File created: 7-Jan-91 17:36:04 **{DSK}<usr>bane>LISP>TEXTMODULES.;3**

changes to:
```
(IL:FUNCTIONS INSTALL-READ-MACRO DEFPRESENTATION PRINT-HASH-BASED-NUMBER READ-HASH-BASED-NUMBER
       TRANSLATE-HASH-BASED-NUMBER HANDLE-READ-MACROS PRINT-HASH-STAR READ-HASH-STAR
       TRANSLATE-HASH-STAR PRINT-READABLE-READ-TIME-CONDITIONAL
       PRINT-UNREADABLE-READ-TIME-CONDITIONAL TRANSLATE-READ-TIME-CONDITIONAL
       READ-READ-TIME-CONDITIONAL)
(IL:PRESENTATIONS HASH-BASED-NUMBER HASH-STAR HASH-IL-READABLE HASH-IL-UNREADABLE)
(IL:VARS IL:TEXTMODULESCOMS)
(IL:VARIABLES *SEDIT-READ-MACROS* *CONDITIONAL-KEYWORDS*)
(IL:STRUCTURES READ-TIME-CONDITIONAL)
```

previous date: 4-Dec-90 00:36:08 {DSK}<usr>bane>LISP>TEXTMODULES.;2

Read Table: XCL

Package: TEXTMODULES

Format: XCCS


```
(IL:RPAQQ IL:TEXTMODULESCOMS
          (
```

;;; TEXTMODULES, a text file to file manager conversion utility.

;; Top-level and top-level internal functions

```
(IL:FUNCTIONS ADD-FORM BEFORE-MAKE-TEXTMODULE-FUNCTIONS CONVERT-LOADED-FILES DEFINER-FILECOM
       DEFPRESENTATION EXPORT-DEFINERS FILECOM-SPECIFIER FORM-SPECIFIER HANDLE-READ-MACROS
       IMPORT-DEFINERS INSTALL-FORM INSTALL-READ-MACRO LOAD-TEXTMODULE MAKE-LISP-FILE-READTABLE
       MAKE-TEXTMODULE NAME-OF PARSE-ENVIRONMENT-SETUP-FILECOMS PRINT-ENVIRONMENT-FORMS PRINT-FILECOM
       PROCESS-COMS-AFTER-LOAD REMOVE-PRESENTATION SYMBOLS-TRANSLATE TOP-LEVEL-FORM
       TOP-LEVEL-FORM-FORM TOP-LEVEL-FORM-P TRANSLATE-FORM)
```

;; Support for semi-colon comments. Semicolon comments are special cased in this code, because rewriting the SEdit support for that ;; presentation would be hard.

```
(IL:FUNCTIONS ADJOIN-COMMENTS MAYBE-ADJOIN-COMMENTS MAYBE-UPGRADE-COMMENTS PRINT-COMMENT-LINE
       PRINT-COPYRIGHT-COMMENTS PRINT-SEMICOLON-COMMENT READ-HASH-BAR-COMMENT READ-SEMICOLON-COMMENT
       SEMICOLON-COMMENT-P UPGRADE-COMMENTS)
```

;; Support for #b #o #x #r

```
(IL:FUNCTIONS PRINT-HASH-BASED-NUMBER READ-HASH-BASED-NUMBER TRANSLATE-HASH-BASED-NUMBER)
```

;; Support for #*

```
(IL:FUNCTIONS PRINT-HASH-STAR READ-HASH-STAR TRANSLATE-HASH-STAR)
```

;; Support for #+ #-

```
(IL:FUNCTIONS PRINT-READABLE-READ-TIME-CONDITIONAL PRINT-UNREADABLE-READ-TIME-CONDITIONAL
       READ-READ-TIME-CONDITIONAL TRANSLATE-READABLE-RTC TRANSLATE-UNREADABLE-RTC KEYWORDIZE
       NON-KEYWORD?)
```

;; Support for #, #,

;; TRANSLATE-PREFIX-QUOTE is believed unnecessary now; check this...

```
(IL:FUNCTIONS PRINT-PREFIX-QUOTE READ-PREFIX-QUOTE TRANSLATE-PREFIX-QUOTE TRANSLATE-HASH-COMMA
       TRANSLATE-HASH-DOT)
```

;; Some functions used in the old implementation of #+/#-

```
(IL:FUNCTIONS PRINT-READ-TIME-CONDITIONAL)
(IL:STRUCTURES PRESENTATION PREFIX-QUOTE PRESENTATION-OPS READ-TIME-CONDITIONAL SEMICOLON-COMMENT
       SPECIFIER UNKNOWN-FORM UNKNOWN-SPECIFIER)
(IL:VARIABLES *CONDITIONAL-KEYWORDS* *CONVERT-LOADED-FILES* *UPGRADE-COMMENT-LENGTH* *JOIN-COMMENTS*
       *DEFDEFINER-MACROS* *DELETE-FORM* COMMENT-LEVEL-MARKERS EOF-MARKER *SEDIT-READ-MACROS*
       *SPECIFIERS*)
(IL:P (UNLESS (FIND-PACKAGE "EMPTY")
              (MAKE-PACKAGE "EMPTY" :USE NIL))
      (MAKE-LISP-FILE-READTABLE))
```

;; PRESENTATIONS handing reading and printing of CL constructs

```
(IL:DEFINE-TYPES IL:PRESENTATIONS)
(IL:PRESENTATIONS HASH-BASED-NUMBER HASH-COMMA HASH-DOT HASH-IL-READABLE HASH-IL-UNREADABLE HASH-STAR
       )
(IL:ADVISE REMOVE-COMMENTS (IL:EVAL :IN IL:\\DO-DEFINE-FILE-INFO))
```

;; (IL:FILES IL:SEDIT-COMMONLISP)

```
(IL:PROP IL:ARGNAMES LOAD-TEXTMODULE MAKE-TEXTMODULE MAKE-SPECIFIER INSTALL-FORM FILECOM-SPECIFIER
       FORM-SPECIFIER ADD-FORM PRINT-FILECOM)
(IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
       IL:TEXTMODULES)))
```

;;; TEXTMODULES, a text file to file manager conversion utility.
;; Top-level and top-level internal functions

```
(DEFUN ADD-FORM (FORM FILECOMS &OPTIONAL (SPECIFIER (FORM-SPECIFIER FORM)))
```

```
   "Call appropriate functions to make definition editable, return new filecoms."
   (FUNCALL (SPECIFIER-ADD-FORM SPECIFIER)
          FORM FILECOMS))


(DEFUN BEFORE-MAKE-TEXTMODULE-FUNCTIONS (FILE STREAM)
   "Things to do before the main body of the textmodule is printed."
   (PRINT-COMMENT-LINE (GET FILE 'IL:MAKEFILE-ENVIRONMENT)
          STREAM)
   (PRINT-COPYRIGHT-COMMENTS FILE STREAM)
   (TERPRI STREAM)
   (PRINT-ENVIRONMENT-FORMS (GET FILE 'IL:MAKEFILE-ENVIRONMENT)
          STREAM)
   (TERPRI STREAM))


(DEFUN CONVERT-LOADED-FILES (FORM)
   "Looks for loaded files to convert in top-level forms."
   (DECLARE (SPECIAL *CONVERT-LOADED-FILES*))
   (WHEN (AND *CONVERT-LOADED-FILES* (MEMBER (CAR FORM)
                                             '(LOAD REQUIRE)
                                             :TEST
                                             'EQ)
              (IF (EQ *CONVERT-LOADED-FILES* :QUERY)
                  (Y-OR-N-P "Convert file loaded by ~s too?" FORM)
                  T))
         (CASE (CAR FORM)
             (LOAD (LOAD-TEXTMODULE (EVAL (SECOND FORM))))
             (REQUIRE (LOAD-TEXTMODULE (EVAL (THIRD FORM))))))
   FORM)


(DEFUN DEFINER-FILECOM (FORM)
   "Examines a form and returns its specifier (file command)."
   (GET (CAR FORM)
        ':DEFINER-FOR))

(DEFDEFINER DEFPRESENTATION IL:PRESENTATIONS (NAME &KEY FIELDS INCLUDE PRINT-FUNCTION (READ-MACRO NIL)
                                                        (TRANSLATOR #'(LAMBDA (PRESENTATION)
                                                                              (CERROR "Ignore the presentation"
                                                                                      "Untranslatable presentation
                                                                                      ~s" PRESENTATION))))
   "Define a presentation type."
   `(PROGN (DEFSTRUCT (,NAME (:INCLUDE ,@(IF (NULL INCLUDE)
                                             (LIST 'PRESENTATION)
                                             (ETYPECASE INCLUDE
                                                 (SYMBOL (LIST INCLUDE))
                                                 (LIST INCLUDE)))
                                         (OPS (MAKE-PRESENTATION-OPS :READ-MACRO ',READ-MACRO :TRANSLATOR
                                                                     ',TRANSLATOR)))
                             (:PRINT-FUNCTION ,PRINT-FUNCTION))
             ,@FIELDS)
           ,@(UNLESS (NULL READ-MACRO)
                 (LIST `(HANDLE-READ-MACROS ',READ-MACRO)))
           ',NAME))


(DEFUN EXPORT-DEFINERS (FORM)
   "Turn definers into macros."
   (IF (EQ (CAR FORM)
           'DEFDEFINER)
       (LET* ((CLEANED-FORM (REMOVE-COMMENTS FORM))
              (NAME (SECOND CLEANED-FORM))
              (DEFINER-FOR (THIRD CLEANED-FORM))
              (BODY (CDR (MEMBER DEFINER-FOR FORM))))
             `(DEFMACRO ,(IF (CONSP NAME)
                             (CAR NAME)
                             NAME) ,@BODY))
       FORM))


(DEFUN FILECOM-SPECIFIER (FILECOM)
   "Return the specifier for the filecom, otherwise warn."
   (OR (SOME #'(LAMBDA (SPECIFIER)
                       (AND (FUNCALL (SPECIFIER-FILECOM-P SPECIFIER)
                                     FILECOM)
                            SPECIFIER))
             *SPECIFIERS*)
       (IL:NILL (WARN 'UNKNOWN-SPECIFIER :SPECIFIER FILECOM))))


(DEFUN FORM-SPECIFIER (FORM)
   "Return the specifier for the form, otherwise warn."
   (OR (SOME #'(LAMBDA (SPECIFIER)
                       (AND (FUNCALL (SPECIFIER-FORM-P SPECIFIER)
                                     FORM)
                            SPECIFIER))
```

```
            *SPECIFIERS*)
       (IL:NILL (WARN 'UNKNOWN-FORM :FORM FORM))))
```

(DEFUN **HANDLE-READ-MACROS** (MAC)

;; Read macros defined by presentations need to be installed in the "LISP-FILE" readtable. In addition those marked :SEDIT need to be on the ;; *SEDIT-READ-MACROS* list. MAC can be a list of the form:

;; (<character> [<sub-character>] <read-macro-function> [:SEDIT]

;; or it can be a list of such lists.

```
   (IF (CONSP (FIRST MAC))                                    ; Handle nested macro specs
       (DOLIST (RM MAC)
           (HANDLE-READ-MACROS RM))
       (PROGN (INSTALL-READ-MACRO MAC (IL:FIND-READTABLE "LISP-FILE"))
              (WHEN (EQ :SEDIT (CAR (LAST MAC)))
                  (LET* ((KEY (IF (AND (CHARACTERP (FIRST MAC))
                                       (CHARACTERP (SECOND MAC)))
                                  (CONS (FIRST MAC)
                                        (SECOND MAC))
                                  (FIRST MAC)))
                         (DATA (IF (CHARACTERP KEY)
                                   (SECOND MAC)
                                   (THIRD MAC))))
                        (SETF (GETHASH KEY *SEDIT-READ-MACROS*)
                              DATA))))))
```

(DEFUN **IMPORT-DEFINERS** (FORM)

```
   "Change a macro to a definer if we've been told its OK."
   (DECLARE (SPECIAL *DEFDEFINER-MACROS*))
   (IF (AND (EQ (CAR FORM)
                'DEFMACRO)
            (MEMBER (CADR FORM)
                    *DEFDEFINER-MACROS* :TEST 'EQ))
       `(DEFDEFINER ,(CADR FORM) IL:FUNCTIONS ,@(CDDR FORM))
       FORM))
```

(DEFUN **INSTALL-FORM** (FORM &OPTIONAL (SPECIFIER (**FORM-SPECIFIER** FORM)))

```
   "Install a definition as current and executable."
   (WHEN (NOT (MEMBER IL:DFNFLG '(IL:PROP IL:ALLPROP)))
       (FUNCALL (SPECIFIER-INSTALL-FORM SPECIFIER)
              FORM)))
```

(DEFUN **INSTALL-READ-MACRO** (READ-MACRO TABLE)

```
   (COND
      ((AND (CHARACTERP (FIRST READ-MACRO))
            (CHARACTERP (SECOND READ-MACRO)))
       (MAKE-DISPATCH-MACRO-CHARACTER (FIRST READ-MACRO)
              T TABLE)
       (SET-DISPATCH-MACRO-CHARACTER (FIRST READ-MACRO)
              (SECOND READ-MACRO)
              (THIRD READ-MACRO)
              TABLE))
      ((CHARACTERP (FIRST READ-MACRO))
       (SET-MACRO-CHARACTER (FIRST READ-MACRO)
              (SECOND READ-MACRO)
              T TABLE))
      (T (ERROR "Bad read macro spec ~s" READ-MACRO))))
```

(DEFUN **LOAD-TEXTMODULE** (PATHNAME &KEY (MODULE (STRING-UPCASE (PATHNAME-NAME PATHNAME)))

```
                                                    ; Name of module which has these contents.
                                    (INSTALL T)     ; Install definitions as current?
                                    (PACKAGE (FIND-PACKAGE "USER"))
                                                    ; Package to read file in.
                                    (UPGRADE-COMMENT-LENGTH *UPGRADE-COMMENT-LENGTH*)
                                                    ; Change single to double semi at this length.
                                    (JOIN-COMMENTS *JOIN-COMMENTS*)
                                                    ; Smash together adjacent comments?
                                    (CONVERT-LOADED-FILES *CONVERT-LOADED-FILES*)
                                    (DEFDEFINER-MACROS *DEFDEFINER-MACROS*)
                                                    ; Names of macros that should become definers.

                                    )
   "Load a text file, creating a content description."
   (SETQ PATHNAME (MERGE-PATHNAMES PATHNAME ".LISP"))
   (ETYPECASE MODULE
       (STRING T)
       (SYMBOL (SETQ MODULE (SYMBOL-NAME MODULE))))
   (LET ((IL:DFNFLG (IF (NULL INSTALL)
                        'IL:PROP
                        INSTALL))
         (*PACKAGE* (IF (PACKAGEP PACKAGE)
                        PACKAGE
```

```
                    (FIND-PACKAGE PACKAGE)))
         (*READTABLE* (IL:FIND-READTABLE "LISP-FILE"))
         (*JOIN-COMMENTS* JOIN-COMMENTS)
         (*UPGRADE-COMMENT-LENGTH* UPGRADE-COMMENT-LENGTH)
         (*CONVERT-LOADED-FILES* CONVERT-LOADED-FILES)
         (*DEFDEFINER-MACROS* DEFDEFINER-MACROS)
         (FILECOMS NIL))
        (DECLARE (SPECIAL *PACKAGE* *READTABLE* *JOIN-COMMENTS* *UPGRADE-COMMENT-LENGTH* *CONVERT-LOADED-FILES*
                          *DEFDEFINER-MACROS*))
        (WITH-OPEN-FILE (STREAM PATHNAME :DIRECTION :INPUT)
               (LET (FORM)
                    (LOOP (SETQ FORM (READ STREAM NIL EOF-MARKER))
                          (MAYBE-ADJOIN-COMMENTS FORM)
                          (MAYBE-UPGRADE-COMMENTS FORM)
                          (WHEN (EQ FORM EOF-MARKER)
                                (RETURN NIL))
                          ;; JRB - This used to be ADD-FORM and then INSTALL-FORM; I believe it needs to be the other way around.
                          ;; Consider:
                          ;; (EVAL-WHEN (EVAL COMPILE LOAD)
                          ;;    (DEFMACRO MUMBLE (...))
                          ;;    (MUMBLE ...)
                          ;; where MUMBLE is on *DEFDEFINER-MACROS*.  The whole EVAL-WHEN goes on the coms before any of it gets
                          ;; installed, so the sub-mumbles don't get made into (FUNCTIONS ...) coms entries.
                          (LET ((SPECIFIER (FORM-SPECIFIER FORM)))
                               (INSTALL-FORM FORM SPECIFIER)
                               (SETQ FILECOMS (ADD-FORM FORM FILECOMS SPECIFIER))))))
        (MULTIPLE-VALUE-BIND (FILECOMS ENVIRONMENT)
               (PARSE-ENVIRONMENT-SETUP-FILECOMS FILECOMS)
               (MAYBE-ADJOIN-COMMENTS FILECOMS)
               (MAYBE-UPGRADE-COMMENTS FILECOMS)            ; Must be done AFTER the environment setup is parsed out, lest
                                                            ; the env comment be joined to the rest of the initial comments
                                                            ; and stripped.
               (SETQ FILECOMS (PROCESS-COMS-AFTER-LOAD FILECOMS))
               (LET* ((NAME (INTERN MODULE "INTERLISP"))
                      (FILEVAR (IL:FILECOMS NAME)))
                     (SETF (SYMBOL-VALUE FILEVAR)
                           FILECOMS)
                     (IL:ADDFILE NAME)
                     (SETF (GET NAME 'IL:FILETYPE)
                           :COMPILE-FILE)
                     (IL:ADDTOFILE `(,NAME IL:FILETYPE)
                            'IL:PROPS NAME)
                     (SETF (GET NAME 'IL:MAKEFILE-ENVIRONMENT)
                           ENVIRONMENT)
                     (IL:ADDTOFILE `(,NAME IL:MAKEFILE-ENVIRONMENT)
                            'IL:PROPS NAME)
                     NAME))))

(DEFUN MAKE-LISP-FILE-READTABLE ()
   "Build and name the LISP-FILE readtable."
   (LET ((TABLE (OR (IL:FIND-READTABLE "LISP-FILE")         ; If this is removed, the file cannot be loaded prop and continue
                                                            ; to work.  The LISP-FILE readtable will be initialized, but the
                                                            ; defpresentation forms not re-evaluated.
                    (COPY-READTABLE (IL:FIND-READTABLE "XCL")))) ; This has to be copied from XCL so that XAIE FIle Manager can
                                                            ; read and write the imported files!

         )
        (SET-MACRO-CHARACTER #\; #'READ-SEMICOLON-COMMENT NIL TABLE)
        (INSTALL-READ-MACRO '(#\# #\| READ-HASH-BAR-COMMENT)
               TABLE)
        (IL:READTABLEPROP TABLE 'IL:NAME "LISP-FILE")
        TABLE))

(DEFUN MAKE-TEXTMODULE (MODULE &KEY (TYPE ".LISP")
                                    (PATHNAME (MERGE-PATHNAMES MODULE (MERGE-PATHNAMES TYPE)))
                                    (FILECOMS (SYMBOL-VALUE (IL:FILECOMS MODULE)))
                                    (WIDTH 80))
   "Write a text file based on the file manager file."
   (SETQ MODULE (ETYPECASE MODULE
                    (STRING MODULE)
                    (SYMBOL (SYMBOL-NAME MODULE))))
   (LET ((*PACKAGE* (FIND-PACKAGE "USER"))
         (*READTABLE* (IL:FIND-READTABLE "LISP-FILE"))
         (*PRINT-BASE* *PRINT-BASE*)
         (*PRINT-CASE* :DOWNCASE)
         (*PRINT-ARRAY* T)
         (*PRINT-LEVEL* NIL)
         (*PRINT-LENGTH* NIL)
         (*PRINT-STRUCTURE* T)
         (*PRINT-PRETTY* T)
         (IL:*PRINT-SEMICOLON-COMMENTS* T)
         (IL:FONTCHANGEFLG NIL)
```

```
         (IL:\#RPARS NIL)
         (IL:**COMMENT**FLG NIL)
         (FILE (IL:MKATOM MODULE)))
        (DECLARE (SPECIAL IL:*PRINT-SEMICOLON-COMMENTS* FILE))
        (WITH-OPEN-FILE (STREAM PATHNAME :DIRECTION :OUTPUT :IF-EXISTS :NEW-VERSION)
               (IL:LINELENGTH WIDTH STREAM)                          ; For Interlisp prettyprinter.
               (BEFORE-MAKE-TEXTMODULE-FUNCTIONS (INTERN MODULE "INTERLISP")
                      STREAM)
               (DOLIST (FILECOM FILECOMS)
                   (FRESH-LINE STREAM)
                   (LET ((TYPE (FILECOM-SPECIFIER FILECOM)))
                        (WHEN TYPE (PRINT-FILECOM FILECOM STREAM TYPE))))))
   PATHNAME)


(DEFUN NAME-OF (FORM)
   (FUNCALL (GET (CAR FORM)
                 ':DEFINITION-NAME)
          (REMOVE-COMMENTS FORM)))


(DEFUN PARSE-ENVIRONMENT-SETUP-FILECOMS (CONTENTS)
   "Parse out any environment specifiers, returning the reduced contents list and an environment object."
   ;; If you change anything in here you must change the printer in print-environment-forms.
   (WHEN (AND (SEMICOLON-COMMENT-P (FIRST CONTENTS))
              (EQL 0 (SEARCH "-*-" (SEMICOLON-COMMENT-STRING (FIRST CONTENTS)))))
                                                                    ; Discard EMACS comment line
       (POP CONTENTS))
   (LET ((PACKAGE-FORM NIL)                                         ; Collects the package setup forms.
         (BASE 10)                                                  ; Default.
         (COMMENT-FORMS NIL)                                        ; Comments to be pushed onto the front of the coms.

         )
        ;; Most of the mechanism below handles comments between the setup forms in the filecoms.  CONTENTS names the last parsed position.
        ;; NEXT-TOP-LEVEL-FORM slides NEXT-TAIL past the comments to the next top-level form.  WHEN-RECOGNIZED checks the form and if
        ;; recognized pops the in-between comments onto COMMENT-FORMS, others onto PACKAGE-FORM.
        (LET ((NEXT-TAIL CONTENTS)                                  ; Contains tail at next top-level form.
              FORM                                                  ; Contains next top level form.

              )
             (BLOCK PARSE-COMPLETE
                 (FLET ((NEXT-TOP-LEVEL-FORM NIL
                                ;; Find tail containing the next top level form.
                                (LOOP (WHEN (NULL NEXT-TAIL)
                                            (RETURN NIL))
                                      (LET ((HEAD (FIRST NEXT-TAIL)))
                                           (COND
                                              ((TOP-LEVEL-FORM-P HEAD)
                                               (SETQ FORM (TOP-LEVEL-FORM-FORM HEAD))
                                               (WHEN (NOT (READ-TIME-CONDITIONAL-P FORM))
                                                     (RETURN NIL)))
                                              ((SEMICOLON-COMMENT-P HEAD)
                                               NIL)
                                              (T (RETURN-FROM PARSE-COMPLETE NIL))))
                                      (POP NEXT-TAIL)))
                         (POP-FORMS NIL
                                ;; Comments between CONTENTS and (not including) NEXT-TAIL are popped onto COMMENT-FORMs.  The
                                ;; form in NEXT-TAIL is discarded and CONTENTS is updated.
                                (LOOP (WHEN (EQ CONTENTS NEXT-TAIL)
                                            (RETURN NIL))
                                      (PUSH (POP CONTENTS)
                                            COMMENT-FORMS))))
                        (MACROLET ((WHEN-RECOGNIZED (TEST &BODY FORMS)
                                ;; Find the next top level form.  Use TEST to recognize whether its an environment setup form.  Then
                                ;; execute the body and discard the processed form.
                                    `(PROGN (NEXT-TOP-LEVEL-FORM)
                                            (WHEN ,TEST
                                                (POP-FORMS)
                                                ,@FORMS
                                                (POP NEXT-TAIL)
                                                (SETQ CONTENTS NEXT-TAIL)))))
                                ;; package setup forms
                                (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                     'PROVIDE)
                                      (PUSH FORM PACKAGE-FORM))
                                (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                     'IN-PACKAGE)
                                      (PUSH FORM PACKAGE-FORM))
                                (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                     'SHADOW)
                                      (PUSH (SYMBOLS-TRANSLATE FORM)
```

```
                                    PACKAGE-FORM))
                              (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                   'EXPORT)
                                     (PUSH (SYMBOLS-TRANSLATE FORM)
                                    PACKAGE-FORM))
                              (WHEN-RECOGNIZED (MEMBER (FIRST FORM)
                                                       '(REQUIRE IL:FILESLOAD)
                                                       :TEST
                                                       #'EQ)
                                     (PUSH FORM PACKAGE-FORM))
                              (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                   'USE-PACKAGE)
                                     (PUSH FORM PACKAGE-FORM))
                              (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                   'IMPORT)
                                     (PUSH (SYMBOLS-TRANSLATE FORM)
                                    PACKAGE-FORM))
                              (WHEN-RECOGNIZED (EQ (FIRST FORM)
                                                   'SHADOWING-IMPORT)
                                     (PUSH (SYMBOLS-TRANSLATE FORM)
                                    PACKAGE-FORM))

                              ;; read-base

                              (WHEN-RECOGNIZED (AND (EQ (FIRST FORM)
                                                        'SETF)
                                                    (EQ (SECOND FORM)
                                                        '*READ-BASE*))
                                     (SETQ BASE (THIRD FORM)))))))

        ;; Return the new contents and a environment.

        (VALUES (APPEND (NREVERSE COMMENT-FORMS)
                        CONTENTS)
                `(:READTABLE "LISP-FILE" :PACKAGE ,(IF PACKAGE-FORM
                                                       `(LET ((*PACKAGE* *PACKAGE*))
                                                             ,@(NREVERSE PACKAGE-FORM)
                                                             *PACKAGE*)
                                                       "USER")
                        :BASE
                        ,BASE))))


(DEFUN PRINT-ENVIRONMENT-FORMS (ENVIRONMENT STREAM)
   "Print the environment initializing forms from ENVIRONMENT onto STREAM."
   (MACROLET
    ((PRINT-AND-EVAL (FORM STREAM)
            `(LET ((FORM ,FORM))
                  (LET ((*PACKAGE* (FIND-PACKAGE "EMPTY")))      ; This allows IMPORT, SHADOW, etc. statements to work,
                                                                 ; although it increases verbosity...
                       (PPRINT FORM ,STREAM))
                  (EVAL FORM))))
     (DO ((TAIL ENVIRONMENT (CDDR TAIL)))
         ((NULL TAIL))
       (LET
        ((NAME (FIRST TAIL))
         (VALUE (SECOND TAIL)))
        (ECASE NAME
            (:READTABLE )
            (:PACKAGE
               (TYPECASE VALUE
                   (NULL (ERROR "NIL given as package name"))
                   ((OR SYMBOL STRING) (PRINT-AND-EVAL `(IN-PACKAGE ,VALUE)
                                                       STREAM))
                   (CONS
                      (CASE (FIRST VALUE)
                          (DEFPACKAGE                            ; We only cover the portable options to defpackage.  Note that
                                                                 ; they're converted once but not back.
                             (PRINT-AND-EVAL
                              `(IN-PACKAGE ,(STRING (SECOND VALUE))
                                      ,@(LET ((NICKNAMES (CDR (ASSOC :NICKNAMES (CDDR VALUE)))))
                                             (WHEN NICKNAMES
                                                 `(:NICKNAMES ',NICKNAMES))))
                              STREAM)
                             (MAPC #'(LAMBDA (OPTION FUNCTION)
                                            (LET ((VALUE (CDR (ASSOC OPTION (CDDR VALUE)))))
                                                 (WHEN VALUE
                                                     (PRINT-AND-EVAL `(,FUNCTION ',VALUE)
                                                                     STREAM))))
                                   '(:SHADOW :EXPORT :USE :IMPORT :SHADOWING-IMPORT)
                                   '(SHADOW EXPORT USE-PACKAGE IMPORT SHADOWING-IMPORT)))
                          (LET                                   ; A fancy LET environment!
                             (MAPCAR #'(LAMBDA (FORM)
                                               (PRINT-AND-EVAL FORM STREAM))
                                     (BUTLAST (CDDR VALUE))      ; Avoid the LET, its bindings and the returned *package*.

                                     ))))
                   (T (ERROR "Unknown package specifier in environment ~s" VALUE))))
            (:BASE (PRINT-AND-EVAL `(SETF *READ-BASE* ,VALUE)
```

```
                         STREAM)))))))

(DEFUN PRINT-FILECOM (FILECOM STREAM &OPTIONAL (SPECIFIER (FILECOM-SPECIFIER FILECOM)))
   "Gets the print form of a specifier."
   (FUNCALL (SPECIFIER-PRINT-FILECOM SPECIFIER)
          FILECOM STREAM))


(DEFUN PROCESS-COMS-AFTER-LOAD (CONTENTS)
   "Destructively optimize COMS; compress adjacent definers, p, files, evert redundant COMS."
   (DO ((TAIL CONTENTS))
       ((NULL TAIL)
        CONTENTS)
     (LET ((ONE (FIRST TAIL)))
          (COND
             ((AND (CDR TAIL)
                   (OR (GET (FIRST ONE)
                            ':DEFINED-BY)
                       (MEMBER (FIRST ONE)
                               '(IL:FILES IL:P)))
                   (EQ (FIRST ONE)
                       (FIRST (SECOND TAIL))))                    ; Adjacent coms of same type.
              (LET ((END-OF-ONE (LAST ONE))                       ; Last cell in first com.
                    (HEAD-OF-TWO (CDR (SECOND TAIL))))            ; The tail of the next com.
                   (RPLACD END-OF-ONE HEAD-OF-TWO)                ; Append next tail onto current.
                   (RPLACD TAIL (CDDR TAIL))                      ; Splice OUT second com.

                   ))
             ((EQ 'IL:COMS (FIRST ONE))                           ; Descend into IL:COMS
              (RPLACD ONE (PROCESS-COMS-AFTER-LOAD (CDR ONE)))
              (WHEN (NULL (CDDR ONE))                             ; Remove redundant COMS enclosures.
                  (RPLACA TAIL (SECOND ONE))                      ; Replace the form with its contents.

                  )
              (POP TAIL))
             ((OR (EQ 'IL:EVAL-WHEN (FIRST ONE))
                  (EQ 'EVAL-WHEN (FIRST ONE)))                    ; Descend into EVAL-WHENs
              (RPLACD (CDR ONE)
                      (PROCESS-COMS-AFTER-LOAD (CDDR ONE)))
              (POP TAIL))
             (T (POP TAIL))))))


(DEFUN REMOVE-PRESENTATION (SEQUENCE INDEX)
   "Translates a presentation by removing it."
   (COND
      ((EQL INDEX 0)
       (SUBSEQ SEQUENCE 1))
      ((EQL INDEX (1- (LENGTH SEQUENCE)))
       (SUBSEQ SEQUENCE 0 INDEX))
      (T (CONCATENATE (IF (LISTP SEQUENCE)
                          'LIST
                          (TYPE-OF SEQUENCE))
                (SUBSEQ SEQUENCE 0 INDEX)
                (SUBSEQ SEQUENCE (1+ INDEX))))))


(DEFUN SYMBOLS-TRANSLATE (FORM)
   `(,(FIRST FORM)
     ,(LET ((NONLOCAL-SYMBOLS NIL)                                ; These are symbols defined elsewhere.
            (STRINGIFY-SYMBOLS NIL)                               ; These are symbols accessible in the current package.

            )
           (DOLIST (SYMBOL (EVAL (TRANSLATE-FORM (SECOND FORM))))
               (COND
                  ((EQ SYMBOL (FIND-SYMBOL (SYMBOL-NAME SYMBOL)))
                   (PUSH (SYMBOL-NAME SYMBOL)
                         STRINGIFY-SYMBOLS))
                  (T (PUSH SYMBOL NONLOCAL-SYMBOLS))))
           (IF (NULL NONLOCAL-SYMBOLS)
               `(MAPCAR #'INTERN ',STRINGIFY-SYMBOLS)
               `(APPEND ',NONLOCAL-SYMBOLS (MAPCAR #'INTERN ',STRINGIFY-SYMBOLS))))
     ,@(CDDR FORM)))


(DEFMACRO TOP-LEVEL-FORM (&BODY FORMS)
   "Wrapped around top level forms to install presentations."
   `(PROGN ,@(TRANSLATE-FORM FORMS)))


(DEFUN TOP-LEVEL-FORM-FORM (PLACE)
   "Return the form in the top-level form specifier."
   ;; JRB - when PLACE is a (P (FOO) (BAR)...) expression, turn it into a PROGN (should really rework the guts of the converter to handle each clause
   ;; seperately, but...)
   (FLET ((STRIP (FORM)
```

```
                (IF (EQ (FIRST FORM)
                        'TOP-LEVEL-FORM)
                    (SECOND FORM)
                    FORM)))
         (IF (CDDR PLACE)
             `(PROGN ,@(MAPCAR #'STRIP (CDR PLACE)))
             (STRIP (SECOND PLACE)))))


(DEFUN TOP-LEVEL-FORM-P (SPECIFIER)
   (EQ 'IL:P (FIRST SPECIFIER)))


(DEFUN TRANSLATE-FORM (SEQUENCE)
   "Create an evaluable form from one with presentations in it."
   (COND
      ((LISTP SEQUENCE)
       (SETQ SEQUENCE (COPY-LIST SEQUENCE))                    ; An optimization for lists, since it would be terrible to ELT into
                                                               ; them at each position.
       (DO ((TAIL SEQUENCE)
            (LAST NIL))
           ((NOT (CONSP TAIL))
            SEQUENCE)
         (LET ((HEAD (FIRST TAIL)))
              (COND
                 ((SEMICOLON-COMMENT-P HEAD)                   ; Special case for old style comments.
                  (IF (NULL LAST)
                      (SETQ SEQUENCE (CDR TAIL))
                      (RPLACD LAST (CDR TAIL)))                ; Last stays the same in either case.
                  (POP TAIL))
                 ((PRESENTATION-P HEAD)
                  (LET* ((INSTALLER (PRESENTATION-OPS-TRANSLATOR (PRESENTATION-OPS HEAD)))
                         (RESULT (IF (EQ INSTALLER :DELETE)
                                     *DELETE-FORM*
                                     (FUNCALL INSTALLER HEAD))))
                        (COND
                           ((EQ RESULT *DELETE-FORM*)
                            (IF (NULL LAST)
                                (SETQ SEQUENCE (CDR TAIL))
                                (RPLACD LAST (CDR TAIL)))      ; Last stays the same in either case.
                            (POP TAIL))
                           (T (RPLACA TAIL RESULT)
                              (SETQ LAST TAIL)
                              (POP TAIL)))))
                 ((TYPEP HEAD 'SEQUENCE)
                  (RPLACA TAIL (TRANSLATE-FORM HEAD))
                  (SETQ LAST TAIL)
                  (POP TAIL))
                 (T (SETQ LAST TAIL)
                    (POP TAIL))))))
      ((AND (NOT (STRINGP SEQUENCE))
            (TYPEP SEQUENCE 'SEQUENCE))                        ; Optimization: avoid strings.
       (SETQ SEQUENCE (COPY-SEQ SEQUENCE))                     ; The general case of a sequence.
       (DO ((INDEX 0)
            (LENGTH (LENGTH SEQUENCE)))
           ((EQL INDEX LENGTH)
            SEQUENCE)
         (LET ((HEAD (ELT SEQUENCE INDEX)))
              (COND
                 ((PRESENTATION-P HEAD)
                  (LET* ((INSTALLER (PRESENTATION-OPS-TRANSLATOR (PRESENTATION-OPS HEAD)))
                         (RESULT (IF (EQ INSTALLER :DELETE)
                                     *DELETE-FORM*
                                     (FUNCALL INSTALLER HEAD))))
                        (COND
                           ((EQ RESULT *DELETE-FORM*)
                            (SETQ SEQUENCE (REMOVE-PRESENTATION SEQUENCE INDEX))
                            (DECF LENGTH))
                           (T (SETF (ELT SEQUENCE INDEX)
                                    RESULT)
                              (INCF INDEX)))))
                 ((TYPEP HEAD 'SEQUENCE)
                  (SETF (ELT SEQUENCE INDEX)
                        (TRANSLATE-FORM HEAD))
                  (INCF INDEX))
                 (T (INCF INDEX))))))
      ((PRESENTATION-P SEQUENCE)
       (LET* ((INSTALLER (PRESENTATION-OPS-TRANSLATOR (PRESENTATION-OPS SEQUENCE)))
              (RESULT (IF (EQ INSTALLER :DELETE)
                          *DELETE-FORM*
                          (FUNCALL INSTALLER SEQUENCE))))
             (IF (EQ RESULT *DELETE-FORM*)
                 NIL
                 RESULT)))
      (T SEQUENCE)))
```

;; Support for semi-colon comments. Semicolon comments are special cased in this code, because rewriting the SEdit support for that presentation

```
;; would be hard.

(DEFUN ADJOIN-COMMENTS (FORM)
   "Smashes same type comments together.  No return value."
   (COND
      ((NOT (LISTP FORM)))
      ((OR (NULL FORM)
           (NULL (CDR FORM)))                                    ; Zero or one element

       )
      (T                                                         ; CONSP form = T
         (LET ((HEAD FORM)
               (FIRST NIL)
               (SECOND NIL))
              (LOOP (UNLESS (CONSP (CDR HEAD))                   ; Dotted lists
                        (RETURN))
                    (SETQ FIRST (FIRST HEAD))
                    (SETQ SECOND (SECOND HEAD))
                    (COND
                       ((AND (SEMICOLON-COMMENT-P FIRST)
                             (SEMICOLON-COMMENT-P SECOND)
                             (EQ (SEMICOLON-COMMENT-MARKER FIRST)
                                 (SEMICOLON-COMMENT-MARKER SECOND)))
                        ;; Smash second onto first
                        (SETF (SEMICOLON-COMMENT-STRING FIRST)
                              (CONCATENATE 'STRING (SEMICOLON-COMMENT-STRING FIRST)
                                     " "
                                     (SEMICOLON-COMMENT-STRING SECOND)))
                        ;; Delete cell from list.
                        (RPLACD HEAD (CDDR HEAD)))
                       (T ;; Recurse
                          (ADJOIN-COMMENTS FIRST)
                          ;; Continue
                          (SETQ HEAD (CDR HEAD))))
                    (WHEN (NULL HEAD)
                          (RETURN)))))))

(DEFUN MAYBE-ADJOIN-COMMENTS (CONTENTS)
   (DECLARE (SPECIAL *JOIN-COMMENTS*))
   (WHEN *JOIN-COMMENTS*
       (SETQ CONTENTS (ADJOIN-COMMENTS CONTENTS)))
   CONTENTS)

(DEFUN MAYBE-UPGRADE-COMMENTS (FORM)
   "Depending on setting of *upgrade-comment-length* we upgrade comments in this form."
   (IF (NUMBERP *UPGRADE-COMMENT-LENGTH*)
       (UPGRADE-COMMENTS FORM)
       FORM))

(DEFUN PRINT-COMMENT-LINE (ENVIRONMENT STREAM)
   "Prints a mode line onto the STREAM based on the ENVIRONMENT."
   (FORMAT STREAM ";;; -*- Mode: LISP")
   (DO ((TAIL ENVIRONMENT (CDDR TAIL)))
       ((NULL TAIL))
     (LET ((NAME (FIRST TAIL))
           (VALUE (SECOND TAIL)))
          (CASE NAME
              (:READTABLE )
              (:PACKAGE (COND
                           ((STRINGP VALUE)
                            (FORMAT STREAM "; Package: ~a" VALUE))
                           ((EQ (FIRST VALUE)
                                'DEFPACKAGE)
                            (FORMAT STREAM "; Package: (~a (~{~a ~}) 1000)" (STRING (SECOND VALUE))
                                   (OR (MAPCAR #'STRING (CDR (ASSOC :USE (CDDR VALUE))))
                                       (LIST "LISP"))))
                           ((EQ (FIRST VALUE)
                                'LET)
                            (LET ((FORM (ASSOC 'IN-PACKAGE (CDDR VALUE))))
                                 (FORMAT STREAM "; Package: (~a (~{~a ~}) 1000)" (STRING (SECOND FORM))
                                        (OR (MAPCAR #'STRING (CDR (GETF FORM :USE NIL)))
                                            (LIST "LISP")))))
                           (T (ERROR "Unknown package specifier in environment ~s" VALUE))))
              (:BASE (FORMAT STREAM "; Base: ~a" VALUE)))))
   (FORMAT STREAM " -*-")
   (TERPRI STREAM))

(DEFUN PRINT-COPYRIGHT-COMMENTS (ROOT-NAME STREAM)
```

```
   (LET ((OWNER (GET ROOT-NAME 'IL:COPYRIGHT)))
        (WHEN (AND OWNER (CONSP OWNER))
            (FORMAT STREAM ";;; Copyright (c) ")
            (DO ((TAIL (CDR OWNER)
                       (CDR TAIL)))
                ((NULL TAIL))
              (FORMAT STREAM "~4d" (CAR TAIL))
              (IF (CDR TAIL)
                  (PRINC ", " STREAM)))
            (FORMAT STREAM " by ~a~&" (CAR OWNER)))))
```

```
(DEFUN PRINT-SEMICOLON-COMMENT (FORM STREAM)
   "Print a semicolon comment.  Depends on IL:*PRINT-SEMICOLON-COMMENTS* being true."
   (WRITE FORM :STREAM STREAM))
```

```
(DEFUN READ-HASH-BAR-COMMENT (STREAM SUB-CHAR INTEGER)
   "Read the characters of a hash bar comment, creating a comment object."
   (WHEN INTEGER (WARN "Spurious integer argument to hash macro ignored."))
   (LET (PEEK-CHAR (COMMENT-BUFFER (MAKE-ARRAY 1024 :ELEMENT-TYPE 'CHARACTER :FILL-POINTER 0 :ADJUSTABLE T)))
        (LOOP (SETQ SUB-CHAR (READ-CHAR STREAM NIL EOF-MARKER))
              (WHEN (EQ SUB-CHAR EOF-MARKER)
                  (RETURN (MAKE-SEMICOLON-COMMENT :MARKER 'IL:\| :STRING COMMENT-BUFFER)))
              (WHEN (EQL SUB-CHAR #\|)
                  (SETQ PEEK-CHAR (PEEK-CHAR NIL STREAM NIL EOF-MARKER))
                  (WHEN (EQL PEEK-CHAR #\#)
                      (READ-CHAR STREAM NIL EOF-MARKER)
                      (RETURN (MAKE-SEMICOLON-COMMENT :MARKER 'IL:\| :STRING COMMENT-BUFFER))))
              (VECTOR-PUSH-EXTEND SUB-CHAR COMMENT-BUFFER))))
```

```
(DEFUN READ-SEMICOLON-COMMENT (STREAM DISP-CHAR &AUX CHAR
                                                     ; Current character.
                                     (LEVEL 0)       ; Comment level.
                                     (STARTING T)    ; In semicolons?
                                     (COMMENT-BUFFER (MAKE-ARRAY 128 :ELEMENT-TYPE 'CHARACTER
                                                                     :FILL-POINTER 0 :ADJUSTABLE T)))
   "Reads the characters of a comment, building a Xerox Lisp style comment."
   ;; Adjacent comments of the same level are smashed together during an after-read pass over the structure.  Another pass upgrades long single
   ;; semi-colon comments to double...
   (LOOP (SETQ CHAR (READ-CHAR STREAM NIL EOF-MARKER))
         (WHEN (OR (EQL CHAR EOF-MARKER)
                   (EQL CHAR #\Newline))
             (SETQ LEVEL (MIN LEVEL (1- (LENGTH COMMENT-LEVEL-MARKERS))))
             (RETURN (MAKE-SEMICOLON-COMMENT :MARKER (ELT COMMENT-LEVEL-MARKERS LEVEL)
                                             :STRING COMMENT-BUFFER)))
         (IF STARTING
             (SETQ STARTING (COND
                                ((EQL CHAR #\;)
                                 (INCF LEVEL))
                                (T (IF (NOT (EQL CHAR #\Space))   ; Ignore a single space after semicolons, save others.
                                       (VECTOR-PUSH-EXTEND CHAR COMMENT-BUFFER))
                                   NIL)))
             (VECTOR-PUSH-EXTEND CHAR COMMENT-BUFFER))))
```

```
(DEFUN SEMICOLON-COMMENT-P (FORM)
   "Is FORM a semicolon comment?"
   ;; All info about the structure of semicolon comments is encapsulated in this function and the semicolon-comment structure.
   (AND (CONSP FORM)
        (EQ (FIRST FORM)
            'IL:*)
        (MEMBER (SECOND FORM)
                COMMENT-LEVEL-MARKERS :TEST #'EQ)
        (STRINGP (THIRD FORM))
        (NULL (NTHCDR 3 FORM))))
```

```
(DEFUN UPGRADE-COMMENTS (FORM)
   "Smash long single semicolon comments into double semies.  No return value."
   ;; Should only be called if *UPGRADE-COMMENT-LENGTH* is a number!
   (WHEN (CONSP FORM)
       (DO ((TAIL FORM (CDR TAIL)))
           ((NOT (CONSP TAIL))                                   ; Dotted lists

            )
         (LET ((FORM (FIRST TAIL)))
              (COND
                 ((AND (SEMICOLON-COMMENT-P FORM)
                       (> (LENGTH (SEMICOLON-COMMENT-STRING FORM))
                          *UPGRADE-COMMENT-LENGTH*))
                  (SETF (SEMICOLON-COMMENT-MARKER FORM)
                        (NTH 1 COMMENT-LEVEL-MARKERS)))
```

```
                ((CONSP FORM)
                 (UPGRADE-COMMENTS FORM)))))))
```

;; Support for #b #o #x #r

```
(DEFUN PRINT-HASH-BASED-NUMBER (OBJECT STREAM DEPTH)
   (CASE (HASH-BASED-NUMBER-BASE OBJECT)
       ((2 8 16) (FORMAT STREAM "#~A~VR" (CASE (HASH-BASED-NUMBER-BASE OBJECT)
                                                ;; Using the atoms here looks a little warped, but it makes this print method obey
                                                ;; *print-case* for free...
                                                (2 'IL:B)
                                                (8 'IL:O)
                                                (16 'IL:X))
                         (HASH-BASED-NUMBER-BASE OBJECT)
                         (HASH-BASED-NUMBER-NUMBER OBJECT)))
       (OTHERWISE
          (UNLESS (< 2 (HASH-BASED-NUMBER-BASE OBJECT)
                     37)
              (ERROR "Bogus base in ~R presentation: ~d" (HASH-BASED-NUMBER-BASE OBJECT)))
          (FORMAT STREAM "#~DR~VR" (HASH-BASED-NUMBER-BASE OBJECT)
                  (HASH-BASED-NUMBER-BASE OBJECT)
                  (HASH-BASED-NUMBER-NUMBER OBJECT)))))

(DEFUN READ-HASH-BASED-NUMBER (STREAM SUB-CHAR ARG)
   (LET ((RBASE (ECASE SUB-CHAR
                       ((#\b #\B) 2)
                       ((#\o #\O) 8)
                       ((#\x #\X) 16)
                       ((#\r #\R)
                          (UNLESS (< 2 ARG 37)
                                  (ERROR "Bogus base in ~R: ~d" ARG))
                          ARG))))
        (MAKE-HASH-BASED-NUMBER :BASE RBASE :NUMBER (LET ((*READ-BASE* RBASE))
                                                         (READ STREAM)))))

(DEFUN TRANSLATE-HASH-BASED-NUMBER (OBJECT)
   (HASH-BASED-NUMBER-NUMBER OBJECT))
```

;; Support for #*

```
(DEFUN PRINT-HASH-STAR (OBJECT STREAM DEPTH)
   (PRINC "#*" STREAM)
   (MAP NIL #'(LAMBDA (B)
                      (PRINC (IF (ZEROP B)
                                 "0"
                                 "1")
                             STREAM))
        (HASH-STAR-VECTOR OBJECT)))

(DEFUN READ-HASH-STAR (STREAM SUB-CHAR ARG)
   (MAKE-HASH-STAR :VECTOR (IL:HASH-STAR STREAM SUB-CHAR ARG)))

(DEFUN TRANSLATE-HASH-STAR (OBJECT)
   (HASH-STAR-VECTOR OBJECT))
```

;; Support for #+ #-

```
(DEFUN PRINT-READABLE-READ-TIME-CONDITIONAL (OBJECT STREAM DEPTH)
   "Form was read as a string, so print it with PRIN1"
   (LET ((*PACKAGE* IL:*KEYWORD-PACKAGE*))
        (FORMAT STREAM "#~a~s " (READ-TIME-CONDITIONAL-SIGN OBJECT)
                (READ-TIME-CONDITIONAL-FEATURE OBJECT)))
   (PRIN1 (READ-TIME-CONDITIONAL-FORM OBJECT)
          STREAM))

(DEFUN PRINT-UNREADABLE-READ-TIME-CONDITIONAL (OBJECT STREAM DEPTH)
   "Form was read as a string, so print it with PRINC"
   (LET ((*PACKAGE* IL:*KEYWORD-PACKAGE*))
        (FORMAT STREAM "#~a~s " (READ-TIME-CONDITIONAL-SIGN OBJECT)
                (READ-TIME-CONDITIONAL-FEATURE OBJECT)))
   (PRINC (READ-TIME-CONDITIONAL-FORM OBJECT)
          STREAM))

(DEFUN READ-READ-TIME-CONDITIONAL (STREAM SUB-CHAR INTEGER)
   (WHEN INTEGER (WARN "Spurious integer argument to hash macro ignored."))
   (LET* ((FEATURE (LET ((*PACKAGE* IL:*KEYWORD-PACKAGE*))
```

```
                    (READ STREAM)))
          (UNREAD-P (ECASE SUB-CHAR
                    (#\- (IL:CMLREAD.FEATURE.PARSER FEATURE))
                    (#\+ (NOT (IL:CMLREAD.FEATURE.PARSER FEATURE)))))
          (FORM (COND
                  (UNREAD-P (LET ((START (FILE-POSITION STREAM)))
                              (LET ((*READ-SUPPRESS* T)
                                    (*READTABLE* (IL:FIND-READTABLE "XCL")))
                                   (DECLARE (SPECIAL *READ-SUPPRESS* *READTABLE*))
                                   (READ STREAM))
                              (LET ((LENGTH (- (FILE-POSITION STREAM)
                                               START)))
                                   (FILE-POSITION STREAM START)
                                   (LET ((BUFFER (MAKE-STRING LENGTH)))
                                        (DOTIMES (I LENGTH BUFFER)
                                            (SETF (SVREF BUFFER I)
                                                  (READ-CHAR STREAM)))))))
                  (T (LET ((FORM (LIST (READ STREAM))))
                          (LOOP (WHEN (NOT (SEMICOLON-COMMENT-P FORM))
                                      (RETURN (IF (EQL 1 (LENGTH FORM))
                                                  (FIRST FORM)
                                                  `(PROGN ,@(NREVERSE FORM)))))
                                (PUSH (READ STREAM)
                                      FORM)))))))
        (FUNCALL (IF UNREAD-P
                     #'MAKE-HASH-IL-UNREADABLE
                     #'MAKE-HASH-IL-READABLE)
                 :FEATURE FEATURE :SIGN SUB-CHAR :FORM FORM)))
```

(DEFUN **TRANSLATE-READABLE-RTC** (OBJECT)

;; Check out the features, just in case someone accidentally put a non-keyword in there

```
(WHEN (AND *CONDITIONAL-KEYWORDS* (NON-KEYWORD? (READ-TIME-CONDITIONAL-FEATURE OBJECT)))
    (CERROR "Make all symbols keywords" "~s contains a non-keyword" (READ-TIME-CONDITIONAL-FEATURE OBJECT))
    (SETF (READ-TIME-CONDITIONAL-FEATURE OBJECT)
          (KEYWORDIZE (READ-TIME-CONDITIONAL-FEATURE OBJECT))))
```

;; For paranoia's sake, we check the feature status again as we translate, in case someone changed the *FEATURES* list behind our back.

```
(IF (ECASE (READ-TIME-CONDITIONAL-SIGN OBJECT)
        (#\+ (IL:CMLREAD.FEATURE.PARSER (READ-TIME-CONDITIONAL-FEATURE OBJECT)))
        (#\- (NOT (IL:CMLREAD.FEATURE.PARSER (READ-TIME-CONDITIONAL-FEATURE OBJECT)))))
    (READ-TIME-CONDITIONAL-FORM OBJECT)
    *DELETE-FORM*))
```

(DEFUN **TRANSLATE-UNREADABLE-RTC** (OBJECT)

;; There just might be something other than a string in this unreadable read-time-conditional; check it out and try to fix it if it's not a string.

```
(UNLESS (STRINGP (READ-TIME-CONDITIONAL-FORM OBJECT))
    (CERROR "Replace it with (FORMAT NIL \"~~s\") " "Non-string ~s found in an unreadable
           read-time-conditional" (READ-TIME-CONDITIONAL-FORM OBJECT))
    (SETF (READ-TIME-CONDITIONAL-FORM OBJECT)
          (FORMAT NIL "~s" (READ-TIME-CONDITIONAL-FORM OBJECT))))
```

;; Check out the features, just in case someone accidentally put a non-keyword in there

```
(WHEN (AND *CONDITIONAL-KEYWORDS* (NON-KEYWORD? (READ-TIME-CONDITIONAL-FEATURE OBJECT)))
    (CERROR "Make all symbols keywords" "~s contains a non-keyword" (READ-TIME-CONDITIONAL-FEATURE OBJECT))
    (SETF (READ-TIME-CONDITIONAL-FEATURE OBJECT)
          (KEYWORDIZE (READ-TIME-CONDITIONAL-FEATURE OBJECT))))
```

;; For paranoia's sake, we check the feature status again as we translate, in case someone changed the *FEATURES* list behind our back.

```
(IF (ECASE (READ-TIME-CONDITIONAL-SIGN OBJECT)
        (#\+ (IL:CMLREAD.FEATURE.PARSER (READ-TIME-CONDITIONAL-FEATURE OBJECT)))
        (#\- (NOT (IL:CMLREAD.FEATURE.PARSER (READ-TIME-CONDITIONAL-FEATURE OBJECT)))))
    (WITH-INPUT-FROM-STRING (S (READ-TIME-CONDITIONAL-FORM OBJECT))
           (LET ((F (IL:NLSETQ (READ S))))
                (IF F
                    (CAR F)
                    (PROGN (IL:PRINTOUT IL:PROMPTWINDOW T "Warning: Problem trying to read conditional
                                  expression.  Not read.")
                           *DELETE-FORM*))))
    *DELETE-FORM*))
```

(DEFUN **KEYWORDIZE** (X)

```
(COND
   ((CONSP X)
    (MAPCAR #'KEYWORDIZE X))
   ((AND X (SYMBOLP X))
    (IF (KEYWORDP X)
        X
        (LET ((*PACKAGE* (FIND-PACKAGE "KEYWORD")))
             (WITH-INPUT-FROM-STRING (S (SYMBOL-NAME X))
                    (READ S)))))
   (T X)))
```

```
(DEFUN NON-KEYWORD? (X)
   (COND
      ((CONSP X)
       (SOME #'NON-KEYWORD? X))
      ((SYMBOLP X)
       (NOT (KEYWORDP X)))
      (T)))
```

;; Support for #, #,

;; TRANSLATE-PREFIX-QUOTE is believed unnecessary now; check this...

```
(DEFUN PRINT-PREFIX-QUOTE (OBJECT STREAM DEPTH)
   (IF (EQ *PRINT-CASE* :DOWNCASE)
       (PRINC (PREFIX-QUOTE-PREFIX OBJECT)
              STREAM)
       (PRINC (STRING-UPCASE (PREFIX-QUOTE-PREFIX OBJECT))
              STREAM))
   (PRIN1 (PREFIX-QUOTE-CONTENTS OBJECT)
          STREAM))

(DEFUN READ-PREFIX-QUOTE (STREAM SUB-CHAR INTEGER)
   "Reads hash quoted forms."
   (WHEN INTEGER (WARN "Spurious integer argument to hash macro ignored."))
   (FUNCALL (ECASE SUB-CHAR
                (#\. #'MAKE-HASH-DOT)
                (#\, #'MAKE-HASH-COMMA)
                ((#\O #\o) #'MAKE-HASH-O)
                ((#\X #\x) #'MAKE-HASH-X)
                ((#\B #\b) #'MAKE-HASH-B))
            :CONTENTS
            (LET ((*READ-BASE* (ECASE SUB-CHAR
                                      ((#\. #\,) *READ-BASE*)
                                      ((#\B #\b) 2)
                                      ((#\O #\o) 8)
                                      ((#\X #\x) 16))))
                 (READ STREAM NIL T))))

(DEFUN TRANSLATE-PREFIX-QUOTE (OBJECT)                    ; This only has to handle numeric base types.
   (PREFIX-QUOTE-CONTENTS OBJECT))

(DEFUN TRANSLATE-HASH-COMMA (OBJECT)
   (COND
      (*READ-SUPPRESS* NIL)
      (COMPILER::*COMPILER-IS-READING* (COMPILER::MAKE-EVAL-WHEN-LOAD :FORM (PREFIX-QUOTE-CONTENTS OBJECT)))
      ((IL:FETCH (READTABLEP IL:COMMONLISP) IL:OF *READTABLE*)
       (EVAL (PREFIX-QUOTE-CONTENTS OBJECT)))
      (T (IL:EVAL (PREFIX-QUOTE-CONTENTS OBJECT)))))

(DEFUN TRANSLATE-HASH-DOT (OBJECT)
   (COND
      (*READ-SUPPRESS* NIL)
      ((IL:FETCH (READTABLEP IL:COMMONLISP) IL:OF *READTABLE*)
       (EVAL (PREFIX-QUOTE-CONTENTS OBJECT)))
      (T (IL:EVAL (PREFIX-QUOTE-CONTENTS OBJECT)))))
```

;; Some functions used in the old implementation of #+/#-

```
(DEFUN PRINT-READ-TIME-CONDITIONAL (OBJECT STREAM DEPTH)
   (PRINC #\# STREAM)
   (ETYPECASE OBJECT
       (HASH-PLUS (PRINC #\+ STREAM))
       (HASH-MINUS (PRINC #\- STREAM)))
   (LET ((*PACKAGE* IL:*KEYWORD-PACKAGE*))
        (PRIN1 (READ-TIME-CONDITIONAL-FEATURE OBJECT)
               STREAM))
   (PRINC " " STREAM)
```

;; JRB - I don't 100% understand why the conditionalization on UNREAD-P is needed here; I DO know, however, that it's causing a conditional
;; expression containing a string to lose big time when I dump a file...

;; (IF (READ-TIME-CONDITIONAL-UNREAD-P OBJECT) (PRINC (READ-TIME-CONDITIONAL-FORM OBJECT) STREAM) (PRIN1
;; (READ-TIME-CONDITIONAL-FORM OBJECT) STREAM))

```
   (PRIN1 (READ-TIME-CONDITIONAL-FORM OBJECT)
          STREAM))

(DEFSTRUCT PRESENTATION
   OPS)
```

```
(DEFSTRUCT (PREFIX-QUOTE (:INCLUDE PRESENTATION)
                         (:PRINT-FUNCTION PRINT-PREFIX-QUOTE))
   TYPE
   PREFIX
   CONTENTS)


(DEFSTRUCT (PRESENTATION-OPS (:TYPE LIST))
   READ-MACRO                                          ; A list with one or two characters followed by a read macro
                                                       ; function.  Installed in the text file readtable to read this
                                                       ; presentation.
   TRANSLATOR                                          ; Either a function on PRESENTATION which translates it, or
                                                       ; :DELETE which always removes it (eg, comments).

   )

(DEFSTRUCT (READ-TIME-CONDITIONAL (:INCLUDE PRESENTATION)
                                  (:PRINT-FUNCTION PRINT-READ-TIME-CONDITIONAL))
   FEATURE
   SIGN
   FORM)

(DEFSTRUCT (SEMICOLON-COMMENT (:TYPE LIST)
                              (:PREDICATE NIL)         ; The real one is SEMICOLON-COMMENT-P

                              )
   (TAG 'IL:*)
   (MARKER 'IL:\;)
   (STRING ""))

(DEFSTRUCT (SPECIFIER (:TYPE LIST))
   NAME                                                ; A string naming the specifier.
   FILECOM-P                                           ; Predicate on FILECOM, answers true if this is the specifier for
                                                       ; this filecom.
   FORM-P                                              ; Predicate on FORM (a form from the text file), answers true if
                                                       ; this is the specifier for the definition in FORM.
   ADD-FORM                                            ; Function of FORM and FILECOMS which adds a specifier for
                                                       ; FORM to the FILECOMS.
   INSTALL-FORM                                        ; Function of a FORM which installs the definition of FORM (may
                                                       ; remove presentations).  Should not actually install the definition
                                                       ; if il:dfnflg is il:prop or il:allprop.
   PRINT-FILECOM                                       ; Function of FILECOM and STREAM which prettyprints a form
                                                       ; onto stream representing the filecom.

   )

(DEFINE-CONDITION UNKNOWN-FORM (WARNING)
   (FORM)
   (:REPORT (LAMBDA (CONDITION STREAM)
                  (FORMAT STREAM "Can't find specifier for form ~s" (UNKNOWN-FORM-FORM CONDITION)))))

(DEFINE-CONDITION UNKNOWN-SPECIFIER (WARNING)
   (SPECIFIER)
   (:REPORT (LAMBDA (CONDITION STREAM)
                  (FORMAT STREAM "Unrecognized filecom ~s" (UNKNOWN-SPECIFIER-SPECIFIER CONDITION)))))

(DEFVAR *CONDITIONAL-KEYWORDS* T
   "Controls whether TEXTMODULES insists on keywords in features of read-time-conditionals")

(DEFPARAMETER *CONVERT-LOADED-FILES* T
   "Convert text files loaded by the first one.")

(DEFPARAMETER *UPGRADE-COMMENT-LENGTH* 40
   "Length at which a single semicolon comment is upgraded to double.")

(DEFPARAMETER *JOIN-COMMENTS* T
   "Should comments be joined together when read?")

(DEFPARAMETER *DEFDEFINER-MACROS* NIL
   "Names of macros to change to definers on read.")

(DEFVAR *DELETE-FORM* "<delete form marker>")

(DEFCONSTANT COMMENT-LEVEL-MARKERS '(IL:\; IL:|;;| IL:|;;;| IL:|;;;;| IL:\|)
```

```
                                        "Comment markers for availible levels.")

(DEFCONSTANT EOF-MARKER "eof"
   "Unique object passed through read at EOF.")

(DEFVAR *SEDIT-READ-MACROS* (MAKE-HASH-TABLE :TEST #'EQUAL)
                            "Presentation read macro entries that need to be added to SEdit Common Lisp
                            readtables")

(DEFPARAMETER *SPECIFIERS*
   (LIST
    (MAKE-SPECIFIER :NAME "Comment" :FILECOM-P #'SEMICOLON-COMMENT-P :FORM-P #'SEMICOLON-COMMENT-P :ADD-FORM
           #'(LAMBDA (FORM FILECOMS)
                     (APPEND FILECOMS (LIST FORM)))
           :INSTALL-FORM
           #'IDENTITY :PRINT-FILECOM #'PPRINT)
    (MAKE-SPECIFIER
     :NAME "eval-when top level form" :FILECOM-P #'(LAMBDA (FILECOM)
                                                           (EQ (FIRST FILECOM)
                                                               'EVAL-WHEN))
     :FORM-P
     #'(LAMBDA (FORM)
               (AND (LISTP FORM)
                    (EQ (FIRST FORM)
                        'EVAL-WHEN)))
     :ADD-FORM
     #'(LAMBDA (FORM FILECOMS)
               (APPEND FILECOMS (LIST `(EVAL-WHEN ,(SECOND FORM)
                                                  ,@(LET ((FILECOMS NIL))
                                                         (MAPC #'(LAMBDA (FORM)
                                                                         (SETQ FILECOMS (ADD-FORM FORM FILECOMS)))
                                                               (CDDR FORM))
                                                         FILECOMS)))))
     :INSTALL-FORM
     #'(LAMBDA (FORM)
               (WHEN (MEMBER 'EVAL (SECOND FORM))
                     (DOLIST (FORM (CDDR FORM))
                             (INSTALL-FORM FORM))))
     :PRINT-FILECOM
     #'(LAMBDA (FILECOM STREAM)
               (TERPRI STREAM)
               (PRINC "(eval-when " STREAM)
               (PRIN1 (SECOND FILECOM)
                      STREAM)
               (DOLIST (FILECOM (CDDR FILECOM))
                       (FRESH-LINE STREAM)
                       (PRINT-FILECOM FILECOM STREAM))
               (FRESH-LINE STREAM)
               (PRINC ")" STREAM)))
    (MAKE-SPECIFIER :NAME "Definer" :FILECOM-P #'(LAMBDA (FILECOM)
                                                         (GET (FIRST FILECOM)
                                                              ':DEFINED-BY))
           :FORM-P
           #'(LAMBDA (FORM)
                     (AND (LISTP FORM)
                          (GET (CAR FORM)
                               ':DEFINER-FOR)))
           :ADD-FORM
           #'(LAMBDA (FORM FILECOMS)
                     (SETQ FORM (IMPORT-DEFINERS FORM))
                     (LET ((IL:DFNFLG 'IL:PROP))
                          (EVAL FORM))
                     (APPEND FILECOMS (LIST `(,(DEFINER-FILECOM FORM)
                                              ,(NAME-OF FORM)))))
           :INSTALL-FORM
           #'(LAMBDA (FORM)
                     (SETQ FORM (IMPORT-DEFINERS FORM))
                     (LET ((IL:DFNFLG T))
                          (EVAL FORM)))
           :PRINT-FILECOM
           #'(LAMBDA (FILECOM STREAM)
                     (LET ((TYPE (FIRST FILECOM)))
                          (DOLIST (NAME (REST FILECOM))
                              (FRESH-LINE STREAM)
                              (COND
                                 ((SEMICOLON-COMMENT-P NAME)
                                  (PRINT-FILECOM NAME STREAM))
                                 ((IL:GETDEF NAME TYPE NIL '(IL:NOERROR))
                                  (PPRINT (EXPORT-DEFINERS (IL:GETDEF NAME TYPE))
                                          STREAM))
                                 (T (WARN "Unrecognised drek in ~S filecom ignored:~%~s" TYPE NAME)))))))
    (MAKE-SPECIFIER
     :NAME "Group of definitions (COMS)" :FILECOM-P #'(LAMBDA (FILECOM)
                                                              (EQ (CAR FILECOM)
```

```
                                                                'IL:COMS))
      :FORM-P
      #'(LAMBDA (FORM)
                (EQ (CAR FORM)
                    'PROGN))
      :ADD-FORM
      #'(LAMBDA (FORM FILECOMS)
                (APPEND FILECOMS (LIST `(IL:COMS ,@(LET ((FILECOMS NIL))
                                                        (MAPC #'(LAMBDA (FORM)
                                                                        (SETQ FILECOMS (ADD-FORM FORM FILECOMS)))
                                                              (CDR FORM))
                                                        FILECOMS)))))
      :INSTALL-FORM
      #'(LAMBDA (FORM)
                (DOLIST (FORM (CDR FORM))
                        (INSTALL-FORM FORM)))
      :PRINT-FILECOM
      #'(LAMBDA (FILECOM STREAM)
                (DOLIST (FILECOM (CDR FILECOM))
                        (FRESH-LINE STREAM)
                        (PRINT-FILECOM FILECOM STREAM))))
    (MAKE-SPECIFIER :NAME "Top-level read-time conditional" :FILECOM-P #'(LAMBDA (FORM)
                                                                                 NIL)
           :FORM-P
           #'READ-TIME-CONDITIONAL-P :ADD-FORM #'(LAMBDA (FORM FILECOMS)
                                                         (APPEND FILECOMS
                                                                 (LIST `(IL:P (TOP-LEVEL-FORM ,FORM)))))
           :INSTALL-FORM
           #'(LAMBDA (FORM)
                     (EVAL (TRANSLATE-FORM FORM)))
           :PRINT-FILECOM
           #'(LAMBDA (FILECOM STREAM)
                     (PPRINT (TOP-LEVEL-FORM-FORM FILECOM)
                             STREAM)))
    (MAKE-SPECIFIER :NAME "VARS com translator" :FILECOM-P #'(LAMBDA (FILECOM)
                                                                     (EQ (CAR FILECOM)
                                                                         'IL:VARS))
           :FORM-P
           #'(LAMBDA (FORM)
                     NIL)
           :PRINT-FILECOM
           #'(LAMBDA (FILECOM STREAM)
                     (FLET ((TRANSLATE-VARS (FILECOM)
                                    (ETYPECASE FILECOM
                                        (SYMBOL `(DEFPARAMETER ,FILECOM ',(SYMBOL-VALUE FILECOM)))
                                        (LIST `(DEFPARAMETER ,(FIRST FILECOM) ,(IF (REST FILECOM)
                                                                                   (SECOND FILECOM)
                                                                                   NIL))))))
                            (DOLIST (SINGLE (REST FILECOM))
                                (FRESH-LINE STREAM)
                                (PPRINT (TRANSLATE-VARS SINGLE)
                                        STREAM)))))
    (MAKE-SPECIFIER :NAME "INITVARS com translator" :FILECOM-P #'(LAMBDA (FILECOM)
                                                                         (EQ (CAR FILECOM)
                                                                             'IL:INITVARS))
           :FORM-P
           #'(LAMBDA (FORM)
                     NIL)
           :PRINT-FILECOM
           #'(LAMBDA (FILECOM STREAM)
                     (FLET ((TRANSLATE-INITVARS (FILECOM)
                                    (ETYPECASE FILECOM
                                        (SYMBOL `(DEFVAR ,FILECOM NIL))
                                        (LIST `(DEFVAR ,(FIRST FILECOM) ,(IF (REST FILECOM)
                                                                             (SECOND FILECOM)
                                                                             NIL))))))
                            (DOLIST (SPEC (REST FILECOM))
                                (FRESH-LINE STREAM)
                                (PPRINT (TRANSLATE-INITVARS SPEC)
                                        STREAM)))))
    (MAKE-SPECIFIER :NAME "CONSTANTS com translator" :FILECOM-P #'(LAMBDA (FILECOM)
                                                                          (EQ (CAR FILECOM)
                                                                              'IL:CONSTANTS))
           :FORM-P
           #'(LAMBDA (FORM)
                     NIL)
           :PRINT-FILECOM
           #'(LAMBDA (FILECOM STREAM)
                     (FLET ((TRANSLATE-CONSTANTS (FILECOM)
                                    (ETYPECASE FILECOM
                                        (SYMBOL `(DEFCONSTANT ,FILECOM ',(SYMBOL-VALUE FILECOM)))
                                        (LIST `(DEFCONSTANT ,(FIRST FILECOM) ,(IF (REST FILECOM)
                                                                                  (SECOND FILECOM)
                                                                                  NIL))))))
                            (DOLIST (SPEC (REST FILECOM))
                                (FRESH-LINE STREAM)
                                (PPRINT (TRANSLATE-CONSTANTS SPEC)
```

```
                                    STREAM)))))
    (MAKE-SPECIFIER
     :NAME "PROPS com translator" :FILECOM-P #'(LAMBDA (FILECOM)
                                                 (EQ (CAR FILECOM)
                                                     'IL:PROPS))
     :FORM-P
     #'(LAMBDA (FORM)
               NIL)
     :PRINT-FILECOM
     #'(LAMBDA (FILECOM STREAM)
               (FLET ((PPRINT-PROPS (FILECOM)
                           (DECLARE (SPECIAL FILE))
                           (LET ((PROP (SECOND FILECOM))
                                 (SYMBOL (FIRST FILECOM)))
                                (IF (MEMBER PROP (SYMBOL-PLIST SYMBOL)
                                            :TEST
                                            'EQ)
                                    (UNLESS (AND (EQ FILE SYMBOL)
                                                 (MEMBER PROP '(IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
                                                        :TEST
                                                        'EQ))
                                        (PPRINT `(SETF (GET ',SYMBOL ',PROP)
                                                       ',(GET SYMBOL PROP))
                                                STREAM))
                                    (WARN "No ~s property for ~s~%" PROP SYMBOL)))))
                      (DOLIST (SPEC (REST FILECOM))
                          (FRESH-LINE STREAM)
                          (PPRINT-PROPS SPEC)))))
    (MAKE-SPECIFIER
     :NAME "PROP com translator" :FILECOM-P #'(LAMBDA (FILECOM)
                                                (EQ (CAR FILECOM)
                                                    'IL:PROP))
     :FORM-P
     #'(LAMBDA (FORM)
               (AND (LISTP FORM)
                    (EQ (FIRST FORM)
                        'SETF)
                    (LISTP (SECOND FORM))
                    (EQ (FIRST (SECOND FORM))
                        'GETF)
                    (EQL 3 (LENGTH (SECOND FORM)))))
     :ADD-FORM
     #'(LAMBDA (FORM FILECOMS)
               (APPEND FILECOMS (LIST `(IL:PROP ,(THIRD (SECOND FORM))
                                                ,(SECOND (SECOND FORM))))))
     :INSTALL-FORM
     #'(LAMBDA (FORM)
               (EVAL (TRANSLATE-FORM FORM)))
     :PRINT-FILECOM
     #'(LAMBDA (FILECOM STREAM)
               (POP FILECOM)
               (LET ((PROPS-SPEC (POP FILECOM)))
                    (FLET ((PPRINT-PROP (SYMBOL PROP)
                                (DECLARE (SPECIAL FILE))
                                (IF (MEMBER PROP (SYMBOL-PLIST SYMBOL)
                                            :TEST
                                            'EQ)
                                    (UNLESS (AND (EQ FILE SYMBOL)
                                                 (MEMBER PROP '(IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
                                                        :TEST
                                                        'EQ))
                                        (PPRINT `(SETF (GET ',SYMBOL ',PROP)
                                                       ',(GET SYMBOL PROP))
                                                STREAM))
                                    (WARN "No ~s property for ~s~%" PROP SYMBOL))))
                           (DOLIST (SYMBOL FILECOM)
                               (FRESH-LINE STREAM)
                               (COND
                                  ((EQ PROPS-SPEC 'IL:ALL)         ; Everything
                                   (DO ((TAIL (SYMBOL-PLIST SYMBOL)
                                              (CDDR TAIL)))
                                       ((NULL TAIL))
                                     (DECLARE (GLOBAL IL:SYSPROPS))
                                     (LET ((PROP (FIRST TAIL))
                                           (VALUE (SECOND TAIL)))
                                          (WHEN (NOT (MEMBER PROP IL:SYSPROPS :TEST 'EQ))
                                                (PPRINT-PROP SYMBOL PROP)))))
                                  ((LISTP PROPS-SPEC)
                                   (DOLIST (PROP PROPS-SPEC)
                                       (PPRINT-PROP SYMBOL PROP)))
                                  ((SYMBOLP PROPS-SPEC)
                                   (PPRINT-PROP SYMBOL PROPS-SPEC))
                                  (T (CERROR "Ignore property" "Bad prop spec ~s in PROPS com" PROPS-SPEC))))))))
    (MAKE-SPECIFIER
     :NAME "FILES com translator" :FILECOM-P #'(LAMBDA (FILECOM)
                                                 (EQ (CAR FILECOM)
                                                     'IL:FILES))
```

```
     :FORM-P
     #'(LAMBDA (FORM)
               NIL)
     :PRINT-FILECOM
     #'(LAMBDA (FILECOM STREAM)
               (POP FILECOM)
               (DO ((NOERROR NIL))
                   ((NULL FILECOM))
                  (FRESH-LINE STREAM)
                  (LET ((ITEM (CAR FILECOM)))
                       (ETYPECASE ITEM
                           (SYMBOL (PPRINT `(LOAD ',ITEM ,@(WHEN NOERROR
                                                                 `(:IF-DOES-NOT-EXIST NIL)))
                                           STREAM))
                           (STRING (PPRINT `(LOAD ,ITEM ,@(WHEN NOERROR
                                                                `(:IF-DOES-NOT-EXIST NIL)))
                                           STREAM))
                           (LIST (WHEN (MEMBER 'IL:NOERROR ITEM :TEST 'EQ)
                                       (SETQ NOERROR T))))))))
    (MAKE-SPECIFIER :NAME "Top level form" :FILECOM-P #'TOP-LEVEL-FORM-P :FORM-P #'TRUE :ADD-FORM
           #'(LAMBDA (FORM FILECOMS)
                     (CONVERT-LOADED-FILES FORM)
                     (APPEND FILECOMS (LIST `(IL:P (TOP-LEVEL-FORM ,FORM)))))
           :INSTALL-FORM
           #'(LAMBDA (FORM)
                     (EVAL (TRANSLATE-FORM FORM)))
           :PRINT-FILECOM
           #'(LAMBDA (FILECOM STREAM)
                     (LET ((FORM (TOP-LEVEL-FORM-FORM FILECOM)))
                          (FRESH-LINE STREAM)
                          (PPRINT FORM STREAM)
                          (WHEN (EQ 'IN-PACKAGE (FIRST FORM))
                                (EVAL FORM))))))
   "A list of all content specifier types for text files.")

(UNLESS (FIND-PACKAGE "EMPTY")
       (MAKE-PACKAGE "EMPTY" :USE NIL))
```

(**MAKE-LISP-FILE-READTABLE**)

;; PRESENTATIONS handing reading and printing of CL constructs

```
(DEF-DEFINE-TYPE IL:PRESENTATIONS "presentation types")

(DEFPRESENTATION HASH-BASED-NUMBER :FIELDS (BASE NUMBER)
   :PRINT-FUNCTION PRINT-HASH-BASED-NUMBER
   :READ-MACRO ((#\# #\B READ-HASH-BASED-NUMBER :SEDIT)
                (#\# #\O READ-HASH-BASED-NUMBER :SEDIT)
                (#\# #\X READ-HASH-BASED-NUMBER :SEDIT)
                (#\# #\R READ-HASH-BASED-NUMBER :SEDIT))
   :TRANSLATOR TRANSLATE-HASH-BASED-NUMBER)


(DEFPRESENTATION HASH-COMMA :INCLUDE (PREFIX-QUOTE (TYPE :HASH-COMMA)
                                                   (PREFIX "#,"))
   :PRINT-FUNCTION PRINT-PREFIX-QUOTE
   :READ-MACRO (#\# #\, READ-PREFIX-QUOTE)
   :TRANSLATOR TRANSLATE-HASH-COMMA)


(DEFPRESENTATION HASH-DOT :INCLUDE (PREFIX-QUOTE (TYPE :HASH-DOT)
                                                 (PREFIX "#."))
   :PRINT-FUNCTION PRINT-PREFIX-QUOTE
   :READ-MACRO (#\# #\. READ-PREFIX-QUOTE)
   :TRANSLATOR TRANSLATE-HASH-DOT)


(DEFPRESENTATION HASH-IL-READABLE :INCLUDE READ-TIME-CONDITIONAL
   :PRINT-FUNCTION PRINT-READABLE-READ-TIME-CONDITIONAL
   :READ-MACRO (#\# #\+ READ-READ-TIME-CONDITIONAL)
   :TRANSLATOR TRANSLATE-READABLE-RTC)


(DEFPRESENTATION HASH-IL-UNREADABLE :INCLUDE READ-TIME-CONDITIONAL
   :PRINT-FUNCTION PRINT-UNREADABLE-READ-TIME-CONDITIONAL
   :READ-MACRO (#\# #\- READ-READ-TIME-CONDITIONAL)
   :TRANSLATOR TRANSLATE-UNREADABLE-RTC)


(DEFPRESENTATION HASH-STAR :FIELDS (VECTOR)
   :PRINT-FUNCTION PRINT-HASH-STAR
   :READ-MACRO (#\# #\* READ-HASH-STAR :SEDIT)
   :TRANSLATOR TRANSLATE-HASH-STAR)

(REINSTALL-ADVICE 'REMOVE-COMMENTS :AROUND '((:LAST (TRANSLATE-FORM (CAR ARGLIST)))))
```

```
(REINSTALL-ADVICE '(IL:EVAL :IN IL:\\DO-DEFINE-FILE-INFO)
       :BEFORE
       '((:LAST (SETQ IL:U (TRANSLATE-FORM IL:U)))))

(IL:READVISE REMOVE-COMMENTS (IL:EVAL :IN IL:\\DO-DEFINE-FILE-INFO))
```

;; (IL:FILES IL:SEDIT-COMMONLISP)

```
(IL:PUTPROPS LOAD-TEXTMODULE IL:ARGNAMES (NIL (PATHNAME &KEY :MODULE :INSTALL :PACKAGE
                                                           :UPGRADE-COMMENT-LENGTH :JOIN-COMMENTS
                                                           :CONVERT-LOADED-FILES :DEFDEFINER-MACROS)))

(IL:PUTPROPS MAKE-TEXTMODULE IL:ARGNAMES (NIL (IL:MODULE &KEY TYPE PATHNAME IL:FILECOMS IL:WIDTH)))

(IL:PUTPROPS MAKE-SPECIFIER IL:ARGNAMES (NIL (&KEY :NAME :FILECOM-P :FORM-P :ADD-FORM :INSTALL-FORM
                                                   :PRINT-FILECOM)))

(IL:PUTPROPS INSTALL-FORM IL:ARGNAMES (NIL (IL:FORM &OPTIONAL IL:SPECIFIER)))

(IL:PUTPROPS FILECOM-SPECIFIER IL:ARGNAMES (NIL (IL:FILECOM)))

(IL:PUTPROPS FORM-SPECIFIER IL:ARGNAMES (NIL (IL:FORM)))

(IL:PUTPROPS ADD-FORM IL:ARGNAMES (NIL (IL:FORM IL:FILECOMS &OPTIONAL IL:SPECIFIER)))

(IL:PUTPROPS PRINT-FILECOM IL:ARGNAMES (NIL (IL:FILECOM STREAM &OPTIONAL IL:SPECIFIER)))

(IL:PUTPROPS IL:TEXTMODULES IL:FILETYPE :COMPILE-FILE)

(IL:PUTPROPS IL:TEXTMODULES IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE
                                                          (LET ((*PACKAGE* *PACKAGE*))
                                                               (IN-PACKAGE (DEFPACKAGE "TEXTMODULES"
                                                                                       (:USE "LISP" "XCL")
                                                                                       (:PREFIX-NAME "TM")
                                                                                       (:EXPORT "LOAD-TEXTMODULE"
                                                                                              "MAKE-TEXTMODULE"
                                                                                              "*SPECIFIERS*"
                                                                                              "MAKE-SPECIFIER"
                                                                                              "INSTALL-FORM"
                                                                                              "FORM-SPECIFIER"
                                                                                              "FILECOM-SPECIFIER"
                                                                                              "ADD-FORM"
                                                                                              "INSTALL-FORM"
                                                                                              "PRINT-FILECOM"
                                                                                              "*UPGRADE-COMMENT-LEN
GTH*" "*JOIN-COMMENTS*" "*CONVERT-LOADED-FILES*" "*DEFDEFINER-MACROS*")))
                                                               (IL:FILESLOAD IL:SEDIT-COMMONLISP)
                                                               *PACKAGE*)
                                                          :BASE 10))

(IL:PUTPROPS IL:TEXTMODULES IL:COPYRIGHT ("Xerox Corporation" 1987 1988 1989 1990 1991))
```

FUNCTION INDEX

VARIABLE INDEX

PROPERTY INDEX

STRUCTURE INDEX

PRESENTATION INDEX

ADVICE INDEX

CONSTANT INDEX

MACRO INDEX

DEFINE-TYPE INDEX

DEFINER INDEX
