File created: 11-Jun-90 16:24:56 **{DSK}<usr>local>lde>lispcore>library>HASH-FILE.;2**

changes to: (IL:VARS IL:HASH-FILECOMS)

previous date: 1-Mar-88 14:55:31 {DSK}<usr>local>lde>lispcore>library>HASH-FILE.;1

Read Table: XCL

Package: HASH-FILE

Format: XCCS


```
(IL:RPAQQ IL:HASH-FILECOMS
          ((IL:P (PROVIDE "HASH-FILE")
                 (EXPORT '(MAKE-HASH-FILE OPEN-HASH-FILE CLOSE-HASH-FILE COPY-HASH-FILE MAP-HASH-FILE
                                GET-HASH-FILE REM-HASH-FILE HASH-FILE-P HASH-FILE-COUNT HASH-FILE)
                         "HASH-FILE"))
           (IL:STRUCTURES HASH-FILE)
           (IL:FUNCTIONS %PRINT-HASH-FILE)
           (IL:VARIABLES BITS-PER-BYTE BYTES-PER-POINTER SIZE-POSITION COUNT-POSITION TABLE-POSITION
                  THE-NULL-POINTER)
           (IL:COMS

;;; public code

                  (IL:FUNCTIONS MAKE-HASH-FILE OPEN-HASH-FILE CLOSE-HASH-FILE COPY-HASH-FILE MAP-HASH-FILE
                         GET-HASH-FILE PUT-HASH-FILE REM-HASH-FILE)
                  (IL:SETFS GET-HASH-FILE)
                  (IL:VARIABLES *DELETE-OLD-VERSION-ON-REHASH* *REHASH-SIZE* *REHASH-THRESHOLD*)


;;; internal code

                  (IL:FUNCTIONS REHASH? REHASH KEY->TABLE-POINTER ADD-ENTRY ENSURE-STREAM-IS-OPEN NEXT-PRIME
                         WRITE-SIZE READ-SIZE WRITE-COUNT READ-COUNT WRITE-POINTER READ-POINTER NULL-POINTER?)
                  ;; conveniences
                  (IL:FUNCTIONS HISTOGRAM CONVERT))
           (IL:COMS

;;; default user code

                  (IL:FUNCTIONS HASH-OBJECT HASH-OBJECT-INTERNAL COMBINE)
                  (IL:VARIABLES *HASH-DEPTH*)
                  (IL:FUNCTIONS DEFAULT-READ-FN DEFAULT-PRINT-FN)
                  (IL:VARIABLES *READER-ENVIRONMENT*))
           (IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
                  IL:HASH-FILE)))

(PROVIDE "HASH-FILE")

(EXPORT '(MAKE-HASH-FILE OPEN-HASH-FILE CLOSE-HASH-FILE COPY-HASH-FILE MAP-HASH-FILE GET-HASH-FILE REM-HASH-FILE
          HASH-FILE-P HASH-FILE-COUNT HASH-FILE)
        "HASH-FILE")


(DEFSTRUCT (HASH-FILE (:COPIER COPY-HASH-FILE-INTERNAL)
                      (:CONSTRUCTOR MAKE-HASH-FILE-INTERNAL)
                      (:PRINT-FUNCTION %PRINT-HASH-FILE))
   "Like a hash-table but on a file instead of in memory"
   (STREAM NIL :TYPE STREAM)
   ;; open stream on the backing file
   (DIRECTION :INPUT :TYPE (MEMBER :INPUT :IO))
   ;; the direction that stream is open in
   (MONITOR (CREATE.MONITORLOCK "HASH-FILE")
          :TYPE MONITORLOCK)
   ;; should always be obtained before changing STREAM's position
   (SIZE NIL :TYPE INTEGER)
   ;; size of the table -- determines the range for key hashing
   (COUNT 0 :TYPE :INTEGER)
   ;; number of entries currently in the hash file
   (REHASH-SIZE *REHASH-SIZE* :TYPE FLOAT)
   ;; factor to increase size by when re-hashing
   (REHASH-THRESHOLD *REHASH-THRESHOLD* :TYPE FLOAT)
   ;; rehash when (= ENTRIES (* SIZE REHASH-THRESHOLD)
   (KEY-PRINT-FN 'DEFAULT-PRINT-FN :TYPE FUNCTION)
   ;; called with KEY and STREAM to write keys on the file
   (KEY-READ-FN 'DEFAULT-READ-FN :TYPE FUNCTION)
```

```
    ;; called with STREAM to read a key from the file
    (KEY-HASH-FN 'HASH-OBJECT :TYPE FUNCTION)
    ;; called with KEY and SIZE to obtain an integer in {0 .. SIZE-1}
    (KEY-COMPARE-FN 'EQUAL :TYPE FUNCTION)
    ;; called with two keys with same hash value to resolve collisions
    (VALUE-PRINT-FN 'DEFAULT-PRINT-FN :TYPE FUNCTION)
    ;; called with VALUE and STREAM to print values on file
    (VALUE-READ-FN 'DEFAULT-READ-FN :TYPE FUNCTION)
    ;; called with STREAM to read a value from the file
    )

(DEFUN %PRINT-HASH-FILE (HASH-FILE STREAM DEPTH)
   (FORMAT STREAM "#<Hash-File on ~A>" (LET* ((STREAM (HASH-FILE-STREAM HASH-FILE))
                                              (NAMESTRING (NAMESTRING (PATHNAME STREAM))))
                                         (IF NAMESTRING
                                             NAMESTRING
                                             STREAM))))

(DEFCONSTANT BITS-PER-BYTE 8)

(DEFCONSTANT BYTES-PER-POINTER 4)

(DEFCONSTANT SIZE-POSITION (* BYTES-PER-POINTER 0))

(DEFCONSTANT COUNT-POSITION (* BYTES-PER-POINTER 1))

(DEFCONSTANT TABLE-POSITION (* BYTES-PER-POINTER 2))

(DEFCONSTANT THE-NULL-POINTER 0)

;;; public code

(DEFUN MAKE-HASH-FILE
    ;; MAKE-HASH-TABLE equivalent for hash files
    ;; creates and returns a new hash file.
     (FILE-NAME SIZE &KEY (REHASH-SIZE *REHASH-SIZE*)
            (REHASH-THRESHOLD *REHASH-THRESHOLD*)
            (KEY-PRINT-FN 'DEFAULT-PRINT-FN)
            (KEY-READ-FN 'DEFAULT-READ-FN)
            (KEY-COMPARE-FN 'EQUAL)
            (KEY-HASH-FN 'HASH-OBJECT)
            (VALUE-PRINT-FN 'DEFAULT-PRINT-FN)
            (VALUE-READ-FN 'DEFAULT-READ-FN))
    (LET ((STREAM (OPEN FILE-NAME :DIRECTION :IO :IF-EXISTS :NEW-VERSION :ELEMENT-TYPE `(UNSIGNED-BYTE
                                                                                         ,BITS-PER-BYTE)))
          (REAL-SIZE (NEXT-PRIME SIZE)))
         ;; write the size & entries
         (WRITE-SIZE REAL-SIZE STREAM)
         (WRITE-COUNT 0 STREAM)
         ;; initialize table -- fill it with null pointers
         (DOTIMES (N REAL-SIZE)
             (WRITE-POINTER THE-NULL-POINTER STREAM))
         ;; make & return a HASH-FILE structure
         (MAKE-HASH-FILE-INTERNAL :STREAM STREAM :DIRECTION :IO :SIZE REAL-SIZE :COUNT 0 :REHASH-SIZE REHASH-SIZE
                :REHASH-THRESHOLD REHASH-THRESHOLD :KEY-PRINT-FN KEY-PRINT-FN :KEY-READ-FN KEY-READ-FN
                :KEY-COMPARE-FN KEY-COMPARE-FN :KEY-HASH-FN KEY-HASH-FN :VALUE-PRINT-FN VALUE-PRINT-FN
                :VALUE-READ-FN VALUE-READ-FN)))

(DEFUN OPEN-HASH-FILE
    ;; open an existing hash file
     (FILE-NAME &KEY (DIRECTION :INPUT)
            (REHASH-SIZE *REHASH-SIZE*)
            (REHASH-THRESHOLD *REHASH-THRESHOLD*)
            (KEY-PRINT-FN 'DEFAULT-PRINT-FN)
            (KEY-READ-FN 'DEFAULT-READ-FN)
            (KEY-COMPARE-FN 'EQUAL)
```

```
          (KEY-HASH-FN 'HASH-OBJECT)
          (VALUE-PRINT-FN 'DEFAULT-PRINT-FN)
          (VALUE-READ-FN 'DEFAULT-READ-FN))
   (CASE DIRECTION
       ((:INPUT :IO) )
       (OTHERWISE (ERROR "~S illegal arg.  Must be :INPUT or :IO" DIRECTION)))
   (LET ((STREAM (OPEN FILE-NAME :DIRECTION DIRECTION :IF-EXISTS :OVERWRITE :ELEMENT-TYPE
                       `(UNSIGNED-BYTE ,BITS-PER-BYTE))))
```

;; make & return a HASH-FILE structure

```
        (MAKE-HASH-FILE-INTERNAL :STREAM STREAM :DIRECTION DIRECTION :SIZE (READ-SIZE STREAM)
               :COUNT
               (READ-COUNT STREAM)
               :REHASH-SIZE REHASH-SIZE :REHASH-THRESHOLD REHASH-THRESHOLD :KEY-PRINT-FN KEY-PRINT-FN
               :KEY-READ-FN KEY-READ-FN :KEY-COMPARE-FN KEY-COMPARE-FN :KEY-HASH-FN KEY-HASH-FN :VALUE-PRINT-FN
               VALUE-PRINT-FN :VALUE-READ-FN VALUE-READ-FN)))
```

```
(DEFUN CLOSE-HASH-FILE (HASH-FILE &KEY ABORT)
```

;; close the stream

```
   (WITH.MONITOR (HASH-FILE-MONITOR HASH-FILE)
       (LET ((STREAM (HASH-FILE-STREAM HASH-FILE)))
            (CLOSE STREAM :ABORT ABORT)
            (PATHNAME STREAM))))
```

```
(DEFUN COPY-HASH-FILE (OLD-HASH-FILE NEW-FILE-NAME &OPTIONAL (NEW-SIZE NIL NEW-SIZE-SPECIFIED?))
```

;; make a new hashfile in NEW-FILE-NAME with the same contents as OLD-HASH-FILE. this will reclaim space lost in OLD-HASH-FILE. also used
;; by REHASH.

```
   (LET ((NEW-HASH-FILE (MAKE-HASH-FILE NEW-FILE-NAME (IF NEW-SIZE-SPECIFIED?
                                                          NEW-SIZE
```

;; default NEW-SIZE to the size of OLD-HASH-FILE

```
                                                          (HASH-FILE-SIZE OLD-HASH-FILE))
```

;; sure wish common lisp had a "using" construct...

```
                                :REHASH-SIZE
                                (HASH-FILE-REHASH-SIZE OLD-HASH-FILE)
                                :REHASH-THRESHOLD
                                (HASH-FILE-REHASH-THRESHOLD OLD-HASH-FILE)
                                :KEY-PRINT-FN
                                (HASH-FILE-KEY-PRINT-FN OLD-HASH-FILE)
                                :KEY-READ-FN
                                (HASH-FILE-KEY-READ-FN OLD-HASH-FILE)
                                :KEY-COMPARE-FN
                                (HASH-FILE-KEY-COMPARE-FN OLD-HASH-FILE)
                                :KEY-HASH-FN
                                (HASH-FILE-KEY-HASH-FN OLD-HASH-FILE)
                                :VALUE-PRINT-FN
                                (HASH-FILE-VALUE-PRINT-FN OLD-HASH-FILE)
                                :VALUE-READ-FN
                                (HASH-FILE-VALUE-READ-FN OLD-HASH-FILE))))
        (MAP-HASH-FILE #'(LAMBDA (KEY VALUE)
                           (SETF (GET-HASH-FILE KEY NEW-HASH-FILE)
                                 VALUE))
               OLD-HASH-FILE)
```

;; write it out for safety

```
        (CLOSE-HASH-FILE NEW-HASH-FILE)
```

;; return the new hash file

```
        NEW-HASH-FILE))
```

```
(DEFUN MAP-HASH-FILE (FN HASH-FILE)
```

;; calls FN on every KEY & VALUE pair in HASH-FILE

```
   (WITH.MONITOR (HASH-FILE-MONITOR HASH-FILE)
       (LET* ((STREAM (ENSURE-STREAM-IS-OPEN HASH-FILE))
              (SIZE (HASH-FILE-SIZE HASH-FILE))
              (LAST-POINTER (+ TABLE-POSITION (* BYTES-PER-POINTER (1- SIZE))))
              NEXT-POINTER)
```

;; loop over table

```
             (DO ((TABLE-POINTER TABLE-POSITION (+ TABLE-POINTER BYTES-PER-POINTER)))
                 ((> TABLE-POINTER LAST-POINTER))
```

;; loop down bucket

```
               (DO ((POINTER (READ-POINTER STREAM TABLE-POINTER)
                             NEXT-POINTER))
                   ((NULL-POINTER? POINTER)                    ; end of bucket or empty bucket

                    )
```

;; read & save next pointer

```
                (SETQ NEXT-POINTER (READ-POINTER STREAM POINTER))
                ;; call FN on KEY and VALUE read from file
                (FUNCALL FN (FUNCALL (HASH-FILE-KEY-READ-FN HASH-FILE)
                                     STREAM)
                       (FUNCALL (HASH-FILE-VALUE-READ-FN HASH-FILE)
                              STREAM)))))))

(DEFUN GET-HASH-FILE (KEY HASH-FILE &OPTIONAL (DEFAULT NIL))
    ;; GETHASH for hash files
    ;; returns the value stored under KEY in HASH-FILE, or DEFAULT if there is no value stored.  second value is T iff a value was found
    (WITH.MONITOR (HASH-FILE-MONITOR HASH-FILE)
        (LET ((STREAM (ENSURE-STREAM-IS-OPEN HASH-FILE))
              NEXT-POINTER)
            ;; loop down linked list in bucket
            (DO ((POINTER (READ-POINTER STREAM (KEY->TABLE-POINTER KEY HASH-FILE))
                          NEXT-POINTER))
                ((NULL-POINTER? POINTER)
                 ;; end of bucket (or empty bucket) - we lost
                 (VALUES DEFAULT NIL))
              ;; read & save next pointer
              (SETQ NEXT-POINTER (READ-POINTER STREAM POINTER))
              (WHEN ;; read key from file and compare with KEY
(FUNCALL (HASH-FILE-KEY-COMPARE-FN HASH-FILE)
       KEY
       (FUNCALL (HASH-FILE-KEY-READ-FN HASH-FILE)
              STREAM))
                   ;; they match -- we won!
                   (RETURN
                          ;; read & return value
                          (VALUES (FUNCALL (HASH-FILE-VALUE-READ-FN HASH-FILE)
                                           STREAM)
                                  T)))))))

(DEFUN PUT-HASH-FILE (KEY HASH-FILE VALUE)
    ;; SETF method for GET-HASH-FILE
    ;; stores a VALUE under KEY in HASH-FILE
    (WITH.MONITOR (HASH-FILE-MONITOR HASH-FILE)
        (LET ((TABLE-POINTER (KEY->TABLE-POINTER KEY HASH-FILE))
              (STREAM (ENSURE-STREAM-IS-OPEN HASH-FILE))
              NEXT-POINTER)
            ;; loop down bucket
            (DO* ((LAST-POINTER TABLE-POINTER POINTER)
                  ;; LAST-POINTER is location of POINTER
                  (POINTER (READ-POINTER STREAM TABLE-POINTER)
                           NEXT-POINTER))
                 ((NULL-POINTER? POINTER)
                  ;; end of bucket (or empty bucket) - nothing hashed under this key
                  ;; time to add a new entry to the hash file
                  (COND
                     ((REHASH? HASH-FILE)
                      ;; pointers are off if we rehashed -- have to start over
                      (PUT-HASH-FILE KEY HASH-FILE VALUE))
                     (T ;; just nconc a new entry onto the end of the bucket
                        (ADD-ENTRY HASH-FILE KEY VALUE LAST-POINTER THE-NULL-POINTER)
                        ;; increment and write out the count of objects
                        (WRITE-COUNT (INCF (HASH-FILE-COUNT HASH-FILE))
                                     STREAM))))
               ;; read & save the pointer to next in bucket
               (SETQ NEXT-POINTER (READ-POINTER STREAM POINTER))
               (WHEN ;; read key from file & compare with KEY
(FUNCALL (HASH-FILE-KEY-COMPARE-FN HASH-FILE)
       KEY
       (FUNCALL (HASH-FILE-KEY-READ-FN HASH-FILE)
              STREAM))
                    ;; they match - already something hashed under this key
                    ;; splice new entry into bucket, old entry out
                    (ADD-ENTRY HASH-FILE KEY VALUE LAST-POINTER NEXT-POINTER)
```

```
                    (RETURN)))
              ;; return VALUE
              VALUE)))

(DEFUN REM-HASH-FILE (KEY HASH-FILE)
    ;; REMHASH for hash files
    ;; removes the entry (if any)  for KEY from HASH-FILE.  returns T if there was one to remove.
    (WITH.MONITOR (HASH-FILE-MONITOR HASH-FILE)
        (LET ((TABLE-POINTER (KEY->TABLE-POINTER KEY HASH-FILE))
              (STREAM (ENSURE-STREAM-IS-OPEN HASH-FILE))
              NEXT-POINTER)
             ;; loop down bucket
             (DO* ((LAST-POINTER TABLE-POINTER POINTER)
                   ;; LAST-POINTER is location of POINTER
                   (POINTER (READ-POINTER STREAM TABLE-POINTER)
                            NEXT-POINTER))
                  ((NULL-POINTER? POINTER)
                   ;; end of bucket (or empty bucket) - nothing hashed under this key
                   'NIL)
                ;; read & save the pointer to next in bucket
                (SETQ NEXT-POINTER (READ-POINTER STREAM POINTER))
                (WHEN ;; read key from file & compare with KEY
(FUNCALL (HASH-FILE-KEY-COMPARE-FN HASH-FILE)
       KEY
       (FUNCALL (HASH-FILE-KEY-READ-FN HASH-FILE)
              STREAM))
                      ;; they match
                      ;; smash NEXT-POINTER into LAST-POINTER
                      (WRITE-POINTER NEXT-POINTER STREAM LAST-POINTER)
                      ;; decrement the count of entries in HASH-FILE
                      (WRITE-COUNT (DECF (HASH-FILE-COUNT HASH-FILE))
                                   STREAM)
                      (RETURN 'T))))))

(DEFSETF GET-HASH-FILE PUT-HASH-FILE)

(DEFVAR *DELETE-OLD-VERSION-ON-REHASH* NIL
   "if non-NIL then delete the old version of a hash file when rehashing")

(DEFVAR *REHASH-SIZE* 2.0
   "default REHASH-SIZE for hash files")

(DEFVAR *REHASH-THRESHOLD* 0.875
   "default REHASH-THRESHOLD for hash files")

;;; internal code

(DEFUN REHASH? (HASH-FILE)
    ;; check if it's time to rehash HASH-FILE.  if it is, then do so and return non-NIL
    (WHEN (>= (1+ (HASH-FILE-COUNT HASH-FILE))
              (* (HASH-FILE-SIZE HASH-FILE)
                 (HASH-FILE-REHASH-THRESHOLD HASH-FILE)))
        (REHASH HASH-FILE (ROUND (* (HASH-FILE-SIZE HASH-FILE)
                                    (HASH-FILE-REHASH-SIZE HASH-FILE))))
        T))

(DEFUN REHASH (HASH-FILE NEW-SIZE)
    ;; caution: assumes we're under hash file monitor
    (LET* ((OLD-PATHNAME (PATHNAME (HASH-FILE-STREAM HASH-FILE)))
           (TEMP-HASH-FILE (COPY-HASH-FILE HASH-FILE (MAKE-PATHNAME :VERSION :NEWEST :DEFAULTS (PATHNAME
                                                                                                OLD-PATHNAME
                                                                                                ))
                                           NEW-SIZE)))
          ;; close the old stream (before we lose pointer to it)
          (CLOSE-HASH-FILE HASH-FILE)
          ;; smash TEMP-HASH-FILE into HASH-FILE
```

```
        (UNINTERRUPTABLY
            (SETF (HASH-FILE-SIZE HASH-FILE)                    ; note: probably not the same as NEW-SIZE
                  (HASH-FILE-SIZE TEMP-HASH-FILE))
            (SETF (HASH-FILE-COUNT HASH-FILE)
                  (HASH-FILE-COUNT TEMP-HASH-FILE))
            (SETF (HASH-FILE-STREAM HASH-FILE)
                  (HASH-FILE-STREAM TEMP-HASH-FILE)))
        ;; our caller [PUT-HASH-FILE] expects the stream to be open
        (ENSURE-STREAM-IS-OPEN HASH-FILE)
        (IF *DELETE-OLD-VERSION-ON-REHASH* (DELETE-FILE OLD-PATHNAME))
        ;; return the hash file
        HASH-FILE))
```

(DEFMACRO **KEY->TABLE-POINTER** (KEY HASH-FILE)

;; return the file position for the head of the bucket which key hashes into. this is the guy who does the hashing.

;; caution: HASH-FILE is evaluated twice

```
`(+ TABLE-POSITION (* BYTES-PER-POINTER (FUNCALL (HASH-FILE-KEY-HASH-FN ,HASH-FILE)
                                                 ,KEY
                                                 (HASH-FILE-SIZE ,HASH-FILE)))))
```

(DEFUN **ADD-ENTRY** (HASH-FILE KEY VALUE LAST-POINTER LINK-POINTER)

;; write an entry at end of file, putting a pointer to it in LAST-POINTER and make it point to LINK-POINTER as next in bucket.

;; caution: we presume we've got the hash-file-monitor.

```
(LET* ((STREAM (HASH-FILE-STREAM HASH-FILE))
       (EOF-POINTER (FILE-LENGTH STREAM)))
        ;; first overwrite LAST-POINTER with a pointer to EOF
        (WRITE-POINTER EOF-POINTER STREAM LAST-POINTER)
        ;; write link to next bucket
        (WRITE-POINTER LINK-POINTER STREAM EOF-POINTER)
        ;; write the key
        (FUNCALL (HASH-FILE-KEY-PRINT-FN HASH-FILE)
                 KEY STREAM)
        ;; write the value
        (FUNCALL (HASH-FILE-VALUE-PRINT-FN HASH-FILE)
                 VALUE STREAM)
        ;; return value
        VALUE))
```

(DEFUN **ENSURE-STREAM-IS-OPEN** (HASH-FILE)

;; makes sure HASH-FILE's stream is open

;; caution: assumes we're under hash file monitor

```
(LET ((STREAM (HASH-FILE-STREAM HASH-FILE)))
     (IF (OPEN-STREAM-P STREAM)
         STREAM
         (SETF (HASH-FILE-STREAM HASH-FILE)
               (OPEN STREAM :DIRECTION (HASH-FILE-DIRECTION HASH-FILE)
                     :IF-EXISTS :OVERWRITE)))))
```

(DEFUN **NEXT-PRIME** (N)

;; return the next prime number greater than N

;; algorithm stolen from CDL's FIND1STPRIME in old HASH library

```
(LET (FOUND?)
     (DO ((P (LOGIOR N 1)
             (+ P 2)))
         ((DO* ((I 3 (+ I 2)))
               ((OR (AND (< I P)
                         (ZEROP (REM P I)))
                    (SETQ FOUND? (< P (* I I))))
                FOUND?))
          P))))
```

(DEFUN **WRITE-SIZE** (SIZE STREAM)

;; write SIZE to file as a pointer sized number

(**WRITE-POINTER** SIZE STREAM SIZE-POSITION))

(DEFUN **READ-SIZE** (STREAM)

;; read size from file as written by WRITE-SIZE

```
    (READ-POINTER STREAM SIZE-POSITION))

(DEFUN WRITE-COUNT (COUNT STREAM)
    ;; write COUNT to file as a pointer sized number
    (WRITE-POINTER COUNT STREAM COUNT-POSITION))

(DEFUN READ-COUNT (STREAM)
    ;; read count as written by WRITE-COUNT
    (READ-POINTER STREAM COUNT-POSITION))

(DEFUN WRITE-POINTER (POINTER STREAM &OPTIONAL POSITION)
    ;; write POINTER (a non-negative integer) as BYTES-PER-POINTER bytes on STREAM s.t. READ-POINTER can reconstruct it.  if POSITION is
    ;; specified then set STREAM's file position to it first.
    (WHEN (> (INTEGER-LENGTH POINTER)
             (* BYTES-PER-POINTER BITS-PER-BYTE))
          (ERROR "~S : pointer too large" POINTER))
    (WHEN POSITION (FILE-POSITION STREAM POSITION))
    (DOTIMES (N BYTES-PER-POINTER)
        (WRITE-BYTE (LDB (BYTE BITS-PER-BYTE (* N BITS-PER-BYTE))
                         POINTER)
                    STREAM))
    ;; return POINTER
    POINTER)

(DEFUN READ-POINTER (STREAM &OPTIONAL POSITION)
    ;; read from STREAM a positive integer written by WRITE-POINTER.  if POSITION is specified the file position will be set to it first.
    ;; read BYTES-PER-POINTER bytes from stream and return them as an integer.  this is the inverse of WRITE-P
    (WHEN POSITION (FILE-POSITION STREAM POSITION))
    (LET ((VALUE 0)
          BYTE)
         (DOTIMES (N BYTES-PER-POINTER)
             (SETQ BYTE (READ-BYTE STREAM))
             (WHEN (NOT (ZEROP BYTE))
                 ;; optimization: DPB is really slow w/ high bytes
                 (SETQ VALUE (DPB BYTE (BYTE BITS-PER-BYTE (* N BITS-PER-BYTE))
                                  VALUE))))
         VALUE))

(DEFMACRO NULL-POINTER? (POINTER)
    `(EQL ,POINTER THE-NULL-POINTER))

;; conveniences

(DEFUN HISTOGRAM (HASH-FILE)
    ;; return an ALIST of bucket depths dotted with number of occurences
    (WITH.MONITOR (HASH-FILE-MONITOR HASH-FILE)
        (LET* ((STREAM (ENSURE-STREAM-IS-OPEN HASH-FILE))
               (SIZE (HASH-FILE-SIZE HASH-FILE))
               (LAST-POINTER (+ TABLE-POSITION (* BYTES-PER-POINTER (1- SIZE))))
               NEXT-POINTER RESULT)
            ;; loop over table
            (DO ((TABLE-POINTER TABLE-POSITION (+ TABLE-POINTER BYTES-PER-POINTER)))
                ((> TABLE-POINTER LAST-POINTER))
              ;; loop down bucket
              (DO ((POINTER (READ-POINTER STREAM TABLE-POINTER)
                            NEXT-POINTER)
                   (BUCKET-LENGTH 0 (1+ BUCKET-LENGTH)))
                  ((NULL-POINTER? POINTER)
                   ;; end of bucket or empty bucket
                   ;; increment count for buckets of this length
                   (INCF (CDR (OR (ASSOC BUCKET-LENGTH RESULT)
                                  (CAR (PUSH (CONS BUCKET-LENGTH 0)
                                             RESULT))))))
                (SETQ NEXT-POINTER (READ-POINTER STREAM POINTER))))
            (SORT RESULT #'(LAMBDA (PAIR-1 PAIR-2)
                                   (< (CAR PAIR-1)
                                      (CAR PAIR-2)))))))
```

```
(DEFUN CONVERT (IL-HASH-FILE CL-HASH-FILE)
   "convert a HASH hash file into a HASH-FILE hash file"
   ;; first make sure HASH is loaded
   (IL:FILESLOAD (IL:SYSLOAD IL:FROM IL:LISPUSERS)
         HASH)
   (LET* ((OLD-HASH-FILE (IL:OPENHASHFILE IL-HASH-FILE))
          (NEW-HASH-FILE (MAKE-HASH-FILE CL-HASH-FILE (IL:HASHFILEPROP OLD-HASH-FILE 'IL:SIZE)))
          (ABORT 'T))
         (UNWIND-PROTECT
             (PROGN (IL:MAPHASHFILE OLD-HASH-FILE #'(LAMBDA (KEY VALUE)
                                                            (PUT-HASH-FILE KEY NEW-HASH-FILE VALUE)))
                    (SETQ ABORT 'NIL))
             (IL:CLOSEHASHFILE OLD-HASH-FILE)
             (CLOSE-HASH-FILE NEW-HASH-FILE :ABORT ABORT))))
```

;;; default user code

```
(DEFUN HASH-OBJECT (OBJECT RANGE)
```

;;; return an integer between 0 and (1- RANGE), inclusive

;;; objects which are EQUAL will return the same integer

```
   (1- (HASH-OBJECT-INTERNAL OBJECT RANGE 0)))

(DEFUN HASH-OBJECT-INTERNAL (OBJECT RANGE DEPTH)
   ;; recursively descend OBJECT, combining characters & integers at leaves with multiplication modulo RANGE.  never descend more than
   ;; *HASH-DEPTH* into a structure.
   ;; return an integer between 1 and RANGE, inclusive
   (IF (EQL DEPTH *HASH-DEPTH*)
       1
       (TYPECASE OBJECT
           (STRING (LET ((VALUE 1)
                         (LENGTH (LENGTH OBJECT)))
                        (DOTIMES (N (MIN LENGTH (- *HASH-DEPTH* DEPTH))
                                    (COMBINE RANGE VALUE LENGTH))
                            (SETF VALUE (COMBINE RANGE VALUE (CHAR-CODE (CHAR OBJECT N)))))))
           (SYMBOL
              ;; combine hash values of name and package name
              (COMBINE RANGE (HASH-OBJECT-INTERNAL (LET ((PKG (SYMBOL-PACKAGE OBJECT)))
                                                        (AND PKG (PACKAGE-NAME PKG)))
                                                   RANGE
                                                   (1+ DEPTH))
                       (HASH-OBJECT-INTERNAL (SYMBOL-NAME OBJECT)
                              RANGE
                              (1+ DEPTH))))
           (CONS
              ;; combine hash values of CAR and CDR
              (COMBINE RANGE (HASH-OBJECT-INTERNAL (CAR OBJECT)
                                                   RANGE
                                                   (1+ DEPTH))
                       (HASH-OBJECT-INTERNAL (CDR OBJECT)
                              RANGE
                              (1+ DEPTH))))
           (NUMBER (TYPECASE OBJECT
                       (INTEGER (COMBINE RANGE (ABS OBJECT)))
                       (FLOAT (MULTIPLE-VALUE-BIND (SIG EXPON)
                                  (INTEGER-DECODE-FLOAT OBJECT)
                                  (COMBINE RANGE SIG (ABS EXPON))))
                       (RATIO (COMBINE RANGE (ABS (NUMERATOR OBJECT))
                                       (DENOMINATOR OBJECT)))
                       (COMPLEX (COMBINE RANGE (HASH-OBJECT-INTERNAL (REALPART OBJECT)
                                                                     RANGE
                                                                     (1+ DEPTH))
                                               (HASH-OBJECT-INTERNAL (IMAGPART OBJECT)
                                                                     RANGE
                                                                     (1+ DEPTH))))))
           (CHARACTER (COMBINE RANGE (CHAR-CODE OBJECT)))
           (PATHNAME (HASH-OBJECT-INTERNAL (NAMESTRING OBJECT)
                              RANGE
                              (1+ DEPTH)))
           (BIT-VECTOR (LET ((VALUE 1)
                             (LENGTH (LENGTH OBJECT)))
                            (DOTIMES (N (MIN LENGTH (- *HASH-DEPTH* DEPTH))
                                        (COMBINE RANGE VALUE LENGTH))
                                (SETF VALUE (COMBINE RANGE VALUE (IF (ZEROP (BIT OBJECT N))
                                                                     0
                                                                     (EXPT 2 N)))))))
           (T ;; can't dependably read/print other objects
```

```
              (ERROR "Can't hash a(n) ~S" (TYPE-OF OBJECT))))))
```

```
(DEFMACRO COMBINE (RANGE &REST INTEGERS)
```

;;; combine non-negative integers returning an integer between 1 and RANGE inclusive (zeros are bad when combining with multiplication). we don't do ;;; the obvious (1+ (mod (* . integers) range)) to avoid making bignums.

;;; caution: RANGE may be evaluated many times.

```
   `(1+ (MOD ,(IF (ENDP (REST INTEGERS))
                  (FIRST INTEGERS)
                  `(* ,(FIRST INTEGERS)
                      (COMBINE ,RANGE ,@(REST INTEGERS))))
             ,RANGE)))
```

```
(DEFVAR *HASH-DEPTH* 17)
```

```
(DEFUN DEFAULT-READ-FN (STREAM)
```

```
   ;; default reader for hash files
   (WITH-READER-ENVIRONMENT *READER-ENVIRONMENT* (READ STREAM)))
```

```
(DEFUN DEFAULT-PRINT-FN (OBJECT STREAM)
```

```
   ;; default printer for hash files
   (WITH-READER-ENVIRONMENT *READER-ENVIRONMENT*
       (LET ((*PRINT-PRETTY* 'NIL))
            (PRINT OBJECT STREAM)))
   OBJECT)
```

```
(DEFVAR *READER-ENVIRONMENT* (MAKE-READER-ENVIRONMENT (FIND-PACKAGE "XCL")
                                        (FIND-READTABLE "XCL")
                                        10))
```

```
(IL:PUTPROPS IL:HASH-FILE IL:FILETYPE :COMPILE-FILE)
```

```
(IL:PUTPROPS IL:HASH-FILE IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE
                                                              (DEFPACKAGE "HASH-FILE" (:USE "LISP" "XCL")
                                                                     (:IMPORT WITH-READER-ENVIRONMENT
                                                                            MAKE-READER-ENVIRONMENT FIND-READTABLE
                                                                            UNINTERRUPTABLY WITH.MONITOR
                                                                            CREATE.MONITORLOCK MONITORLOCK))))
```

```
(IL:PUTPROPS IL:HASH-FILE IL:COPYRIGHT ("Venue & Xerox Corporation" 1987 1988 1990))
```

FUNCTION INDEX

CONSTANT INDEX

VARIABLE INDEX

MACRO INDEX

PROPERTY INDEX

SETF INDEX

STRUCTURE INDEX