File created: 8-Dec-2023 15:46:10 **{WMEDLEY}<sources>CLSTREAMS.;43**

edit by: rmk

changes to: (FUNCTIONS %BROADCAST-STREAM-DEVICE-CHARSETFN %CONCATENATED-STREAM-DEVICE-CHARSETFN)
(FNS %SYNONYM-STREAM-DEVICE-CHARSETFN %TWO-WAY-STREAM-DEVICE-CHARSETFN)

previous date: 20-Jul-2022 00:03:06 {WMEDLEY}<sources>CLSTREAMS.;41

Read Table: XCL

Package: INTERLISP

Format: XCCS

```
(RPAQQ CLSTREAMSCOMS
       (
```

;;; Implements a number of stream functions from CommonLisp. See CLtL chapter 21

```
       (COMS ;; documented functions and macros
             (FUNCTIONS OPEN CL:CLOSE CL:STREAM-EXTERNAL-FORMAT)
             (FUNCTIONS CL:STREAM-ELEMENT-TYPE CL:INPUT-STREAM-P CL:OUTPUT-STREAM-P XCL:OPEN-STREAM-P)
             (COMS (FUNCTIONS FILE-STREAM-POSITION)
                   (SETFS FILE-STREAM-POSITION))
             (FUNCTIONS CL:MAKE-SYNONYM-STREAM XCL:SYNONYM-STREAM-P XCL:SYNONYM-STREAM-SYMBOL
                    XCL:FOLLOW-SYNONYM-STREAMS)
             (FUNCTIONS CL:MAKE-BROADCAST-STREAM XCL:BROADCAST-STREAM-P XCL:BROADCAST-STREAM-STREAMS)
             (FUNCTIONS CL:MAKE-CONCATENATED-STREAM XCL:CONCATENATED-STREAM-P XCL:CONCATENATED-STREAM-STREAMS)
             (FUNCTIONS CL:MAKE-TWO-WAY-STREAM XCL:TWO-WAY-STREAM-P XCL:TWO-WAY-STREAM-OUTPUT-STREAM
                    XCL:TWO-WAY-STREAM-INPUT-STREAM)
             (FUNCTIONS CL:MAKE-ECHO-STREAM XCL:ECHO-STREAM-P XCL:ECHO-STREAM-INPUT-STREAM
                    XCL:ECHO-STREAM-OUTPUT-STREAM)
             (FUNCTIONS CL:MAKE-STRING-INPUT-STREAM MAKE-CONCATENATED-STRING-INPUT-STREAM)
             (FUNCTIONS %MAKE-INITIAL-STRING-STREAM-CONTENTS)
             (FUNCTIONS CL:WITH-OPEN-STREAM CL:WITH-INPUT-FROM-STRING CL:WITH-OUTPUT-TO-STRING
                    CL:WITH-OPEN-FILE)
             (FUNCTIONS CL:MAKE-STRING-OUTPUT-STREAM MAKE-FILL-POINTER-OUTPUT-STREAM
                    CL:GET-OUTPUT-STREAM-STRING \\STRING-STREAM-OUTCHARFN \\ADJUSTABLE-STRING-STREAM-OUTCHARFN)
             )
       (COMS ;; helpers
             (FUNCTIONS %NEW-FILE PREDICT-NAME)
             (DECLARE\: EVAL@COMPILE DONTCOPY (FUNCTIONS INTERLISP-ACCESS)))
```

;; methods for the special devices

```
       (COMS                                                    ; broadcast streams
             (FNS %BROADCAST-STREAM-DEVICE-BOUT %BROADCAST-STREAM-DEVICE-CLOSEFILE
                  %BROADCAST-STREAM-DEVICE-FORCEOUTPUT)
             (FUNCTIONS %BROADCAST-STREAM-DEVICE-CHARSETFN)
             (FNS %BROADCAST-STREAM-OUTCHARFN))
       (COMS                                                    ; Concatenated streams
             (FNS %CONCATENATED-STREAM-DEVICE-BIN %CONCATENATED-STREAM-DEVICE-CLOSEFILE
                  %CONCATENATED-STREAM-DEVICE-EOFP %CONCATENATED-STREAM-DEVICE-PEEKBIN
                  %CONCATENATED-STREAM-DEVICE-BACKFILEPTR)
             (FNS %CONCATENATED-STREAM-INCCODEFN %CONCATENATED-STREAM-PEEKCCODEFN
                  %CONCATENATED-STREAM-BACKCCODEFN)
             (FUNCTIONS %CONCATENATED-STREAM-DEVICE-CHARSETFN))
       (FNS %ECHO-STREAM-DEVICE-BIN %ECHO-STREAM-INCCODEFN)
       (COMS                                                    ; Synonym streams
             (FUNCTIONS %SYNONYM-STREAM-DEVICE-GET-INDIRECT-STREAM)
             (FNS %SYNONYM-STREAM-DEVICE-BIN %SYNONYM-STREAM-DEVICE-BOUT %SYNONYM-STREAM-DEVICE-EOFP
                  %SYNONYM-STREAM-DEVICE-FORCEOUTPUT %SYNONYM-STREAM-DEVICE-GETFILEINFO
                  %SYNONYM-STREAM-DEVICE-PEEKBIN %SYNONYM-STREAM-DEVICE-READP
                  %SYNONYM-STREAM-DEVICE-BACKFILEPTR %SYNONYM-STREAM-DEVICE-SETFILEINFO
                  %SYNONYM-STREAM-DEVICE-CHARSETFN %SYNONYM-STREAM-DEVICE-CLOSEFILE)
             ;; helper
             (FNS %SYNONYM-STREAM-DEVICE-GET-STREAM)
             ;; Synonym external format
             (FNS %SYNONYM-STREAM-OUTCHARFN %SYNONYM-STREAM-INCCODEFN %SYNONYM-STREAM-PEEKCCODEFN
                  %SYNONYM-STREAM-BACKCCODEFN))
       (COMS                                                    ; Two-way streams
             (FNS %TWO-WAY-STREAM-BACKCCODEFN %TWO-WAY-STREAM-INCCODEFN %TWO-WAY-STREAM-OUTCHARFN
                  %TWO-WAY-STREAM-PEEKCCODEFN)
             (FNS %TWO-WAY-STREAM-DEVICE-BIN %TWO-WAY-STREAM-DEVICE-INPUTSTREAM %TWO-WAY-STREAM-DEVICE-BOUT
                  %TWO-WAY-STREAM-DEVICE-OUTPUTSTREAM %TWO-WAY-STREAM-DEVICE-OUTCHARFN
                  %TWO-WAY-STREAM-DEVICE-CLOSEFILE %TWO-WAY-STREAM-DEVICE-EOFP %TWO-WAY-STREAM-DEVICE-READP
                  %TWO-WAY-STREAM-DEVICE-BACKFILEPTR %TWO-WAY-STREAM-DEVICE-FORCEOUTPUT
                  %TWO-WAY-STREAM-DEVICE-PEEKBIN %TWO-WAY-STREAM-DEVICE-CHARSETFN))
       (COMS                                                    ; Fill-pointer streams
             (FUNCTIONS %FILL-POINTER-STREAM-DEVICE-CLOSEFILE %FILL-POINTER-STREAM-DEVICE-GETFILEPTR))
       (GLOBALVARS %SYNONYM-STREAM-DEVICE %BROADCAST-STREAM-DEVICE %CONCATENATED-STREAM-DEVICE
              %TWO-WAY-STREAM-DEVICE %ECHO-STREAM-DEVICE \\FILL-POINTER-STREAM-DEVICE)
       (COMS ;; module initialization
```

```
            (VARIABLES *DEBUG-IO* *QUERY-IO* *TERMINAL-IO* *ERROR-OUTPUT* *STANDARD-OUTPUT* *STANDARD-INPUT*)
            (FUNCTIONS %INITIALIZE-STANDARD-STREAMS)
            (FNS %INITIALIZE-CLSTREAM-TYPES)
            (DECLARE\: DONTEVAL@LOAD DOCOPY                    ; initialization
                   (P (%INITIALIZE-CLSTREAM-TYPES)
                      (%INITIALIZE-STANDARD-STREAMS))))
      (PROP FILETYPE CLSTREAMS)))
```

;;; Implements a number of stream functions from CommonLisp. See CLtL chapter 21

;; documented functions and macros

```
(CL:DEFUN OPEN (FILENAME &KEY (DIRECTION :INPUT)
                        (ELEMENT-TYPE 'CL:STRING-CHAR)
                        (IF-EXISTS NIL EXISTS-P)
                        (IF-DOES-NOT-EXIST NIL DOES-NOT-EXIST-P)
                        (EXTERNAL-FORMAT :DEFAULT))
```

;;; Return a stream which reads from or writes to Filename. Defined keywords: :direction (one of :input, :output or :probe :element-type), Type of object
;;; to read or write, default String-Char, :if-exists (one of :error, :new-version, :overwrite, :append or nil), :if-does-not-exist (one of :error, :create or nil).
;;; :external-format (one of :DEFAULT, :EUC, :JIS, :W-MS, :MS or :XCCS). The specification of :external-format is based on the JEIDA proposal. See
;;; the manual for details.

```
   (CL:UNLESS (MEMQ DIRECTION '(:INPUT :OUTPUT :IO :PROBE))
          (CL:ERROR "~S isn't a valid direction for open." DIRECTION))
   (CL:UNLESS (CL:MEMBER ELEMENT-TYPE '(CL:STRING-CHAR CL:SIGNED-BYTE CL:UNSIGNED-BYTE (CL:UNSIGNED-BYTE 8)
                                                (CL:SIGNED-BYTE 8)
                                                CL:CHARACTER :DEFAULT)
                         :TEST
                         'CL:EQUAL)
          (CL:ERROR "~S isn't an implemented element-type for open." ELEMENT-TYPE))
   (LET ((PATHNAME (PATHNAME FILENAME))
         (FOR-INPUT (MEMQ DIRECTION '(:IO :INPUT)))
         (FOR-OUTPUT (MEMQ DIRECTION '(:IO :OUTPUT)))
         (ACCESS (INTERLISP-ACCESS DIRECTION))
         (FILE-TYPE (IF (CL:MEMBER ELEMENT-TYPE '(CL:UNSIGNED-BYTE CL:SIGNED-BYTE (CL:UNSIGNED-BYTE 8)
                                                       (CL:SIGNED-BYTE 8))
                                :TEST
                                'CL:EQUAL)
                        THEN 'BINARY
                        ELSE 'TEXT))
         (STREAM NIL))
```

;;; Do hairy defaulting of :if-exists and :if-does-not-exist keywords.

```
        (CL:UNLESS EXISTS-P
            (SETQ IF-EXISTS (CL:IF (EQ (CL:PATHNAME-VERSION PATHNAME)
                                       :NEWEST)
                                   :NEW-VERSION
                                   :ERROR)))                   ; If the file does not exist, it is OK to have :if-exists :overwrite.
        (CL:UNLESS DOES-NOT-EXIST-P
            (SETQ IF-DOES-NOT-EXIST (COND
                                        ((OR (EQ IF-EXISTS :APPEND)
                                             (EQ DIRECTION :INPUT))
                                         :ERROR)
                                        ((EQ DIRECTION :PROBE)
                                         NIL)
                                        (T :CREATE))))
        (CL:LOOP                                               ; See if the file exists and handle the existential keywords.
         (LET* ((NAME (PREDICT-NAME PATHNAME))
                (CL:NAMESTRING (MKSTRING NAME)))
               (IF NAME
                   THEN                                        ; file exists
                        (IF FOR-OUTPUT
                            THEN ;; open for output/both
                                 (CASE IF-EXISTS
                                     (:ERROR
                                        (CL:CERROR "write it anyway." "File ~A already exists." CL:NAMESTRING)
                                        (SETQ STREAM (OPENSTREAM CL:NAMESTRING ACCESS NIL
                                                                 `((TYPE ,FILE-TYPE)
                                                                   (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                        (RETURN NIL))
                                     ((:NEW-VERSION :SUPERSEDE :RENAME :RENAME-AND-DELETE)
                                        (SETQ STREAM (OPENSTREAM PATHNAME ACCESS 'NEW
                                                                 `((TYPE ,FILE-TYPE)
                                                                   (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                        (RETURN NIL))
                                     (:OVERWRITE
                                        (SETQ STREAM (OPENSTREAM CL:NAMESTRING ACCESS 'OLD
                                                                 `((TYPE ,FILE-TYPE)
                                                                   (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                        (RETURN NIL))
                                     (:APPEND
                                        (IF (EQ DIRECTION :OUTPUT)
                                            THEN                ; if the direction is output it is the same as interlisp append
```

```
                                        (SETQ STREAM (OPENSTREAM CL:NAMESTRING 'APPEND 'OLD
                                                               `((TYPE ,FILE-TYPE)
                                                                 (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                    ELSE                    ; if direction is io it opens the file for both and goes to the end of
                                                            ; the file
                                          (SETQ STREAM (OPENSTREAM CL:NAMESTRING 'BOTH 'OLD
                                                                 `((TYPE ,FILE-TYPE)
                                                                   (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                          (SETFILEPTR STREAM -1))
                                     (RETURN NIL))
                                ((NIL) (CL:RETURN-FROM OPEN NIL))
                                (T (CL:ERROR "~S is not a valid value for :if-exists." IF-EXISTS)))
                          |elseif| FOR-INPUT
                            |then| ;; open for input/both
                                 (SETQ STREAM (OPENSTREAM CL:NAMESTRING ACCESS 'OLD
                                                        `((TYPE ,FILE-TYPE)
                                                          (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                 (RETURN NIL)
                          |else| ;; open for probe
                                 (SETQ STREAM (|create| STREAM
                                                FULLFILENAME _ (FULLNAME CL:NAMESTRING)))
                                 (RETURN NIL))
                    |else| ;; file does not exist
                          (|if| FOR-OUTPUT
                              |then| (CASE IF-DOES-NOT-EXIST
                                         (:ERROR
                                            (CL:CERROR "prompt for a new name." 'XCL:FILE-NOT-FOUND :PATHNAME
                                                  PATHNAME)
                                            (CL:FORMAT *QUERY-IO* "~&New file name: ")
                                            (SETQ PATHNAME (PATHNAME (CL:READ-LINE *QUERY-IO*))))
                                         (:CREATE
                                            (SETQ STREAM (OPENSTREAM PATHNAME ACCESS 'NEW
                                                                   `((TYPE ,FILE-TYPE)
                                                                     (EXTERNALFORMAT ,EXTERNAL-FORMAT))))
                                            (RETURN NIL))
                                         ((NIL) (CL:RETURN-FROM OPEN NIL))
                                         (T (CL:ERROR "~S is not a valid value for :if-does-not-exist."
                                                  IF-DOES-NOT-EXIST)))
                                |elseif| FOR-INPUT
                                  |then| (CASE IF-DOES-NOT-EXIST
                                               (:ERROR
                                                  (CL:CERROR "prompt for a new name." 'XCL:FILE-NOT-FOUND :PATHNAME
                                                        PATHNAME)
                                                  (CL:FORMAT *QUERY-IO* "~&New file name: ")
                                                  (SETQ PATHNAME (PATHNAME (CL:READ-LINE *QUERY-IO*))))
                                               (:CREATE (%NEW-FILE PATHNAME))
                                               ((NIL) (CL:RETURN-FROM OPEN NIL))
                                               (T (CL:ERROR "~S is not a valid value for :if-does-not-exist."
                                                        IF-DOES-NOT-EXIST)))
                                |else|                      ; Open for probe.
                                     (RETURN NIL)))))
        (STREAMPROP STREAM :FILE-STREAM-P T)
        STREAM))
```

(CL:DEFUN **CL:CLOSE** (STREAM &KEY ABORT)

;;; Close a stream. If ABORT, then don't keep the file

```
    (|if| (STREAMP STREAM)
        |then| (|if| (OPENP STREAM)
                   |then| ;; determine 'deletability' of stream's file before closing, as that trashes the info
                        (LET ((ABORTABLE (AND (DIRTYABLE STREAM)
                                              (NOT (APPENDONLY STREAM)))))
                             (CLOSEF STREAM)
                             (|if| (AND ABORT ABORTABLE)
                                 |then|                     ; eventually we will change device CLOSEF methods to take an
                                                            ; ABORT arg.  For now, simulate it.
                                    (DELFILE (CL:NAMESTRING STREAM)))))
      |else| (ERROR "Closing a non-stream" STREAM))
    T)
```

(CL:DEFUN **CL:STREAM-EXTERNAL-FORMAT** (STREAM)
   (\\EXTERNALFORMAT STREAM))

(CL:DEFUN **CL:STREAM-ELEMENT-TYPE** (STREAM)
   'CL:UNSIGNED-BYTE)

(CL:DEFUN **CL:INPUT-STREAM-P** (STREAM)
   (CL:WHEN (NOT (STREAMP STREAM))

```
          (\\ILLEGAL.ARG STREAM))
    ;; we return T instead of the stream because Symbolics does
    (AND (\\IOMODEP STREAM 'INPUT T)
         T))

(CL:DEFUN CL:OUTPUT-STREAM-P (STREAM)
    (CL:WHEN (NOT (STREAMP STREAM))
          (\\ILLEGAL.ARG STREAM))
    ;; we return T instead of the stream because Symbolics does
    (AND (\\IOMODEP STREAM 'OUTPUT T)
         T))

(CL:DEFUN XCL:OPEN-STREAM-P (STREAM)
    ;; is stream an open stream?
    (AND (STREAMP STREAM)
         (OPENED STREAM)))

(CL:DEFUN FILE-STREAM-POSITION (STREAM)
    (GETFILEPTR STREAM))

(CL:DEFSETF FILE-STREAM-POSITION SETFILEPTR)

(CL:DEFUN CL:MAKE-SYNONYM-STREAM (CL:SYMBOL)          ; Edited 6-Jul-2022 11:53 by rmk
                                                      ; Edited 3-Jul-2022 22:03 by rmk
    ;; A CommonLisp function for shadowing a stream.  See CLtL p.  329 or Steele p 500
    (LET ((STREAM (|create| STREAM
                        DEVICE _ %SYNONYM-STREAM-DEVICE
                        ACCESS _ 'BOTH
                        F1 _ CL:SYMBOL
                        LINELENGTH _ (|fetch| (STREAM LINELENGTH) |of| (CL:SYMBOL-VALUE CL:SYMBOL))
                        READONLY-EXTERNALFORMAT _ T)))
          (STREAMPROP STREAM 'XCL:SYNONYM-STREAM-P T)
          ;; save the synonym stream in the OPENFILELST field of %SYNONYM-STREAM-DEVICE
          (|replace| (FDEV OPENFILELST) |of| %SYNONYM-STREAM-DEVICE |with| (CONS STREAM (|fetch| (FDEV OPENFILELST)
                                                                                              |of| %SYNONYM-STREAM-DEVICE)))
          STREAM))

(CL:DEFUN XCL:SYNONYM-STREAM-P (STREAM)
    (STREAMPROP STREAM 'XCL:SYNONYM-STREAM-P))

(CL:DEFUN XCL:SYNONYM-STREAM-SYMBOL (STREAM)
    (AND (XCL:SYNONYM-STREAM-P STREAM)
         (FETCH (STREAM F1) OF STREAM)))

(CL:DEFUN XCL:FOLLOW-SYNONYM-STREAMS (STREAM)

;;; Return the non-synonym stream at the heart of STREAM.

    (CL:IF (XCL:SYNONYM-STREAM-P STREAM)
        (XCL:FOLLOW-SYNONYM-STREAMS (CL:SYMBOL-VALUE (XCL:SYNONYM-STREAM-SYMBOL STREAM)))
        STREAM))

(CL:DEFUN CL:MAKE-BROADCAST-STREAM (&REST STREAMS)    ; Edited 6-Jul-2022 11:53 by rmk
    (FOR STREAM? IN STREAMS DO (\\GETSTREAM STREAM? 'OUTPUT))
    (LET ((STREAM (|create| STREAM
                        DEVICE _ %BROADCAST-STREAM-DEVICE
                        ACCESS _ 'OUTPUT
                        F1 _ STREAMS
                        READONLY-EXTERNALFORMAT _ T)))
          (STREAMPROP STREAM 'XCL:BROADCAST-STREAM-P T)
          STREAM))

(CL:DEFUN XCL:BROADCAST-STREAM-P (STREAM)
    ;; is stream a broadcast stream?
    (STREAMPROP STREAM 'XCL:BROADCAST-STREAM-P))

(CL:DEFUN XCL:BROADCAST-STREAM-STREAMS (STREAM)
    ;; return all of the streams that STREAM broadcasts to
    (AND (XCL:BROADCAST-STREAM-P STREAM)
```

```
        (FETCH (STREAM F1) OF STREAM)))

(CL:DEFUN CL:MAKE-CONCATENATED-STREAM (&REST STREAMS) ; Edited 6-Jul-2022 11:54 by rmk
   ;; CommonLisp function that creates a  concatenated stream.  See CLtL p.  329
   (FOR STREAM? IN STREAMS DO (\\GETSTREAM STREAM? 'INPUT))
   (LET ((STREAM (|create| STREAM
                       DEVICE _ %CONCATENATED-STREAM-DEVICE
                       ACCESS _ 'INPUT
                       F1 _ STREAMS
                       READONLY-EXTERNALFORMAT _ T)))
        (STREAMPROP STREAM 'XCL:CONCATENATED-STREAM-P T)
        STREAM))

(CL:DEFUN XCL:CONCATENATED-STREAM-P (STREAM)
   (STREAMPROP STREAM 'XCL:CONCATENATED-STREAM-P))

(CL:DEFUN XCL:CONCATENATED-STREAM-STREAMS (STREAM)
   ;; return all of STREAM's concatenated streams
   (AND (XCL:CONCATENATED-STREAM-P STREAM)
        (FETCH (STREAM F1) OF STREAM)))

(CL:DEFUN CL:MAKE-TWO-WAY-STREAM (CL::INPUT-STREAM CL::OUTPUT-STREAM)
                                                          ; Edited 6-Jul-2022 11:55 by rmk
                                                          ; Edited 4-Jul-2022 00:05 by rmk
   ;; A CommonLisp function for splicing together two streams.  See CLtL p.  329
   (CL:SETQ CL::INPUT-STREAM (\\GETSTREAM CL::INPUT-STREAM 'INPUT))
   (CL:SETQ CL::OUTPUT-STREAM (\\GETSTREAM CL::OUTPUT-STREAM 'OUTPUT))
   (LET ((STREAM (|create| STREAM
                       DEVICE _ %TWO-WAY-STREAM-DEVICE
                       ACCESS _ 'BOTH
                       F1 _ CL::INPUT-STREAM
                       F2 _ CL::OUTPUT-STREAM
                       LINELENGTH _ (|fetch| (STREAM LINELENGTH) |of| CL::OUTPUT-STREAM)
                       READONLY-EXTERNALFORMAT _ T)))
        (STREAMPROP STREAM 'XCL:TWO-WAY-STREAM-P T)
        ;; save STREAM  in the OPENFILELST field of %TWO-WAY-STREAM-DEVICE
        (|replace| (FDEV OPENFILELST) |of| %TWO-WAY-STREAM-DEVICE |with| (CONS STREAM (|fetch| (FDEV OPENFILELST)
                                                                                          |of| %TWO-WAY-STREAM-DEVICE)))
        STREAM))

(CL:DEFUN XCL:TWO-WAY-STREAM-P (STREAM)
   ;; is STREAM a two-way stream?
   (STREAMPROP STREAM 'XCL:TWO-WAY-STREAM-P))

(CL:DEFUN XCL:TWO-WAY-STREAM-OUTPUT-STREAM (STREAM)
   (AND (XCL:TWO-WAY-STREAM-P STREAM)
        (FETCH (STREAM F2) OF STREAM)))

(CL:DEFUN XCL:TWO-WAY-STREAM-INPUT-STREAM (STREAM)
   (AND (XCL:TWO-WAY-STREAM-P STREAM)
        (FETCH (STREAM F1) OF STREAM)))

(CL:DEFUN CL:MAKE-ECHO-STREAM (CL::INPUT-STREAM CL::OUTPUT-STREAM)
                                                          ; Edited 6-Jul-2022 11:54 by rmk
   ;; See Steele p 500
   (CL:SETQ CL::INPUT-STREAM (\\GETSTREAM CL::INPUT-STREAM 'INPUT))
   (CL:SETQ CL::OUTPUT-STREAM (\\GETSTREAM CL::OUTPUT-STREAM 'OUTPUT))
   (LET ((STREAM (|create| STREAM
                       DEVICE _ %ECHO-STREAM-DEVICE
                       ACCESS _ 'BOTH
                       F1 _ CL::INPUT-STREAM
                       F2 _ CL::OUTPUT-STREAM
                       LINELENGTH _ (|fetch| (STREAM LINELENGTH) |of| CL::OUTPUT-STREAM)
                       READONLY-EXTERNALFORMAT _ T)))
        (STREAMPROP STREAM 'XCL:ECHO-STREAM-P T)
        ;; save STREAM  in the OPENFILELST field of %ECHO-STREAM-DEVICE
        (|replace| (FDEV OPENFILELST) |of| %ECHO-STREAM-DEVICE |with| (CONS STREAM (|fetch| (FDEV OPENFILELST)
                                                                                        |of| %ECHO-STREAM-DEVICE)))
        STREAM))

(CL:DEFUN XCL:ECHO-STREAM-P (STREAM)
```

```
   ;; is stream an echo stream?
   (STREAMPROP STREAM 'XCL:ECHO-STREAM-P))


(CL:DEFUN XCL:ECHO-STREAM-INPUT-STREAM (STREAM)
   (AND (XCL:ECHO-STREAM-P STREAM)
        (FETCH (STREAM F1) OF STREAM)))


(CL:DEFUN XCL:ECHO-STREAM-OUTPUT-STREAM (STREAM)
   (AND (XCL:ECHO-STREAM-P STREAM)
        (FETCH (STREAM F2) OF STREAM)))


(CL:DEFUN CL:MAKE-STRING-INPUT-STREAM (STRING &OPTIONAL (CL::START 0)
                                                        (CL::END NIL))

;;; A CommonLisp function for producing a stream from a string.  See CLtL p.  330

   (OPENSTRINGSTREAM (|if| (OR (NOT (CL:ZEROP CL::START))
                               (NOT (NULL CL::END)))
                          |then| ;; A displaced array is ok here because the stream's uses GETBASEBYTE directly and doesn't go through the
                                 ;; array code at all.
                                 (SUBSTRING STRING (CL:1+ CL::START)
                                            CL::END)
                          |else| STRING)
         'INPUT))


(CL:DEFUN MAKE-CONCATENATED-STRING-INPUT-STREAM (STRINGS)
   (COND
      ((NULL STRINGS)
       NIL)
      ((NULL (CL:REST STRINGS))
       (CL:MAKE-STRING-INPUT-STREAM (CL:FIRST STRINGS)))
      (T (CL:APPLY 'CL:MAKE-CONCATENATED-STREAM (FOR STRING IN STRINGS COLLECT (CL:MAKE-STRING-INPUT-STREAM
                                                                                STRING))))))


(CL:DEFUN %MAKE-INITIAL-STRING-STREAM-CONTENTS ()
   (CL:MAKE-ARRAY '(256)
          :ELEMENT-TYPE
          'CL:STRING-CHAR :EXTENDABLE T :FILL-POINTER 0))


(DEFMACRO CL:WITH-OPEN-STREAM ((VAR STREAM)
                               &BODY
                               (BODY DECLS))
   (LET ((ABORTP (GENSYM)))
        `(LET ((,VAR ,STREAM)
               (,ABORTP T))
             ,@DECLS
             (CL:UNWIND-PROTECT
                 (CL:MULTIPLE-VALUE-PROG1 (PROGN ,@BODY)
                        (SETQ ,ABORTP NIL))
                 (CL:CLOSE ,VAR :ABORT ,ABORTP)))))


(DEFMACRO CL:WITH-INPUT-FROM-STRING ((CL::VAR STRING &KEY (CL::INDEX NIL CL::INDEXP)
                                                          (CL::START 0 CL::STARTP)
                                                          (CL::END NIL CL:ENDP))
                                     &BODY
                                     (CL::BODY CL::DECLS))
   `(LET* ((CL::$STRING$ ,STRING)
           (CL::$START$ ,CL::START))
          (DECLARE (LOCALVARS CL::$STRING$ CL::$START$))
          (CL:WITH-OPEN-STREAM (,CL::VAR (CL:MAKE-STRING-INPUT-STREAM CL::$STRING$ CL::$START$ ,CL::END))
                 ,@CL::DECLS
                 ,@(CL:IF CL::INDEXP
                          ;; This exists as a fudge for the fat string problem.  It WILL GO AWAY when STRINGSTREAMS HAVE THEIR OWN
                          ;; DEVICE.
                          `((CL:MULTIPLE-VALUE-PROG1 (PROGN ,@CL::BODY)
                                   ;; (IF (FASL::FAT-STRING-P $STRING$) (SETF ,INDEX (+ $START$ (IL:IQUOTIENT (IL:GETFILEPTR
                                   ;; ,VAR) 2))) (SETF ,INDEX (+ $START$ (IL:GETFILEPTR ,VAR))))
                                   (CL:SETF ,CL::INDEX (+ CL::$START$ (GETFILEPTR ,CL::VAR)))))
                          CL::BODY))))


(DEFMACRO CL:WITH-OUTPUT-TO-STRING ((VAR &OPTIONAL (STRING NIL ST-P))
                                    &BODY
                                    (FORMS DECLS))
   (COND
      (ST-P `(CL:WITH-OPEN-STREAM (,VAR (MAKE-FILL-POINTER-OUTPUT-STREAM ,STRING))
```

```
                    ,@DECLS
                    ,@FORMS))
      (T `(CL:WITH-OPEN-STREAM (,VAR (CL:MAKE-STRING-OUTPUT-STREAM))
                ,@DECLS
                (PROGN ,@FORMS (CL:GET-OUTPUT-STREAM-STRING ,VAR))))))

(DEFMACRO CL:WITH-OPEN-FILE ((VAR &REST OPEN-ARGS)
                             &BODY
                             (FORMS DECLS))
```

;;; The file whose name is File-Name is opened using the OPEN-ARGS and bound to the variable VAR. The Forms are executed, and when they ;;; terminate, normally or otherwise, the file is closed.

```
   (LET ((ABORTP (GENSYM)))
        `(LET ((,VAR (OPEN ,@OPEN-ARGS))
               (,ABORTP T))
              ,@DECLS
              (CL:UNWIND-PROTECT
                  (CL:MULTIPLE-VALUE-PROG1 (PROGN ,@FORMS)
                        (SETQ ,ABORTP NIL))
                  (CL:CLOSE ,VAR :ABORT ,ABORTP)))))

(DEFINLINE CL:MAKE-STRING-OUTPUT-STREAM ()
```

;;; A function for producing a string stream. See also the function get-output-stream-string. Also, see CLtL p. 330

```
   (MAKE-FILL-POINTER-OUTPUT-STREAM))

(CL:DEFUN MAKE-FILL-POINTER-OUTPUT-STREAM (&OPTIONAL (STRING (%MAKE-INITIAL-STRING-STREAM-CONTENTS)))
   (DECLARE (GLOBALVARS \\FILL-POINTER-STREAM-DEVICE))
   (|if| (NOT (CL:ARRAY-HAS-FILL-POINTER-P STRING))
       |then| (\\ILLEGAL.ARG STRING)
     |else| (LET ((STREAM (|create| STREAM
                                    DEVICE _ \\FILL-POINTER-STREAM-DEVICE
                                    F1 _ STRING
                                    ACCESS _ 'OUTPUT
                                    OTHERPROPS _ '(STRING-OUTPUT-STREAM T))))
                                                          ; give it a canned property list to save some consing.
                 (|replace| (STREAM OUTCHARFN) |of| STREAM |with| (|if| (EXTENDABLE-ARRAY-P STRING)
                                                                        |then| (FUNCTION
                                                                                  \\ADJUSTABLE-STRING-STREAM-OUTCHARFN)
                                                                      |else| (FUNCTION \\STRING-STREAM-OUTCHARFN)))
                 (|replace| (STREAM STRMBOUTFN) |of| STREAM |with| (FUNCTION \\OUTCHAR))
                 STREAM)))

(CL:DEFUN CL:GET-OUTPUT-STREAM-STRING (STRING-OUTPUT-STREAM)
```

;;; A CommonLisp function for getting the contents of the buffer created by a call to make-string-output-stream. See CLtL p. 330

```
   (|if| (NOT (STREAMPROP STRING-OUTPUT-STREAM 'STRING-OUTPUT-STREAM))
       |then| (ERROR "Stream not a string-output-stream" STRING-OUTPUT-STREAM)
     |else| (PROG1 (|fetch| (STREAM F1) |of| STRING-OUTPUT-STREAM)
                   (|replace| (STREAM F1) |of| STRING-OUTPUT-STREAM |with| (%MAKE-INITIAL-STRING-STREAM-CONTENTS)))))


(CL:DEFUN \\STRING-STREAM-OUTCHARFN (STREAM CHAR)
   (IF (OR (IEQP (FETCH (STREAM CHARPOSITION) OF STREAM)
                 (FETCH (STREAM LINELENGTH) OF STREAM))
           (EQ CHAR (CHARCODE EOL)))
       THEN (REPLACE (STREAM CHARPOSITION) OF STREAM WITH 0)
     ELSE (ADD (FETCH (STREAM CHARPOSITION) OF STREAM)
               1))
   (CL:VECTOR-PUSH (CL:CHARACTER CHAR)
          (FETCH (STREAM F1) OF STREAM)))


(CL:DEFUN \\ADJUSTABLE-STRING-STREAM-OUTCHARFN (STREAM CHAR)
   (LET ((STRING (FETCH (STREAM F1) OF STREAM))
         (CH (CL:CHARACTER CHAR)))
        (IF (OR (IEQP (FETCH (STREAM CHARPOSITION) OF STREAM)
                      (FETCH (STREAM LINELENGTH) OF STREAM))
                (EQ CHAR (CHARCODE EOL)))
            THEN (REPLACE (STREAM CHARPOSITION) OF STREAM WITH 0)
          ELSE (ADD (FETCH (STREAM CHARPOSITION) OF STREAM)
                    1))
        ;; Do the equivalent of VECTOR-PUSH-EXTEND inline to save the significant! overhead of calculating the new length at each character.
        (CL:UNLESS (CL:VECTOR-PUSH CH STRING)
            (LET ((CURRENT-LENGTH (CL:ARRAY-TOTAL-SIZE STRING)))
                 (IF (>= CURRENT-LENGTH (CL:1- CL:ARRAY-TOTAL-SIZE-LIMIT))
                     THEN (PROCEED-CASE (CL:ERROR 'END-OF-FILE :STREAM STREAM)
                                 (SI::RETRY-OUTCHAR NIL :REPORT "VECTOR-PUSH the character anyway" :CONDITION
                                        END-OF-FILE (CL:VECTOR-PUSH CH (FETCH (STREAM F1) OF STREAM))))
```

```
                    ELSE (CL:ADJUST-ARRAY STRING (MIN (CL:1- CL:ARRAY-TOTAL-SIZE-LIMIT)
                                                      (+ CURRENT-LENGTH (MAX (LRSH CURRENT-LENGTH 1)
                                                                             *DEFAULT-PUSH-EXTENSION-SIZE*))))
                         (CL:VECTOR-PUSH CH STRING))))))
```

;; helpers

```
(CL:DEFUN %NEW-FILE (FILENAME)
   (CLOSEF (OPENSTREAM FILENAME 'OUTPUT 'NEW)))

(CL:DEFUN PREDICT-NAME (PATHNAME)
   (LET ((PATH (CL:PROBE-FILE PATHNAME)))
        (IF PATH
            THEN (CL:NAMESTRING PATH))))

(DECLARE\: EVAL@COMPILE DONTCOPY

(DEFMACRO INTERLISP-ACCESS (DIRECTION)
   `(CASE ,DIRECTION
        (:INPUT 'INPUT)
        (:OUTPUT 'OUTPUT)
        (:IO 'BOTH)
        (T NIL)))
)
```

;; methods for the special devices

;; broadcast streams

```
(DEFINEQ

(%BROADCAST-STREAM-DEVICE-BOUT
  (LAMBDA (STREAM BYTE)                                    ; Edited 13-Jan-87 14:45 by hdj

    ;; The BOUT method for the broadcast-stream device

    (|for| S |in| (|fetch| F1 |of| STREAM) |do| (\\BOUT S BYTE))
    BYTE))

(%BROADCAST-STREAM-DEVICE-CLOSEFILE
  (LAMBDA (STREAM)                                         (* |hdj| "26-Mar-86 16:28")

;;; The CLOSEFILE method for the broadcast-stream device

    (|replace| ACCESS |of| STREAM |with| NIL)
    (|replace| F1 |of| STREAM |with| NIL)
    STREAM))

(%BROADCAST-STREAM-DEVICE-FORCEOUTPUT
  (LAMBDA (|stream| |waitForFinish?|)                      (* |smL| "14-Aug-85 15:55")

;;; The FORCEOUTPUT method for the broadcast-stream device

    (|for| \s |in| (|fetch| F1 |of| |stream|) |do| (FORCEOUTPUT \s |waitForFinish?|))))

)

(CL:DEFUN %BROADCAST-STREAM-DEVICE-CHARSETFN (STREAM NEWVALUE DONTMARKFILE)
                                                           ; Edited  8-Dec-2023 15:43 by rmk

   ;; charset function for broadcast streams.  Not clear what the value should be, so we arbitrarily return the value of the last stream.

   (FOR S IN (FETCH (STREAM F1) OF STREAM) DO (SETQ $$VAL (ACCESS-CHARSET S NEWVALUE DONTMARKFILE))))

(DEFINEQ

(%BROADCAST-STREAM-OUTCHARFN
  (LAMBDA (STREAM CHARCODE)                                ; Edited  5-Jul-2022 12:50 by rmk
                                                           ; Edited 18-Mar-87 11:00 by lal

    ;; outcharfn for broadcast streams

    ;; Using the charposition from the first stream in the broadcast stream list

    (LET ((STREAMS (|fetch| (STREAM F1) |of| STREAM)))
         (CL:WHEN STREAMS
              (|for| S |in| STREAMS |do| (\\OUTCHAR S CHARCODE))
              (|replace| (STREAM CHARPOSITION) |of| STREAM |with| (|fetch| (STREAM CHARPOSITION) |of| (CAR STREAMS)))))
    CHARCODE))

)
```

;; Concatenated streams

```
(DEFINEQ

(%CONCATENATED-STREAM-DEVICE-BIN
  (LAMBDA (STREAM)                                          ; Edited 13-Jan-87 14:52 by hdj

    ;; The BIN method for the concatenated-stream device

    (WHILE (FETCH (STREAM F1) OF STREAM) DO (IF (EOFP (CAR (FETCH (STREAM F1) OF STREAM)))
                                                THEN (CLOSEF (POP (FETCH (STREAM F1) OF STREAM)))
                                              ELSE (RETURN (\\BIN (CAR (FETCH (STREAM F1) OF STREAM)))))
      FINALLY                                               ; the EOF case
             (\\EOF.ACTION STREAM))))

(%CONCATENATED-STREAM-DEVICE-CLOSEFILE
  (LAMBDA (|stream|)                                        (* |smL| "14-Aug-85 16:53")

;;; The CLOSEFILE method for the concatenated-stream device

    (|replace| ACCESS |of| |stream| |with| NIL)
    (|for| \s |in| (|fetch| F1 |of| |stream|) |do| (CLOSEF \s))
    (|replace| F1 |of| |stream| |with| NIL)
    |stream|))

(%CONCATENATED-STREAM-DEVICE-EOFP
  (LAMBDA (|stream|)                                        ; Edited 17-Mar-87 09:20 by lal

;;; The EOFP method for the concatenated-stream device

    (|while| (|fetch| F1 |of| |stream|) |do| (|if| (EOFP (CAR (|fetch| F1 |of| |stream|)))
                                                  |then| (CLOSEF (|pop| (|fetch| F1 |of| |stream|)))
                                                |else| (RETURN NIL))
      |finally|                                             ; the EOF case
             (RETURN T))))

(%CONCATENATED-STREAM-DEVICE-PEEKBIN
  (LAMBDA (|stream| |noErrorFlg?|)                          (* |smL| "14-Aug-85 16:53")

;;; The PEEKBIN method for the concatenated-stream device

    (|while| (|fetch| F1 |of| |stream|) |do| (|if| (EOFP (CAR (|fetch| F1 |of| |stream|)))
                                                  |then| (CLOSEF (|pop| (|fetch| F1 |of| |stream|)))
                                                |else| (RETURN (\\PEEKBIN (CAR (|fetch| F1 |of| |stream|))))))
      |finally|                                             ; the EOF case
             (|if| |noErrorFlg?|
                  |then| (RETURN NIL)
                |else| (\\EOF.ACTION |stream|)))))

(%CONCATENATED-STREAM-DEVICE-BACKFILEPTR
  (LAMBDA (|stream|)                                        ; Edited 24-Mar-87 10:47 by lal

    ;; concatenated streams are read sequentially and a list of them are kept in F1.  as they are read, the used stream is removed from the list.
    ;; \backfileptr will work because 1) when a file is stream is used up the new one is read, at least one character's worth and 2) \backfileptr only
    ;; needs to back up one character

    (\\BACKFILEPTR (CAR (|fetch| F1 |of| |stream|)))))

)

(DEFINEQ

(%CONCATENATED-STREAM-INCCODEFN
  (LAMBDA (STREAM)                                          ; Edited  5-Jul-2022 16:16 by rmk
                                                            ; Edited 13-Jan-87 14:52 by hdj

    ;; The INCCODE method for the concatenated-stream device

    (WHILE (FETCH (STREAM F1) OF STREAM) DO (IF (EOFP (CAR (FETCH (STREAM F1) OF STREAM)))
                                                THEN (CLOSEF (POP (FETCH (STREAM F1) OF STREAM)))
                                              ELSE (RETURN (\\INCCODE (CAR (FETCH (STREAM F1) OF STREAM))
                                                                  BYTECOUNTVAR BYTECOUNTVAL)))
      FINALLY                                               ; the EOF case
             (\\EOF.ACTION STREAM))))

(%CONCATENATED-STREAM-PEEKCCODEFN
  (LAMBDA (STREAM)                                          ; Edited  5-Jul-2022 16:16 by rmk
                                                            ; Edited 13-Jan-87 14:52 by hdj

    ;; The INCCODE method for the concatenated-stream device

    (WHILE (FETCH (STREAM F1) OF STREAM) DO (IF (EOFP (CAR (FETCH (STREAM F1) OF STREAM)))
                                                THEN (CLOSEF (POP (FETCH (STREAM F1) OF STREAM)))
                                              ELSE (RETURN (\\INCCODE (CAR (FETCH (STREAM F1) OF STREAM))
                                                                  BYTECOUNTVAR BYTECOUNTVAL)))
      FINALLY                                               ; the EOF case
             (\\EOF.ACTION STREAM))))
```

```
(%CONCATENATED-STREAM-BACKCCODEFN
  (LAMBDA (STREAM)                                          ; Edited  5-Jul-2022 16:16 by rmk
                                                            ; Edited 13-Jan-87 14:52 by hdj

    ;; The INCCODE method for the concatenated-stream device

    (WHILE (FETCH (STREAM F1) OF STREAM) DO (IF (EOFP (CAR (FETCH (STREAM F1) OF STREAM)))
                                                THEN (CLOSEF (POP (FETCH (STREAM F1) OF STREAM)))
                                              ELSE (RETURN (\\INCCODE (CAR (FETCH (STREAM F1) OF STREAM))
                                                                      BYTECOUNTVAR BYTECOUNTVAL)))
      FINALLY                                               ; the EOF case
             (\\EOF.ACTION STREAM))))

)


(CL:DEFUN %CONCATENATED-STREAM-DEVICE-CHARSETFN (STREAM NEWVALUE DONTMARKFILE)
                                                            ; Edited  8-Dec-2023 15:46 by rmk
    ;; the charset method for concatenated stream devices

    (LET ((STREAMS (FETCH (STREAM F1) OF STREAM)))
         (IF STREAMS
             THEN (ACCESS-CHARSET (CAR STREAMS)
                                  NEWVALUE DONTMARKFILE)
           ELSE 0)))

(DEFINEQ

(%ECHO-STREAM-DEVICE-BIN
  (LAMBDA (STREAM)                                          (* |hdj| "21-Apr-86 18:33")

;;; The BIN method for the echo-stream device

    (LET ((BYTE (%TWO-WAY-STREAM-DEVICE-BIN STREAM)))
         (\\BOUT STREAM BYTE)
         BYTE)))

(%ECHO-STREAM-INCCODEFN
  (LAMBDA (STREAM BYTECOUNTVAR BYTECOUNTVAL)                ; Edited  5-Jul-2022 23:07 by rmk

;;; The INCCODE method for the echo-stream device

    (%TWO-WAY-STREAM-OUTCHARFN STREAM (%TWO-WAY-STREAM-INCCODEFN STREAM BYTECOUNTVAR BYTECOUNTVAL))))

)

;; Synonym streams


(CL:DEFUN %SYNONYM-STREAM-DEVICE-GET-INDIRECT-STREAM (SYNONYM-STREAM)
    ;; given a synonym-stream, find out what it is currently tracking
    (CL:SYMBOL-VALUE (XCL:SYNONYM-STREAM-SYMBOL SYNONYM-STREAM)))

(DEFINEQ

(%SYNONYM-STREAM-DEVICE-BIN
  (LAMBDA (STREAM)                                          (* |hdj| "19-Mar-86 17:19")

;;; The BIN method for the synonym-stream device.

    (\\BIN (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM))))

(%SYNONYM-STREAM-DEVICE-BOUT
  (LAMBDA (STREAM BYTE)                                     (* |hdj| "19-Mar-86 17:20")

;;; The BOUT method for the synonym-stream device.

    (\\BOUT (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           BYTE)))

(%SYNONYM-STREAM-DEVICE-EOFP
  (LAMBDA (STREAM)                                          (* |hdj| "19-Mar-86 17:20")

;;; The EOFP method for the synonym-stream device.

    (\\EOFP (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM))))

(%SYNONYM-STREAM-DEVICE-FORCEOUTPUT
  (LAMBDA (STREAM WAITFORFINISH)                            (* |hdj| "19-Mar-86 17:09")

;;; The FORCEOUTPUT method for the synonym-stream device.
```

```
    (FORCEOUTPUT (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           WAITFORFINISH)))
```

(%SYNONYM-STREAM-DEVICE-GETFILEINFO

```
  (LAMBDA (STREAM ATTRIBUTE DEVICE)                    (* |hdj| "19-Mar-86 17:10")
```

;;; The GETFILEINFO method for the synonym-stream device.

```
    (GETFILEINFO (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           ATTRIBUTE)))
```

(%SYNONYM-STREAM-DEVICE-PEEKBIN

```
  (LAMBDA (STREAM NOERRORFLG?)                         (* |hdj| "19-Mar-86 17:12")
```

;;; The PEEKBIN method for the synonym-stream device

```
    (\\PEEKBIN (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           NOERRORFLG?)))
```

(%SYNONYM-STREAM-DEVICE-READP

```
  (LAMBDA (STREAM FLG)
    (READP (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           FLG)))
```

(%SYNONYM-STREAM-DEVICE-BACKFILEPTR

```
  (LAMBDA (STREAM)                                     (* |hdj| "26-Aug-86 17:35")
    (\\BACKFILEPTR (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM))))
```

(%SYNONYM-STREAM-DEVICE-SETFILEINFO

```
  (LAMBDA (STREAM ATTRIBUTE VALUE DEVICE)              (* |hdj| "19-Mar-86 17:17")
```

;;; The SETFILEINFO method for the synonym-stream device.

```
    (SETFILEINFO (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           ATTRIBUTE VALUE)))
```

(%SYNONYM-STREAM-DEVICE-CHARSETFN

```
  (LAMBDA (STREAM NEWVALUE DONTMARKFILE)               ; Edited  8-Dec-2023 15:40 by rmk
                                                       ; Edited 11-Sep-87 16:01 by bvm:
    ;; The charset method for the synonym-stream device.
    (ACCESS-CHARSET (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)
           NEWVALUE DONTMARKFILE)))
```

(%SYNONYM-STREAM-DEVICE-CLOSEFILE

```
  (LAMBDA (STREAM)                                     ; Edited 18-Dec-87 12:17 by sye
```

;;; the CLOSEFILE method for the synonym-stream device

```
    (|replace| F1 |of| STREAM |with| NIL)
    ;; remove the synonym stream STREAM from the OPENFILELST field of %SYNONYM-STREAM-DEVICE
    (|replace| (FDEV OPENFILELST) |of| %SYNONYM-STREAM-DEVICE |with| (DREMOVE STREAM (|fetch| (FDEV OPENFILELST)
                                                                                       |of| %SYNONYM-STREAM-DEVICE)))
    STREAM))

)

;; helper

(DEFINEQ

(%SYNONYM-STREAM-DEVICE-GET-STREAM
  (LAMBDA (|stream|)                                   ; Edited 12-Jan-87 14:46 by hdj
    ;; given a synonym-stream, find out what it is currently tracking
    (CL:SYMBOL-VALUE (|fetch| (STREAM F1) |of| |stream|))))

)

;; Synonym external format

(DEFINEQ

(%SYNONYM-STREAM-OUTCHARFN
  (LAMBDA (STREAM CHARCODE)                            ; Edited  5-Jul-2022 23:12 by rmk
                                                       ; Edited  3-Jul-2022 21:16 by rmk
                                                       ; Edited  3-Jan-90 15:25 by jds
    ;;  OUTCHARFN for synonym streams
```

```
    (LET ((OTHER-STREAM (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)))
         (|freplace| (STREAM EOLCONVENTION) |of| STREAM |with| (|ffetch| (STREAM EOLCONVENTION) |of| OTHER-STREAM))
         (\\OUTCHAR OTHER-STREAM CHARCODE)
         (|freplace| (STREAM CHARPOSITION) |of| STREAM |with| (|ffetch| (STREAM CHARPOSITION) |of| OTHER-STREAM))
         CHARCODE)))
```

(%SYNONYM-STREAM-INCCODEFN

```
  (LAMBDA (STREAM BYTECOUNTVAR BYTECOUNTVAL)                    ; Edited  3-Jul-2022 21:28 by rmk
    ;; INCCODEFN for synonym streams
    (LET ((OTHER-STREAM (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)))
         (|freplace| (STREAM EOLCONVENTION) |of| STREAM |with| (|ffetch| (STREAM EOLCONVENTION) |of| OTHER-STREAM))
         (\\INCCODE OTHER-STREAM BYTECOUNTVAR BYTECOUNTVAL))))
```

(%SYNONYM-STREAM-PEEKCCODEFN

```
  (LAMBDA (STREAM NOERROR)                                      ; Edited 19-Jul-2022 22:58 by rmk
                                                                ; Edited  3-Jul-2022 21:31 by rmk
                                                                ; Edited  3-Jan-90 15:25 by jds
    ;; PEEKCCODEFN for synonym streams
    (LET ((OTHER-STREAM (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)))
         (|freplace| (STREAM EOLCONVENTION) |of| STREAM |with| (|ffetch| (STREAM EOLCONVENTION) |of| OTHER-STREAM))
         (CL:FUNCALL (|ffetch| (STREAM PEEKCCODEFN) |of| OTHER-STREAM)
                OTHER-STREAM NOERROR))))
```

(%SYNONYM-STREAM-BACKCCODEFN

```
  (LAMBDA (STREAM BYTECOUNTVAR BYTECOUNTVAL)                    ; Edited  3-Jul-2022 21:31 by rmk
                                                                ; Edited  3-Jan-90 15:25 by jds
    ;; BACKCCODEFN for synonym streams
    (LET ((OTHER-STREAM (%SYNONYM-STREAM-DEVICE-GET-STREAM STREAM)))
         (|freplace| (STREAM EOLCONVENTION) |of| STREAM |with| (|ffetch| (STREAM EOLCONVENTION) |of| OTHER-STREAM))
         (\\BACKCCODE OTHER-STREAM BYTECOUNTVAR BYTECOUNTVAL))))

)

;; Two-way streams

(DEFINEQ
```

(%TWO-WAY-STREAM-BACKCCODEFN

```
  (LAMBDA (STREAM BYTECOUNTVAR BYTECOUNTVAL)                    ; Edited  3-Jul-2022 23:52 by rmk
                                                                ; Edited  3-Jan-90 15:26 by jds
    ;; backccodefn for two-way streams
    (\\BACKCCODE (|fetch| (STREAM F1) |of| STREAM)
           BYTECOUNTVAR BYTECOUNTVAL)))
```

(%TWO-WAY-STREAM-INCCODEFN

```
  (LAMBDA (STREAM BYTECOUNTVAR BYTECOUNTVAL)                    ; Edited  3-Jul-2022 23:52 by rmk
                                                                ; Edited  3-Jan-90 15:26 by jds
    ;; inccodefn for two-way streams
    (\\INCCODE (|fetch| (STREAM F1) |of| STREAM)
           BYTECOUNTVAR BYTECOUNTVAL)))
```

(%TWO-WAY-STREAM-OUTCHARFN

```
  (LAMBDA (STREAM CHARCODE)                                     ; Edited  5-Jul-2022 23:06 by rmk
                                                                ; Edited  3-Jan-90 15:26 by jds
    ;; outcharfn for two-way streams
    (PROG1 (\\OUTCHAR (|fetch| (STREAM F2) |of| STREAM)
                  CHARCODE)
        (|freplace| (STREAM CHARPOSITION) |of| STREAM |with| (|ffetch| (STREAM CHARPOSITION) |of| (|ffetch| (STREAM F2)
                                                                                                  |of| STREAM))))))
```

(%TWO-WAY-STREAM-PEEKCCODEFN

```
  (LAMBDA (STREAM NOERROR)
    ;; Edited 20-Jul-2022 00:02 by rmk: No EOL argument at this level, make direct FUNCALL.
    ;; Edited  4-Jul-2022 00:02 by rmk
    ;; Edited  3-Jan-90 15:26 by jds
    ;; peekccodefn for two-way streams
    (CL:FUNCALL (|ffetch| (STREAM PEEKCCODEFN) |of| (|fetch| (STREAM F1) |of| STREAM))
           (|fetch| (STREAM F1) |of| STREAM)
           NOERROR)))

)
```

```
(DEFINEQ
```

(%TWO-WAY-STREAM-DEVICE-BIN

```
  (LAMBDA (|stream|)                                    (* |smL| "14-Aug-85 16:44")
```

;;; The BIN method for the two-way-stream device

```
    (\\BIN (|fetch| F1 |of| |stream|))))
```

(%TWO-WAY-STREAM-DEVICE-INPUTSTREAM

```
  (LAMBDA (|stream|)                                    ; Edited 14-Apr-87 16:59 by bvm:
```

;;; Fetch the real input for the two-way-stream device

```
    (|fetch| F1 |of| |stream|)))
```

(%TWO-WAY-STREAM-DEVICE-BOUT

```
  (LAMBDA (STREAM BYTE)                                 (* |hdj| "17-Sep-86 15:28")
    ;;  the BOUT method for two-way streams
    (\\BOUT (|fetch| F2 |of| STREAM)
           BYTE)))
```

(%TWO-WAY-STREAM-DEVICE-OUTPUTSTREAM

```
  (LAMBDA (STREAM BYTE)                                 ; Edited 14-Apr-87 16:59 by bvm:
    ;; Fetch the real output stream for two-way streams
    (|fetch| F2 |of| STREAM)))
```

(%TWO-WAY-STREAM-DEVICE-OUTCHARFN

```
  (LAMBDA (STREAM CHARCODE)                             ; Edited  3-Jan-90 15:26 by jds
    ;; outcharfn for two-way streams
    (\\OUTCHAR (|fetch| (STREAM F2) |of| STREAM)
           CHARCODE)
    (|freplace| (STREAM CHARPOSITION) |of| STREAM |with| (|ffetch| (STREAM CHARPOSITION) |of| (|ffetch| (STREAM F2)
                                                                                                |of| STREAM)))))
```

(%TWO-WAY-STREAM-DEVICE-CLOSEFILE

```
  (LAMBDA (|stream|)                                    ; Edited 18-Dec-87 12:32 by sye
```

;;; The CLOSEFILE method for the two-way-stream device and echo-stream device

```
    (LET ((STREAMDEVICE (|if| (XCL:TWO-WAY-STREAM-P |stream|)
                              |then| %TWO-WAY-STREAM-DEVICE
                            |else| %ECHO-STREAM-DEVICE)))
         (|replace| ACCESS |of| |stream| |with| NIL)
         (CLOSEF? (|fetch| F1 |of| |stream|))
         (|replace| F1 |of| |stream| |with| NIL)
         (CLOSEF? (|fetch| F2 |of| |stream|))
         (|replace| F2 |of| |stream| |with| NIL)
         ;; remove  STREAM from the OPENFILELST field of %TWO-WAY-STREAM-DEVICE  or %ECHO-STREAM-DEVICE
         (|replace| (FDEV OPENFILELST) |of| STREAMDEVICE |with| (DREMOVE |stream| (|fetch| (FDEV OPENFILELST)
                                                                                    |of| STREAMDEVICE)))
         |stream|)))
```

(%TWO-WAY-STREAM-DEVICE-EOFP

```
  (LAMBDA (|stream|)                                    (* |smL| "14-Aug-85 16:47")
```

;;; The EOFP method for the two-way-stream device

```
    (\\EOFP (|fetch| F1 |of| |stream|))))
```

(%TWO-WAY-STREAM-DEVICE-READP

```
  (LAMBDA (STREAM FLG)                                  ; Edited 14-Apr-87 17:01 by bvm:
```

;;; The READP method for the two-way-stream device

```
    (READP (|fetch| F1 |of| STREAM)
           FLG)))
```

(%TWO-WAY-STREAM-DEVICE-BACKFILEPTR

```
  (LAMBDA (STREAM)                                      (* |hdj| "15-Sep-86 15:02")
    (\\BACKFILEPTR (|fetch| (STREAM F1) |of| STREAM))))
```

(%TWO-WAY-STREAM-DEVICE-FORCEOUTPUT

```
  (LAMBDA (|stream| |waitForFinish?|)                    (* |smL| "14-Aug-85 16:49")
```

;;; the FORCEOUTPUT method for the two-way-stream device

```
    (FORCEOUTPUT (|fetch| F2 |of| |stream|)
           |waitForFinish?|)))
```

(%TWO-WAY-STREAM-DEVICE-PEEKBIN

```
  (LAMBDA (|stream| |noErrorFlg?|)                       (* |smL| "14-Aug-85 16:46")
```

;;; The PEEKBIN method for the two-way-stream device

```
    (\\PEEKBIN (|fetch| F1 |of| |stream|)
           |noErrorFlg?|)))
```

(%TWO-WAY-STREAM-DEVICE-CHARSETFN

```
  (LAMBDA (STREAM NEWVALUE DONTMARKFILE)                 ; Edited  8-Dec-2023 15:41 by rmk
                                                         ; Edited 11-Sep-87 16:00 by bvm:
```

;; The charset method for two-way streams. Unclear what this is supposed to mean--let's apply it only to the input side (in which case newvalue is ;; senseless)

```
    (ACCESS-CHARSET (|fetch| (STREAM F1) |of| STREAM)
           NEWVALUE DONTMARKFILE)))
```

)

;; Fill-pointer streams

(CL:DEFUN %FILL-POINTER-STREAM-DEVICE-CLOSEFILE (STREAM &OPTIONAL ABORTFLAG)

;;; the CLOSEFILE method for the fill-pointer-string-stream device

```
   (|replace| F1 |of| STREAM |with| NIL)
   STREAM)
```

(CL:DEFUN %FILL-POINTER-STREAM-DEVICE-GETFILEPTR (STREAM)

```
   (CL:LENGTH (|fetch| (STREAM F1) |of| STREAM)))
```

```
(DECLARE\: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS %SYNONYM-STREAM-DEVICE %BROADCAST-STREAM-DEVICE %CONCATENATED-STREAM-DEVICE %TWO-WAY-STREAM-DEVICE
       %ECHO-STREAM-DEVICE \\FILL-POINTER-STREAM-DEVICE)
)
```

;; module initialization

(CL:DEFVAR *DEBUG-IO*)

(CL:DEFVAR *QUERY-IO*)

(CL:DEFVAR *TERMINAL-IO*)

(CL:DEFVAR *ERROR-OUTPUT*)

(CL:DEFVAR *STANDARD-OUTPUT*)

(CL:DEFVAR *STANDARD-INPUT*)

(CL:DEFUN %INITIALIZE-STANDARD-STREAMS ()

; Edited 3-Jul-2022 23:18 by rmk

;; Called when CLSTREAMS is loaded. Almost everything is same as *TERMINAL-IO* to start with.

```
   (CL:SETQ *QUERY-IO* (CL:MAKE-TWO-WAY-STREAM (CL:MAKE-SYNONYM-STREAM '\\LINEBUF.OFD)
                              (CL:MAKE-SYNONYM-STREAM '\\TERM.OFD)))
   (CL:SETQ *DEBUG-IO* *QUERY-IO*)
   (CL:SETQ *TERMINAL-IO* *QUERY-IO*)
   (CL:SETQ *ERROR-OUTPUT* (CL:MAKE-SYNONYM-STREAM '\\TERM.OFD)))
```

```
(DEFINEQ
```

(%INITIALIZE-CLSTREAM-TYPES

```
  (LAMBDA NIL                                            ; Edited  5-Jul-2022 21:20 by rmk
                                                         ; Edited  3-Jul-2022 23:57 by rmk
                                                         ; Edited 14-Apr-87 17:08 by bvm:
```

;; Initialize the CLSTREAMS package. This sets up some file devices for the functions make-two-way-stream-device, etc. See CLtL chapter 21

```
;; The input functions for broadcast streams should never be called, because they are guarded by the fact that the stream itself is output only.
(MAKE-EXTERNALFORMAT :BROADCAST-STREAM-FORMAT (FUNCTION SHOULDNT)
       (FUNCTION SHOULDNT)
       (FUNCTION SHOULDNT)
       (FUNCTION %BROADCAST-STREAM-OUTCHARFN))
(SETQ %BROADCAST-STREAM-DEVICE
 (|create| FDEV
        DEVICENAME _ 'BROADCAST-STREAM-DEVICE
        RESETABLE _ NIL
        RANDOMACCESSP _ NIL
        NODIRECTORIES _ T
        BUFFERED _ NIL
        PAGEMAPPED _ NIL
        FDBINABLE _ NIL
        FDBOUTABLE _ NIL
        FDEXTENDABLE _ NIL
        DEVICEINFO _ NIL
        HOSTNAMEP _ (FUNCTION NILL)
        EVENTFN _ (FUNCTION NILL)
        DIRECTORYNAMEP _ (FUNCTION NILL)
        REOPENFILE _ (FUNCTION NILL)
        CLOSEFILE _ (FUNCTION %BROADCAST-STREAM-DEVICE-CLOSEFILE)
        GETFILENAME _ (FUNCTION NILL)
        DELETEFILE _ (FUNCTION NILL)
        GENERATEFILES _ (FUNCTION \\GENERATENOFILES)
        RENAMEFILE _ (FUNCTION NILL)
        BIN _ (FUNCTION NILL)
        BOUT _ (FUNCTION %BROADCAST-STREAM-DEVICE-BOUT)
        PEEKBIN _ (FUNCTION NILL)
        READP _ (FUNCTION NILL)
        EOFP _ (FUNCTION TRUE)
        BLOCKIN _ (FUNCTION \\GENERIC.BINS)
        BLOCKOUT _ (FUNCTION NILL)
        FORCEOUTPUT _ (FUNCTION %BROADCAST-STREAM-DEVICE-FORCEOUTPUT)
        GETFILEINFO _ (FUNCTION NILL)
        SETFILEINFO _ (FUNCTION NILL)
        CHARSETFN _ (FUNCTION %BROADCAST-STREAM-DEVICE-CHARSETFN)
        DEFAULTEXTERNALFORMAT _ :BROADCAST-STREAM-FORMAT))
(MAKE-EXTERNALFORMAT :CONCATENATED-STREAM-FORMAT (FUNCTION %CONCATENATED-STREAM-INCCODEFN)
       (FUNCTION %CONCATENATED-STREAM-PEEKCCODEFN)
       (FUNCTION %CONCATENATED-STREAM-BACKCCODEFN)
       (FUNCTION SHOULDNT))
(SETQ %CONCATENATED-STREAM-DEVICE
 (|create| FDEV
        DEVICENAME _ 'CONCATENATED-STREAM-DEVICE
        RESETABLE _ NIL
        RANDOMACCESSP _ NIL
        NODIRECTORIES _ T
        BUFFERED _ NIL
        PAGEMAPPED _ NIL
        FDBINABLE _ NIL
        FDBOUTABLE _ NIL
        FDEXTENDABLE _ NIL
        DEVICEINFO _ NIL
        HOSTNAMEP _ (FUNCTION NILL)
        EVENTFN _ (FUNCTION NILL)
        DIRECTORYNAMEP _ (FUNCTION NILL)
        REOPENFILE _ (FUNCTION NILL)
        CLOSEFILE _ (FUNCTION %CONCATENATED-STREAM-DEVICE-CLOSEFILE)
        GETFILENAME _ (FUNCTION NILL)
        DELETEFILE _ (FUNCTION NILL)
        GENERATEFILES _ (FUNCTION \\GENERATENOFILES)
        RENAMEFILE _ (FUNCTION NILL)
        BIN _ (FUNCTION %CONCATENATED-STREAM-DEVICE-BIN)
        BOUT _ (FUNCTION NILL)
        PEEKBIN _ (FUNCTION %CONCATENATED-STREAM-DEVICE-PEEKBIN)
        READP _ (FUNCTION \\GENERIC.READP)
        BACKFILEPTR _ (FUNCTION %CONCATENATED-STREAM-DEVICE-BACKFILEPTR)
        EOFP _ (FUNCTION %CONCATENATED-STREAM-DEVICE-EOFP)
        BLOCKIN _ (FUNCTION \\GENERIC.BINS)
        BLOCKOUT _ (FUNCTION NILL)
        FORCEOUTPUT _ (FUNCTION NILL)
        GETFILEINFO _ (FUNCTION NILL)
        SETFILEINFO _ (FUNCTION NILL)
        CHARSETFN _ (FUNCTION %CONCATENATED-STREAM-DEVICE-CHARSETFN)
        DEFAULTEXTERNALFORMAT _ :CONCATENATED-STREAM-FORMAT))
(MAKE-EXTERNALFORMAT :TWO-WAY-STREAM-FORMAT (FUNCTION %TWO-WAY-STREAM-INCCODEFN)
       (FUNCTION %TWO-WAY-STREAM-PEEKCCODEFN)
       (FUNCTION %TWO-WAY-STREAM-BACKCCODEFN)
       (FUNCTION %TWO-WAY-STREAM-OUTCHARFN))
(SETQ %TWO-WAY-STREAM-DEVICE
 (|create| FDEV
        DEVICENAME _ 'TWO-WAY-STREAM-DEVICE
        RESETABLE _ NIL
        RANDOMACCESSP _ NIL
        NODIRECTORIES _ T
        BUFFERED _ NIL
```

```
            PAGEMAPPED _ NIL
            FDBINABLE _ NIL
            FDBOUTABLE _ NIL
            FDEXTENDABLE _ NIL
            INPUT-INDIRECTED _ T
            OUTPUT-INDIRECTED _ T
            DEVICEINFO _ NIL
            HOSTNAMEP _ (FUNCTION NILL)
            EVENTFN _ (FUNCTION NILL)
            DIRECTORYNAMEP _ (FUNCTION NILL)
            REOPENFILE _ (FUNCTION NILL)
            CLOSEFILE _ (FUNCTION %TWO-WAY-STREAM-DEVICE-CLOSEFILE)
            GETFILENAME _ (FUNCTION NILL)
            DELETEFILE _ (FUNCTION NILL)
            GENERATEFILES _ (FUNCTION \\GENERATENOFILES)
            RENAMEFILE _ (FUNCTION NILL)
            BIN _ (FUNCTION %TWO-WAY-STREAM-DEVICE-BIN)
            BOUT _ (FUNCTION %TWO-WAY-STREAM-DEVICE-BOUT)
            PEEKBIN _ (FUNCTION %TWO-WAY-STREAM-DEVICE-PEEKBIN)
            READP _ (FUNCTION %TWO-WAY-STREAM-DEVICE-READP)
            BACKFILEPTR _ (FUNCTION %TWO-WAY-STREAM-DEVICE-BACKFILEPTR)
            EOFP _ (FUNCTION %TWO-WAY-STREAM-DEVICE-EOFP)
            BLOCKIN _ (FUNCTION \\GENERIC.BINS)
            BLOCKOUT _ (FUNCTION \\GENERIC.BOUTS)
            FORCEOUTPUT _ (FUNCTION %TWO-WAY-STREAM-DEVICE-FORCEOUTPUT)
            GETFILEINFO _ (FUNCTION NILL)
            SETFILEINFO _ (FUNCTION NILL)
            CHARSETFN _ (FUNCTION %TWO-WAY-STREAM-DEVICE-CHARSETFN)
            INPUTSTREAM _ (FUNCTION %TWO-WAY-STREAM-DEVICE-INPUTSTREAM)
            OUTPUTSTREAM _ (FUNCTION %TWO-WAY-STREAM-DEVICE-OUTPUTSTREAM)
            DEFAULTEXTERNALFORMAT _ :TWO-WAY-STREAM-FORMAT))
    (MAKE-EXTERNALFORMAT :ECHO-STREAM-FORMAT (FUNCTION %ECHO-STREAM-INCCODEFN)
           (FUNCTION %TWO-WAY-STREAM-PEEKCCODEFN)
           (FUNCTION %TWO-WAY-STREAM-BACKCCODEFN)
           (FUNCTION %TWO-WAY-STREAM-OUTCHARFN))
    (SETQ %ECHO-STREAM-DEVICE (|create| FDEV |using| %TWO-WAY-STREAM-DEVICE DEVICENAME _ 'ECHO-STREAM-DEVICE BIN _
                                                    (FUNCTION %ECHO-STREAM-DEVICE-BIN)
                                                   DEFAULTEXTERNALFORMAT _ :ECHO-STREAM-FORMAT))
    (MAKE-EXTERNALFORMAT :SYNONYM-STREAM (FUNCTION %SYNONYM-STREAM-INCCODEFN)
           (FUNCTION %SYNONYM-STREAM-PEEKCCODEFN)
           (FUNCTION %SYNONYM-STREAM-BACKCCODEFN)
           (FUNCTION %SYNONYM-STREAM-OUTCHARFN))
    (SETQ %SYNONYM-STREAM-DEVICE
     (|create| FDEV
            DEVICENAME _ 'SYNONYM-STREAM-DEVICE
            RESETABLE _ NIL
            RANDOMACCESSP _ NIL
            NODIRECTORIES _ T
            BUFFERED _ NIL
            PAGEMAPPED _ NIL
            FDBINABLE _ NIL
            FDBOUTABLE _ NIL
            FDEXTENDABLE _ NIL
            DEVICEINFO _ NIL
            INPUT-INDIRECTED _ T
            OUTPUT-INDIRECTED _ T
            HOSTNAMEP _ (FUNCTION NILL)
            EVENTFN _ (FUNCTION NILL)
            DIRECTORYNAMEP _ (FUNCTION NILL)
            REOPENFILE _ (FUNCTION NILL)
            CLOSEFILE _ (FUNCTION %SYNONYM-STREAM-DEVICE-CLOSEFILE)
            GETFILENAME _ (FUNCTION NILL)
            DELETEFILE _ (FUNCTION NILL)
            GENERATEFILES _ (FUNCTION \\GENERATENOFILES)
            RENAMEFILE _ (FUNCTION NILL)
            BIN _ (FUNCTION %SYNONYM-STREAM-DEVICE-BIN)
            BOUT _ (FUNCTION %SYNONYM-STREAM-DEVICE-BOUT)
            PEEKBIN _ (FUNCTION %SYNONYM-STREAM-DEVICE-PEEKBIN)
            READP _ (FUNCTION %SYNONYM-STREAM-DEVICE-READP)
            BACKFILEPTR _ (FUNCTION %SYNONYM-STREAM-DEVICE-BACKFILEPTR)
            EOFP _ (FUNCTION %SYNONYM-STREAM-DEVICE-EOFP)
            BLOCKIN _ (FUNCTION \\GENERIC.BINS)
            BLOCKOUT _ (FUNCTION \\GENERIC.BOUTS)
            FORCEOUTPUT _ (FUNCTION %SYNONYM-STREAM-DEVICE-FORCEOUTPUT)
            GETFILEINFO _ (FUNCTION %SYNONYM-STREAM-DEVICE-GETFILEINFO)
            SETFILEINFO _ (FUNCTION %SYNONYM-STREAM-DEVICE-SETFILEINFO)
            INPUTSTREAM _ (FUNCTION %SYNONYM-STREAM-DEVICE-GET-INDIRECT-STREAM)
            OUTPUTSTREAM _ (FUNCTION %SYNONYM-STREAM-DEVICE-GET-INDIRECT-STREAM)
            CHARSETFN _ (FUNCTION %SYNONYM-STREAM-DEVICE-CHARSETFN)
            DEFAULTEXTERNALFORMAT _ :SYNONYM-STREAM))
    (SETQ \\FILL-POINTER-STREAM-DEVICE
     (|create| FDEV
            DEVICENAME _ 'FILL-POINTER-STREAM-DEVICE
            RESETABLE _ NIL
            RANDOMACCESSP _ NIL
            NODIRECTORIES _ T
            BUFFERED _ NIL
```

```
            PAGEMAPPED _ NIL
            FDBINABLE _ NIL
            FDBOUTABLE _ NIL
            FDEXTENDABLE _ NIL
            DEVICEINFO _ NIL
            HOSTNAMEP _ (FUNCTION NILL)
            EVENTFN _ (FUNCTION NILL)
            DIRECTORYNAMEP _ (FUNCTION NILL)
            OPENFILE _ (FUNCTION NILL)
            REOPENFILE _ (FUNCTION NILL)
            CLOSEFILE _ (FUNCTION %FILL-POINTER-STREAM-DEVICE-CLOSEFILE)
            GETFILENAME _ (FUNCTION NILL)
            DELETEFILE _ (FUNCTION NILL)
            GENERATEFILES _ (FUNCTION \\GENERATENOFILES)
            RENAMEFILE _ (FUNCTION NILL)
            BIN _ (FUNCTION \\ILLEGAL.DEVICEOP)
            BOUT _ (FUNCTION NILL)
            PEEKBIN _ (FUNCTION \\ILLEGAL.DEVICEOP)
            READP _ (FUNCTION \\ILLEGAL.DEVICEOP)
            EOFP _ (FUNCTION NILL)
            BLOCKIN _ (FUNCTION \\ILLEGAL.DEVICEOP)
            BLOCKOUT _ (FUNCTION \\GENERIC.BOUTS)
            FORCEOUTPUT _ (FUNCTION NILL)
            GETFILEPTR _ (FUNCTION %FILL-POINTER-STREAM-DEVICE-GETFILEPTR)
            SETFILEINFO _ (FUNCTION \\ILLEGAL.DEVICEOP)))))

)

(DECLARE\: DONTEVAL@LOAD DOCOPY

(%INITIALIZE-CLSTREAM-TYPES)

(%INITIALIZE-STANDARD-STREAMS)
)

(PUTPROPS CLSTREAMS FILETYPE CL:COMPILE-FILE)
```

FUNCTION INDEX

VARIABLE INDEX

MACRO INDEX

PROPERTY INDEX

SETF INDEX