to pop up to where it is correct)

              (IM.ERROR (DESCRIBE.TO BEGIN.END ORIG.NAME TAG.LIST)
                     " not legal in "
                     (DESCRIBE.CURRENT.TO)
                     " --- auto-completed")
              (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
              (RETURN NIL)))
          (COND
             ((AND (NOT BEGIN.END)
                   (NOT (DEFINED.TO NAME))
                   (OR (CORRECT.ARG ORIG.NAME)
                       (ANY.PARENT.CORRECT.ARG ORIG.NAME)))

          (* if name is not in a {begin and is undefined, AND it is a legal arg for the current TO or a parent TO, auto-complete current
          TO or arg)

              (IM.ERROR "Possible Argument " (DESCRIBE.TO NIL ORIG.NAME NIL)
                     " occurred in "
                     (DESCRIBE.CURRENT.TO)
                     " --- auto-completed")
              (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
              (RETURN NIL)))
          [PROG ((TO.NAME NAME)
                 (TO.ORIG.NAME ORIG.NAME)
                 (TO.BEGIN.FLG (NOT (NULL BEGIN.END)))
                 (TO.TAG.LIST TAG.LIST)
                 (TO.ARG.NAME NIL)
                 (TO.ORIG.ARG.NAME NIL)
                 (TO.ARGS.REMAINING (fetch TO.ARGS of NAME))
                 (TO.DONE.FLG NIL)
                 (TO.PROP.LIST (CONS)))
                (DECLARE (SPECVARS TO.NAME TO.ORIG.NAME TO.BEGIN.FLG TO.ARG.NAME TO.TAG.LIST TO.ORIG.ARG.NAME
                                TO.ARGS.REMAINING TO.DONE.FLG TO.PROP.LIST))
                (if (DEFINED.TO NAME)
                    then (APPLY (fetch TO.PROG of NAME))
                  else (GOBBLE.DUMP.UNDEFINED))
                (if (NOT TO.DONE.FLG)
                    then (if (AND (fetch TO.ARGS of NAME)
                                  (NULL TO.ARG.NAME)
                                  (NULL TO.ARGS.REMAINING))
                             then

          (* if the TO has used up all args, but hasn't formally completed scanning, close it automatically)

                                  (GET.ARG)
                           else (ERROR (CONCAT "TO function: " (fetch TO.PROG of NAME)
                                              " ended with input in bad state"]
          (GO loop])

(GOBBLE.DUMP
  [LAMBDA (NO.SKIP.FLG)                                          (* mjs "21-APR-83 13:48")
    (PROG NIL
          (COND
             ((NOT NO.SKIP.FLG)
              (SKIP.WHITE.SPACE)))
          (GOBBLE)
          (SETQ TO.ORIG.ARG.NAME NIL)
          (SETQ TO.ARG.NAME NIL])

(GOBBLE.DUMP.UNDEFINED
  [LAMBDA NIL                                                    (* mjs "10-Apr-85 09:53")

          (* * old fn def%: (WARNING T TO.ORIG.NAME " is undefined as a TO  --- contents will be dumped")
          (DUMP.ARG))

          (* * current behavior%: for unidentified name, print out "{<bad-name> <arg>}")
```

```
    (SAVE.INFILE.NOTE 'BAD.TO.NAME TO.ORIG.NAME)
    (IM.DUMP.CHARS "{")
    (IM.DUMP.CHARS TO.ORIG.NAME)
    (DUMP.ARG T)                                        (* dump rest of TO, WITHOUT SKIPPING SPACES)
    (IM.DUMP.CHARS "}"])
```

```
(GOBBLE.FLUSH
  [LAMBDA NIL                                           (* mjs "22-APR-83 10:37")

          (* this flushes all gobbled chars by means of an incredible hack ---
          by initializing GOBBLE.SAVE.CONC to NIL rather than (NIL)%, none of the TCONCs can update the saved list)

    (PROG ((GOBBLE.SAVE.CONC NIL))
          (DECLARE (SPECVARS GOBBLE.SAVE.CONC))
          (GOBBLE)
          (SETQ TO.ORIG.ARG.NAME NIL)
          (SETQ TO.ARG.NAME NIL])
```

```
(GOBBLE.SAVE
  [LAMBDA (NO.SKIP.FLG)                                 (* mjs "21-APR-83 13:49")
    (PROG ((GOBBLE.SAVE.CONC (CONS)))
          (DECLARE (SPECVARS GOBBLE.SAVE.CONC))
          (COND
             ((NOT NO.SKIP.FLG)
              (SKIP.WHITE.SPACE)))
          (GOBBLE)
          (SETQ TO.ORIG.ARG.NAME NIL)
          (SETQ TO.ARG.NAME NIL)
          (RETURN GOBBLE.SAVE.CONC])
```

```
(GOBBLE.TRIVIAL
  [LAMBDA NIL                                           (* mjs "12-Jul-85 12:41")
    (PROG (C SAVE.FILE.PTR PARSED.NAME)
          (SKIP.WHITE.SPACE)
          (SETQ SAVE.FILE.PTR (SUB1 (GETFILEPTR IM.INFILE)))
          (SETQ C (if (EOFP IM.INFILE)
                      then (IM.FILE.EOF)
                    else (BIN IM.INFILE)))
          (if (EQ C 'EOF)
              then (if (NEQ TO.NAME NIL)
                       then (IM.ERROR "EOF - closing bracket for " (DESCRIBE.CURRENT.TO)))
                   (RETURN NIL)
            elseif (EQ C (CHARCODE }))
              then (if (AND TO.BEGIN.FLG (NOT TO.ARG.NAME))
                       then (IM.ERROR "} closes " (DESCRIBE.CURRENT.TO)))
                   (RETURN NIL)
            elseif (EQ C (CHARCODE {))
              then (if TO.ARG.NAME
                       then (IM.ERROR "{ terminates trivial arg " (DESCRIBE.CURRENT.TO))
                            (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                            (RETURN NIL)
                     else (SETQ PARSED.NAME (PARSE.TO.NAME))
                          (if (AND (LISTP PARSED.NAME)
                                   (EQ (CAR PARSED.NAME)
                                       'END)
                                   (TO.MATCH (CADR PARSED.NAME)
                                          T
                                          (CADDR PARSED.NAME)))
                              then (RETURN NIL)
                            else (IM.ERROR (if (LISTP PARSED.NAME)
                                               then (DESCRIBE.TO (CAR PARSED.NAME)
                                                           (CADR PARSED.NAME)
                                                           (CADDR PARSED.NAME))
                                             else (DESCRIBE.TO NIL PARSED.NAME NIL))
                                        " found in trivial "
                                        (DESCRIBE.CURRENT.TO)
                                        " --- auto-completed")
                                 (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                                 (RETURN NIL)))
            else (IM.ERROR "garbage character found in trivial " (DESCRIBE.CURRENT.TO)
                        " --- auto-completed")
                 (SETFILEPTR IM.INFILE SAVE.FILE.PTR)
                 (RETURN NIL])
```

```
(IM.DUMP.CHARS
  [LAMBDA (X)                                           (* mjs "13-APR-83 17:01")
    (COND
       [(AND (LISTP X)
             (LISTP (CDR X)))
        (COND
           ((BOUNDP 'GOBBLE.SAVE.CONC)
            (CONCCONC GOBBLE.SAVE.CONC X))
           (T (for C in (CAR X) do (DUMP C]
        (X (IM.DUMP.CHARS (MAKE.SAVE X])

(IM.FILE.EOF
  [LAMBDA NIL                                                   (* mjs "25-Jul-85 10:49")
    (if INFILE.STACK
        then (IM.WARNING "EOF - included file: " IM.INFILE.FILENAME)
             (PRINT.INFILE.NOTES)
             (CLOSEF IM.INFILE)
             (PROG ((LASTFILEINFO (pop INFILE.STACK)))
                   (SETQ IM.INFILE (CAR LASTFILEINFO))
                   (SETQ IM.INFILE.FILENAME (CDR LASTFILEINFO)))
             (BIN IM.INFILE)
      else (IM.WARNING "EOF - top-level file: " IM.INFILE.FILENAME)
           (PRINT.INFILE.NOTES)
           'EOF])

(IM.PRINT.MESSAGE
  [LAMBDA (WARNING.STRING PRINT.CONTEXT.FLAG MESSAGE.LIST)      (* mjs "12-Sep-85 08:30")
    (PROG NIL
          [if [AND ERRFILE.NAME (OR (NULL ERRFILE)
                                    (NOT (OPENP ERRFILE]
              then (SETQ ERRFILE (OPENSTREAM ERRFILE.NAME 'OUTPUT 'NEW]
          (for PRINTFILE in (LIST T ERRFILE) when PRINTFILE
             do

          (* * print out identical warning on terminal <PRINTFILE=T> and in error file <PRINTFILE=ERRFILE>.
          If ERRFILE=NIL, warning is only printed on terminal)

                (PRIN3 WARNING.STRING PRINTFILE)
                (for Y in MESSAGE.LIST do (PRIN3 Y PRINTFILE))
                (TERPRI PRINTFILE)
                (if (AND PRINT.CONTEXT.FLAG (OPENP IM.INFILE)
                         (NOT (EOFP IM.INFILE)))
                    then (PROG ((SAVEPOS (GETFILEPTR IM.INFILE))
                                BEGINPOS ENDPOS)
                               (SETQ BEGINPOS (IMAX 0 (IDIFFERENCE SAVEPOS 100)))
                               (SETQ ENDPOS (IMIN (GETEOFPTR IM.INFILE)
                                                  (IPLUS SAVEPOS 100)))
                               (SETFILEPTR IM.INFILE BEGINPOS)
                               (printout PRINTFILE "CONTEXT:" T)
                               (until (IGEQ (GETFILEPTR IM.INFILE)
                                            ENDPOS)
                                  do (BOUT PRINTFILE (if (FIXP (BIN IM.INFILE))
                                                         else (CHARCODE ~)))
                                     (if (EQ (GETFILEPTR IM.INFILE)
                                             SAVEPOS)
                                         then (printout PRINTFILE T "---->" T)))
                               (printout PRINTFILE T "ENDCONTEXT:" T T)
                               (SETFILEPTR IM.INFILE SAVEPOS])

(IMTRAN
  [LAMBDA (INFILE.NAME)                                         (* mjs "25-Jul-85 10:44")
    (PROG ((IM.INFILE NIL)
           (INFILE.STACK NIL)
           (IM.INFILE.FILENAME)
           (WARNING.COUNT.DOWN -1)
           (SUBSEC.COUNT.LIST '(0))
           (SUBSEC.LAST.SUB 0)
           (FOOTNOTE.NUM 0)
           (FIGURE.NUM 0)
           (INITIAL.WIDTH 462)
           (INITIAL.INDENT 0))
          (DECLARE (SPECVARS IM.INFILE INFILE.STACK IM.INFILE.FILENAME WARNING.COUNT.DOWN SUBSEC.COUNT.LIST
                             SUBSEC.LAST.SUB FOOTNOTE.NUM FIGURE.NUM INITIAL.WIDTH INITIAL.INDENT))
          (OPEN.IM.FILE INFILE.NAME)

          (* * note%: SUBSEC.COUNT.LIST and SUBSEC.LAST.SUB set so that if no CHAPTER TO is given, this is treated like
          chapter 0)

          (if IM.NOTE.FLG
              then (IM.WARNING "
                                    WARNING: --- Note facility enabled --- Notes will be printed ---
                    "))
          (do (PROG ((TO.NAME NIL)
                     (TO.ORIG.NAME NIL)
                     (TO.BEGIN.FLG NIL)
                     (TO.TAG.LIST NIL)
                     (TO.ARG.NAME NIL)
                     (TO.ORIG.ARG.NAME NIL)
                     (TO.ARGS.REMAINING NIL)
                     (TO.DONE.FLG NIL)
                     (TO.PROP.LIST (CONS)))
                    (DECLARE (SPECVARS TO.NAME TO.ORIG.NAME TO.BEGIN.FLG TO.ARG.NAME TO.TAG.LIST
                                       TO.ORIG.ARG.NAME TO.DONE.FLG TO.PROP.LIST))
```

```
                                                                      (* set initial indentation and width)
                    (PUT.MY.PROP 'WIDTH INITIAL.WIDTH)
                    (PUT.MY.PROP 'INDENT INITIAL.INDENT)              (* dump everything, including white space at beginning)
                    (DUMP.ARG T))
             repeatuntil (COND
                            ((AND (EOFP IM.INFILE)
                                  (NULL INFILE.STACK)))
                            (T (IM.ERROR "unmatched right bracket at top level -- ignored")
                               NIL)))
          (CLOSEF IM.INFILE])

(INCLUDE.FILE
  [LAMBDA (FILENAME)                                                  (* mjs "12-Jul-85 15:41")
    (IM.WARNING)
    (IM.WARNING)
    (OPEN.IM.FILE FILENAME])

(LIST.ORDER
  [LAMBDA (A B)                                                       (* mjs "20-JUN-83 16:33")
    (if (AND (LISTP A)
             (LISTP B))
        then (if (EQUAL (CAR A)
                        (CAR B))
                 then (LIST.ORDER (CDR A)
                                  (CDR B))
               else (LIST.ORDER (CAR A)
                                (CAR B)))
      else (ALPHORDER A B])

(LIST.TO.STRING
  [LAMBDA (LST)                                                       (* mjs "26-SEP-83 15:20")
    (PROG ((STR ""))
          [for X on LST do (SETQ STR (CONCAT STR (CAR X)
                                             (if (CDR X)
                                                 then " "
                                               else ""]
          (RETURN STR])

(MAKE.SAVE
  [LAMBDA (X)                                                         (* mjs "10-Apr-85 17:31")
    (if (NULL X)
        then NIL
      elseif (LISTP X)
        then (CONCCONC (MAKE.SAVE (CAR X))
                       (MAKE.SAVE (CDR X)))
      else (PROG [(S (if (IMAGEOBJP X)
                         then (CONS X)
                       else (CHCON X]
                 (RETURN (CONS S (LAST S])

(OPEN.IM.FILE
  [LAMBDA (FILENAME)                                                  (* mjs "10-Jul-86 15:31")
    (PROG ((DIR.LIST (if IM.INFILE.FILENAME
                         then (CONS (PACKFILENAME 'HOST (FILENAMEFIELD IM.INFILE.FILENAME 'HOST)
                                                  'DIRECTORY
                                                  (FILENAMEFIELD IM.INFILE.FILENAME 'DIRECTORY))
                                    DIRECTORIES)
                       else DIRECTORIES))
           (OPEN.FILE.RESULT NIL)
           NEW.FILENAME NEW.FILE)
          (DECLARE (SPECVARS NEW.FILENAME NEW.FILE))
          (SETQ NEW.FILENAME (if (INFILEP FILENAME)
                               elseif (if (NULL (FILENAMEFIELD FILENAME 'EXTENSION))
                                          then (FINDFILE (PACKFILENAME 'BODY FILENAME 'EXTENSION 'IM)
                                                         T DIR.LIST))
                               elseif (FINDFILE FILENAME T DIR.LIST)
                               else NIL))
          [if (NULL NEW.FILENAME)
              then (IM.WARNING "Can't find file: " FILENAME "  --- using NULL file")
                   (SETQ NEW.FILENAME '{NULL}FOO.IM)
                   (SETQ NEW.FILE (OPENSTREAM '{NULL}FOO.IM 'INPUT))
            else (for W in '(5000 10000 20000 40000 80000)
                    do (IM.WARNING "Opening File: " NEW.FILENAME)
                       [SETQ OPEN.FILE.RESULT (NLSETQ (SETQ NEW.FILE (OPENTEXTSTREAM NEW.FILENAME]
                       (if (NULL OPEN.FILE.RESULT)
                           then (IM.WARNING "Error opening file " NEW.FILENAME "  -- waiting " W " msecs and
                                       trying again")
                                (BLOCK W))
                    repeatuntil OPEN.FILE.RESULT)
                 (if (NULL OPEN.FILE.RESULT)
                     then (IM.WARNING "Can't open file " FILENAME "  --- using NULL file")
                          (SETQ NEW.FILENAME '{NULL}FOO.IM)
                    (SETQ NEW.FILE (OPENSTREAM '{NULL}FOO.IM 'INPUT]
          (if IM.INFILE
              then (push INFILE.STACK (CONS IM.INFILE IM.INFILE.FILENAME)))
          (SETQ IM.INFILE.FILENAME NEW.FILENAME)
          (SETQ IM.INFILE NEW.FILE)
          (for X in IM.INFILE.NOTE.TAGS do (PUTPROP IM.INFILE.FILENAME X NIL])
```

(PARSE.ATOM

```
  [LAMBDA (SAV)                                               (* mjs "19-MAY-83 12:20")

          (* * note%: eventually, this should be modified so that it doesn't just throw away everything but the first atom, so that you
          can DEF things with multiple words (low priority))

    (CAR (PARSE.LIST SAV])
```

(PARSE.INDEX.SPEC

```
  [LAMBDA (SAV END.FLG)                                       (* mjs "11-AUG-83 15:26")

          (* * parses the argument to INDEX#TOPROG or REF#TOPROG, which may include an object type.
          For {index...}, the argument has the form "xxx xxx <object-type>" %.
          For {SectionRef ...} or {PageRef...}, the arg has the for "<object-type> xxx xxx" %.
          If END.FLG is true, the object-type is at the end of the arg. PARSE.INDEX.SPEC returns a list
          (<list of arg words> . <object-type>))

    (PROG ((BEGIN.PAREN (CHARCODE %())
           (END.PAREN (CHARCODE %)))
           (TYPE 'TERM)
           (LST NIL)
           TYPE.LST CHAR.LST TEMP (PARENS.TYPE.FLG NIL))
          (SETQ CHAR.LST (APPEND (CAR SAV)))
          (if END.FLG
              then (SETQ CHAR.LST (DREVERSE CHAR.LST))
                   (SETQ BEGIN.PAREN (CHARCODE %)))
                   (SETQ END.PAREN (CHARCODE %()))
          (while (MEMB (CAR CHAR.LST)
                       (CHARCODE (SPACE TAB CR)))
             do (SETQ CHAR.LST (CDR CHAR.LST)))
          (if (AND (EQ BEGIN.PAREN (CAR CHAR.LST))
                   (SETQ TEMP (MEMB END.PAREN CHAR.LST)))
              then (SETQ TYPE.LST (LDIFF (CDR CHAR.LST)
                                         TEMP))
                   (SETQ CHAR.LST (CDR TEMP))
                   (if END.FLG
                       then (SETQ TYPE.LST (REVERSE TYPE.LST)))
                   [SETQ TYPE (PARSE.LIST (CONS TYPE.LST (LAST TYPE.LST]
                   (if (NULL TYPE)
                       then (SETQ TYPE 'TERM))
                   (SETQ PARENS.TYPE.FLG T))
          (if END.FLG
              then (SETQ CHAR.LST (DREVERSE CHAR.LST)))
          [SETQ LST (PARSE.LIST (CONS CHAR.LST (LAST CHAR.LST]
          [if [AND (NOT PARENS.TYPE.FLG)
                   (SETQ TEMP (TRANSLATE.SPECIAL.TYPES (U-CASE (if END.FLG
                                                                    then (CAR (LAST LST))
                                                                  else (CAR LST]
              then (SETQ TYPE TEMP)
                   (SETQ LST (if END.FLG
                                 then (CAR (LASTN LST 1))
                               else (CDR LST]
          (RETURN (CONS LST TYPE])
```

(PARSE.LIST

```
  [LAMBDA (SAV PAREN.FLG)                                     (* mjs " 5-AUG-83 14:42")

          (* * if PAREN.FLG then the left and right paren chars parse as self-delimiting atoms)

    (if (NULL SAV)
        then NIL
      elseif (AND (LISTP SAV)
                  (LISTP (CDR SAV)))
        then [PROG [(CHARS (CAR SAV))
                    (ELIST NIL)
                    (CLIST NIL)
                    (DELIM.CHARS (if PAREN.FLG
                                     then (CHARCODE (SP TAB CR %( %)))
                                   else (CHARCODE (SP TAB CR]
                   [for X in CHARS when (SMALLP X)
                      do (if (FMEMB X DELIM.CHARS)
                             then (if CLIST
                                      then (SETQ ELIST (CONS (PACKC (DREVERSE CLIST))
                                                             ELIST))
                                           (SETQ CLIST NIL))
                                  (if [AND PAREN.FLG (FMEMB X (CHARCODE (%( %)]
                                      then (SETQ ELIST (CONS (CHARACTER X)
                                                                  ELIST)))
                                  else (SETQ CLIST (CONS X CLIST]
                          (RETURN (DREVERSE (if CLIST
                                                then (CONS (PACKC (DREVERSE CLIST))
                                                           ELIST)
                                              else ELIST]
           else NIL])
```

(PARSE.NUMBER.OR.PERCENTAGE

```
  [LAMBDA (SAV TOTAL IF.BAD.NUM)                                   (* mjs "20-APR-83 11:47")
    (PROG ((NUM (PARSE.ATOM SAV)))
          (if (NUMBERP NUM)
              then (RETURN (FIX NUM)))
          [if (STRPOS "PERCENT" (U-CASE NUM))
              then (SETQ NUM
                         (PACK (for X in (UNPACK NUM)
                                  while (MEMB X '(0 1 2 3 4 5 6 7 8 9 %.)) collect X]
          [if (NUMBERP NUM)
              then (RETURN (FIX (FTIMES 0.01 TOTAL NUM]
          (RETURN IF.BAD.NUM])
```

(PARSE.STRING

```
  [LAMBDA (SAV)                                                    (* mjs "20-APR-83 11:44")
    (if (NULL SAV)
        then ""
      elseif (AND (LISTP SAV)
                  (LISTP (CAR SAV)))
        then (CONCATLIST (for X in (CAR SAV) when (SMALLP X) collect (CHARACTER X)))
      else ""])
```

(PARSE.TO.NAME

```
  [LAMBDA NIL                                                      (* mjs "12-Jul-85 12:48")

              (* * PARSE.TO.NAME is called just after GOBBLE finds a left-bracket.
              It scans the file for the TO name which should follow. It also processes {Begin and {End constructs.
              Note%: this function will not scan past the end of the current possibly-included file, so that GOBBLE can move the pointer
              back if necessary.)

              (* * PARSE.TO.NAME returns%: (1) If the current word is not BEGIN or END, a litatom which is the upper-case version of
              the name, and leaves the file pointer to the next character after the name.
              (2) If the current word is BEGIN/END, a list of the form (BEGIN/END TONAME TAG.LIST) In this case, the file ptr is left after
              the closing }, or at the closing {, if one is found.)

    (PROG (CLIST ELIST NAME C)
          (SETQ CLIST NIL)
          (until (EOFP IM.INFILE) do (SETQ C (BIN IM.INFILE))
                                     (SETQ CLIST (CONS C CLIST))
             repeatwhile (TO.NAME.CHAR C))
          [COND
             ((NOT (TO.NAME.CHAR C))
              (SETQ CLIST (CDR CLIST))
              (SETFILEPTR IM.INFILE (SUB1 (GETFILEPTR IM.INFILE]
          [SETQ NAME (U-CASE (PACKC (DREVERSE CLIST]               (* if the scan was ternimated by end of file, print warning)
          (COND
             ((EOFP IM.INFILE)
              (IM.ERROR "TO Spec " (DESCRIBE.TO NIL NAME NIL)
                     " terminated by EOF" " --- auto-completed")))
          (COND
             ((AND (NEQ NAME 'BEGIN)
                   (NEQ NAME 'END))
              (RETURN NAME)))
          (SETQ ELIST NIL)
          (SETQ CLIST NIL)
          [until (EOFP IM.INFILE) do (SETQ C (BIN IM.INFILE))
                                     [COND
                                        [(FMEMB C (CHARCODE ({ } SP TAB CR)))
                                         (COND
                                            (CLIST (SETQ ELIST (CONS (U-CASE (PACKC (DREVERSE CLIST)))
                                                                     ELIST))
                                                   (SETQ CLIST NIL]
                                        (T (if (SMALLP C)
                                               then                (* ignore non-smallp chars in TO tag)
                                                    (SETQ CLIST (CONS C CLIST]
             repeatuntil (FMEMB C (CHARCODE ({ }]
          (SETQ ELIST (DREVERSE ELIST))
          (COND
             ((AND (EOFP IM.INFILE)
                   (EQ C (CHARCODE })))
              (IM.ERROR "TO Spec " (DESCRIBE.TO NAME (CAR ELIST)
                                          (CDR ELIST))
                     " terminated by EOF" " --- auto-completed")))
          [COND
             ((EQ C (CHARCODE {))                                 (* if {Begin or {End terminated by {, auto-complete,)
              (IM.ERROR "TO Spec " (DESCRIBE.TO NAME (CAR ELIST)
                                      (CDR ELIST))
                    " terminated by {" " --- auto-completed")
               (SETFILEPTR IM.INFILE (SUB1 (GETFILEPTR IM.INFILE]
           (RETURN (LIST NAME (CAR ELIST)
                         (CDR ELIST])
```

(**PARTITION.LIST**

```
  [LAMBDA (LST EQUAL.FN MAKE.TOKEN.FN SORT.TOKEN.FN)                   ; Edited  8-Dec-91 15:03 by jds

    ;; LST is a list of objects (X1 X2 X3 X4 ...);  PARTITION.LIST partitions them into sublists ((X1 X4 X7 ...) (X2 X56 ...) ...) where the members of
    ;; each sublist are 'equal' as determined by EQUAL.FN (default EQUAL), and returns the resulting list

    ;; if MAKE.TOKEN.FN is given, this is a function applied to each object to create 'tokens' , which are compared with EQUAL.FN .  This is done so
    ;; that MAKE.TOKEN.FN is done as few times as possible, which is nice if MAKE.TOKEN.FN is expensive.  For example, (PARTITION.LIST L
    ;; (QUOTE EQ) (QUOTE LENGTH)) is the same as (PARTITION L (QUOTE (LAMBDA (A B) (EQ (LENGTH A) (LENGTH B))))), but much less
    ;; expensive, since LENGTH is only computed once for each element of L

    ;; if SORT.TOKEN.FN is given, the list is sorted by the 'order' of the tokens as determined by essentially evaluating (SORT <tokenlist>
    ;; SORT.TOKEN.FN)

    (PROG ((NEWLIST NIL)
           SUBNEWLIST)
          (DECLARE (SPECVARS SORT.TOKEN.FN))
          (if (NULL EQUAL.FN)
              then (SETQ EQUAL.FN 'EQUAL))
          [if (NULL MAKE.TOKEN.FN)
              then (SETQ MAKE.TOKEN.FN (FUNCTION (LAMBDA (X)
                                                   X]
          [for OBJECT in LST bind OBJECT.TOKEN do (SETQ OBJECT.TOKEN (APPLY* MAKE.TOKEN.FN OBJECT))
                                                                       ; find first sublist in NEWLIST whose CAR is a token 'equal' to
                                                                       ; OBJECT.TOKEN
                                                  [SETQ SUBNEWLIST (for X in NEWLIST
                                                                      thereis (APPLY* EQUAL.FN OBJECT.TOKEN
                                                                                     (CAR X]
                                                  (if SUBNEWLIST
                                                      then             ; if there is such a sublist, put OBJECT second in list <after
                                                                       ; token>
                                                           (RPLACD SUBNEWLIST (CONS OBJECT (CDR SUBNEWLIST)))
                                                    else (SETQ NEWLIST (CONS (LIST OBJECT.TOKEN OBJECT)
                                                                             NEWLIST]
                                                                       ; if SORT.TOKEN.FN is given, sort the lists by the tokens
          [if SORT.TOKEN.FN
              then (SORT NEWLIST (FUNCTION (LAMBDA (X Y)
                                             (APPLY* SORT.TOKEN.FN (CAR X)
                                                    (CAR Y]            ; get rid of token at beginning of each sublist
          (for X in NEWLIST do (RPLNODE2 X (CDR X)))
          (RETURN NEWLIST])
```

(**PRINT.INFILE.NOTES**

```
  [LAMBDA NIL                                                          (* mjs "25-Jul-85 10:48")
    (for TYP in IM.INFILE.NOTE.TAGS bind LST when (SETQ LST (GETPROP IM.INFILE.FILENAME TYP))
       do (SETQ LST (SORT (INTERSECTION LST LST)))
          (IM.WARNING " ----------  " (SELECTQ TYP
                                          (UNDEF.FN "Undefined Functions")
                                          (UNBOUND.VAR "Unbound Variables")
                                          (IM.TAG "Tags")
                                          (IM.NO.REFS "Unresolved References")
                                          (MKSTRING TYP)))
          (for X in LST do (IM.WARNING "      " X))
       finally (IM.WARNING])
```

(**PUT.MY.PROP**

```
  [LAMBDA (PROPNAME VAL)                                               (* mjs "13-APR-83 10:30")
    (LISTPUT TO.PROP.LIST PROPNAME VAL])
```

(**SAVE.ARG**

```
  [LAMBDA (NO.SKIP.FLG)                                                (* mjs "21-APR-83 14:14")
    (PROG (SAVE)
          (if (fetch TO.ARGS of TO.NAME)
              then (if TO.ARG.NAME
                       then (SETQ SAVE (GOBBLE.SAVE NO.SKIP.FLG))
                            (SETQ TO.ARG.NAME NIL)
                     else (ERROR "TO PROG BUG: attempting to dump arg not yet gotten"))
            else (if (NOT TO.DONE.FLG)
                     then (SETQ SAVE (GOBBLE.SAVE NO.SKIP.FLG))
                          (SETQ TO.DONE.FLG T)
                   else (ERROR "TO PROG BUG: attempting to process unlabeled arg twice")))
          (RETURN SAVE])
```

(**SAVE.ARGS**

```
  [NLAMBDA GOOD.ARGS                                                   (* mjs "18-APR-83 16:55")

          (* * gets and saves args with the names on GOOD.ARGS until an unknown name comes up.
          If duplicate names come up, the later arg replaces the formar.)
```

```
    (PROG (ANAME (ARG.PROPLIST (CONS)))
          (if TO.DONE.FLG
              then (PUT.MY.PROP 'SAVE.ARGS.PROPLIST NIL)
                   (RETURN NIL))
      loop
          (SETQ ANAME (GET.ARG))
          (if (AND ANAME (FMEMB ANAME GOOD.ARGS))
              then (LISTPUT ARG.PROPLIST ANAME (SAVE.ARG))
                   (GO loop)
            else (PUT.MY.PROP 'SAVE.ARGS.PROPLIST (COND
                                                    ((CDR ARG.PROPLIST)
                                                     ARG.PROPLIST)
                                                    (T NIL)))
                 (RETURN ANAME])
```

(SAVE.ARGS.EMPTY

```
  [LAMBDA NIL                                          (* mjs "13-APR-83 10:29")
```

(* if called right after a call to SAVE.ARGS, this will return T if SAVE.ARGS did not find any args, and the TO has been closed.)

```
    (AND TO.DONE.FLG (NOT (GET.MY.PROP 'SAVE.ARGS.PROPLIST])
```

(SAVE.INFILE.NOTE

```
  [LAMBDA (TYP NAM)                                    (* mjs "29-Jul-85 16:29")
    (PUTPROP IM.INFILE.FILENAME TYP (CONS NAM (GETPROP IM.INFILE.FILENAME TYP])
```

(SCRUNCH.REFS

```
  [LAMBDA (REFS)                                       (* mjs "28-SEP-83 12:01")
    (PROG (REFS.BY.TYPE)
```

(* * REFS is list with elements of form%: (type text info section file fileptr))

(* * REFS.BY.TYPE is list with elements of form%: ((type text info section file fileptr)
(type text info section file fileptr) |...|) partioned by type, case-independent, sorted by ALPHORDER of U-CASE of type)

```
          [SETQ REFS.BY.TYPE (PARTITION.LIST REFS NIL [FUNCTION (LAMBDA (A)
                                                                  (U-CASE (CAR A]
                                  (FUNCTION (LAMBDA (A B)
                                              (LIST.ORDER A B]
          (RETURN
           (for REFS.ON.ONE.TYPE in REFS.BY.TYPE bind REFS.BY.FILE
              collect
```

(* REFS.BY.FILE is list of refs partioned by file, case-independent, sorted by ALPHORDER of U-CASE of file)

```
                (SETQ REFS.BY.FILE (PARTITION.LIST REFS.ON.ONE.TYPE NIL [FUNCTION (LAMBDA (A)
                                                                                (U-CASE (CAR (CDDDDR A]
                                        (FUNCTION ALPHORDER)))
                (CONS (if (NLISTP (CAAR REFS.ON.ONE.TYPE))
                          then (MKSTRING (L-CASE (CAAR REFS.ON.ONE.TYPE)
                                                 T))
                        else (LIST.TO.STRING (L-CASE (CAAR REFS.ON.ONE.TYPE)
                                                     T)))
                      (for REFS.ON.ONE.FILE in REFS.BY.FILE
                         collect (CONS [U-CASE (CAR (CDDDDR (CAR REFS.ON.ONE.FILE]
                                       (PROG [(SMALLREFS (for REF in REFS.ON.ONE.FILE
                                                            collect (LIST (if (MEMB '*PRIMARY* (CADDR REF))
                                                                              then 'Primary
                                                                            elseif (MEMB '*DEF* (CADDR REF))
                                                                              then 'Definition
                                                                            else NIL)
                                                                          (CADR (CDDDDR REF]
                                             [SORT SMALLREFS (FUNCTION (LAMBDA (A B)
                                                                         (IGEQ (CADR B)
                                                                               (CADR A]
                                             (RETURN (APPLY 'NCONC SMALLREFS])
```

(SEND.IMPLICIT

```
  [LAMBDA (NAME TYPE SAV)                              (* mjs " 5-AUG-83 13:55")
```

(* if IM.SEND.IMPLICIT is true and there are no references to this object in the index table, then send an implicit reference to this object)

```
    (if (AND IM.SEND.IMPLICIT (for X in (GETHASH NAME IMPTR.HASH) never (EQUAL (CAR X)
                                                                                TYPE)))
        then (if (U-CASEP NAME)
                 then (SEND.INFO NAME TYPE NIL '(*IMPLICIT*))
               else (SEND.INFO (U-CASE NAME)
                           TYPE NAME '(*IMPLICIT*])
```

```
(SKIP.WHITE.SPACE
  [LAMBDA NIL                                                    (* mjs "10-Apr-85 10:09")
    (PROG (X)
          [do (SETQ X (if (EOFP IM.INFILE)
                          then (IM.FILE.EOF)
                        else (BIN IM.INFILE)))
             repeatwhile (FMEMB X (CHARCODE (SPACE CR TAB]
          (COND
             ((NEQ X 'EOF)
              (SETFILEPTR IM.INFILE (SUB1 (GETFILEPTR IM.INFILE])

(STANDARD.DUMMY.TO.PROG
  [LAMBDA NIL                                                    (* mjs " 9-Apr-85 16:00")
    (IM.ERROR "Dummy TO - " TO.NAME)
    (DUMP.ARG])

(TAG.LIST.MATCH
  [LAMBDA (TLIST1 TLIST2)                                        (* mjs " 8-APR-83 15:56")
    (EQUAL (U-CASE TLIST1)
           (U-CASE TLIST2])

(TO.MATCH
  [LAMBDA (NAME BEGINFLG TAGLIST)                                (* mjs " 8-APR-83 15:58")
    (AND (EQ NAME TO.NAME)
         (EQ BEGINFLG TO.BEGIN.FLG)
         (OR (NOT BEGINFLG)
             (TAG.LIST.MATCH TAGLIST TO.TAG.LIST])

(TO.NAME.CHAR
  [LAMBDA (C)                                                    (* mjs "12-Jul-85 12:43")
                                                                 (* returns true if C is the char code of a legal TO name char)

    (AND (SMALLP C)
         (OR (AND (IGEQ C (CHARCODE A))
                  (ILEQ C (CHARCODE Z)))
             (AND (IGEQ C (CHARCODE a))
                  (ILEQ C (CHARCODE z)))
             (AND (IGEQ C (CHARCODE 0))
                  (ILEQ C (CHARCODE 9])

(TRANSLATE.SPECIAL.TYPES
  [LAMBDA (X)                                                    (* mjs " 9-NOV-83 12:17")
    (SELECTQ (U-CASE X)
        (FN '(Function))
        (VAR '(Variable))
        (PROP '(Property Name))
        (BREAKCOM '(Break Command))
        (EDITCOM '(Editor Command))
        (PACOM '(Prog. Asst. Command))
        (FILECOM '(File Package Command))
        ((LITATOM ATOM)
            '(Litatom))
        (ERROR '(Error Message))
        (TERM 'TERM)
        ((TAG FIGURE)
            'TAG)
        NIL])

(TRIVIAL.ARG
  [LAMBDA NIL                                                    (* mjs "21-APR-83 14:10")
    (if (fetch TO.ARGS of TO.NAME)
        then (if TO.ARG.NAME
                 then (GOBBLE.TRIVIAL)
                      (SETQ TO.ARG.NAME NIL)
               else (ERROR "TO PROG BUG: attempting to process ungotten arg "))
      else (if (NOT TO.DONE.FLG)
               then (GOBBLE.TRIVIAL)
                    (SETQ TO.DONE.FLG T)
             else (ERROR "TO PROG BUG: attempting to process unlabeled arg twice"])
)

(DECLARE%: EVAL@COMPILE

(ATOMRECORD TO (TO.PROG TO.ARGS TO.ARG.SYNONYMS TO.TYPE TO.ARG.TYPE))
)

(RPAQQ IM.INFILE.NOTE.TAGS (UNDEF.FN UNBOUND.VAR IM.TAG IM.NO.REFS IM.FIGURE BAD.TO.NAME))

(RPAQ? IM.NOTE.FLG NIL)

(RPAQ? IM.DRAFT.FLG NIL)
```

```
(RPAQ? IM.INDEX.FILE.FLG NIL)

(RPAQ? IM.REF.FLG NIL)

(RPAQ? IM.SEND.IMPLICIT NIL)

(RPAQ? IM.EVEN.FLG NIL)

(RPAQ? IM.CHECK.DEFS NIL)

(PUTPROPS DUMPOUT INFO EVAL)

(DECLARE%: EVAL@COMPILE

(PUTPROPS IM.ERROR MACRO [X `(IM.PRINT.MESSAGE "ERROR: " T %, (CONS 'LIST X])

(PUTPROPS IM.WARNING MACRO [X `(IM.PRINT.MESSAGE "  Warning:  " NIL %, (CONS 'LIST X])
)

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

(ADDTOVAR NLAMA SAVE.ARGS)

(ADDTOVAR NLAML GET.ARG.SAV)

(ADDTOVAR LAMA )
)

(PUTPROPS IMTRAN COPYRIGHT ("Xerox Corporation" 1983 1984 1985 1986 1991))
```

FUNCTION INDEX

VARIABLE INDEX

MACRO INDEX

PROPERTY INDEX

RECORD INDEX