File created: 18-Oct-93 14:37:58 **{Pele:mv:envos}<LispCore>Sources>CLTL2>CMLSEQBASICS.;2**

previous date: 29-Aug-91 16:36:55 {Pele:mv:envos}<LispCore>Sources>CLTL2>CMLSEQBASICS.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
(RPAQQ CMLSEQBASICSCOMS
       ((DECLARE%: EVAL@COMPILE DONTCOPY (FILES CMLSEQCOMMON))
        (FUNCTIONS CL:CONCATENATE CL:COPY-SEQ CL:ELT CL:LENGTH CL:MAKE-SEQUENCE CL:NREVERSE CL:REVERSE CL:SUBSEQ
               %%SETELT)
        (FUNCTIONS MAKE-SEQUENCE-OF-TYPE)
        (SETFS CL:ELT CL:SUBSEQ)
        (PROPS (CMLSEQBASICS FILETYPE))
        (DECLARE%: EVAL@COMPILE DONTCOPY DONTEVAL@LOAD (LOCALVARS . T))))

(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD CMLSEQCOMMON)
)


(CL:DEFUN CL:CONCATENATE (RESULT-TYPE &REST SEQUENCES)
   [LET [(RESULT (MAKE-SEQUENCE-OF-TYPE RESULT-TYPE (LET ((CNT 0))
                                                          (CL:DOLIST (SEQ SEQUENCES CNT)
                                                              (SETQ CNT (+ CNT (CL:LENGTH SEQ))))]
        (SEQ-DISPATCH RESULT [LET ((TAIL RESULT))
                                  (CL:DOLIST (SEQUENCE SEQUENCES RESULT)
                                      [SEQ-DISPATCH SEQUENCE (CL:DOLIST (ELEMENT SEQUENCE)
                                                                 (RPLACA TAIL ELEMENT)
                                                                 (SETQ TAIL (CDR TAIL)))
                                             (CL:DOTIMES (I (VECTOR-LENGTH SEQUENCE))
                                                 (RPLACA TAIL (CL:AREF SEQUENCE I))
                                                 (SETQ TAIL (CDR TAIL)))])]
               (LET ((INDEX 0))
                    (CL:DOLIST (SEQUENCE SEQUENCES RESULT)
                        [SEQ-DISPATCH SEQUENCE (CL:DOLIST (ELEMENT SEQUENCE)
                                                   (CL:SETF (CL:AREF RESULT INDEX)
                                                          ELEMENT)
                                                   (SETQ INDEX (CL:1+ INDEX)))
                               (CL:DOTIMES (I (VECTOR-LENGTH SEQUENCE))
                                   (CL:SETF (CL:AREF RESULT INDEX)
                                          (CL:AREF SEQUENCE I))
                                   (SETQ INDEX (CL:1+ INDEX)))])]))


(CL:DEFUN CL:COPY-SEQ (SEQUENCE)
   "Returns a copy of SEQUENCE which is EQUALP to SEQUENCE but not EQ."
   [LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (SEQ-DISPATCH SEQUENCE (FORWARD-LIST-LOOP SEQUENCE 0 LENGTH (INDEX CURRENT COPY TAIL)
                                      COPY
                                      (COLLECT-ITEM CURRENT COPY TAIL))
               (LET [(COPY (MAKE-VECTOR LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE SEQUENCE]
                    (COPY-VECTOR-SUBSEQ SEQUENCE 0 LENGTH COPY 0 LENGTH])


(CL:DEFUN CL:ELT (SEQUENCE INDEX)
                                                                             (* amd " 5-Jun-86 17:48")
   (CL:IF (NOT (< -1 INDEX (CL:LENGTH SEQUENCE)))
       (CL:ERROR 'INDEX-BOUNDS-ERROR :NAME SEQUENCE :INDEX INDEX))
   (SEQ-DISPATCH SEQUENCE (CL:NTH INDEX SEQUENCE)
          (CL:AREF SEQUENCE INDEX)))


(CL:DEFUN CL:LENGTH (SEQUENCE)
   (SEQ-DISPATCH SEQUENCE [LET ((SIZE 0)
                                (REST SEQUENCE))
                               (CL:LOOP (CL:IF (NOT (CL:CONSP REST))
                                                (RETURN SIZE))
                                      (SETQ REST (CDR REST))
                                      (SETQ SIZE (CL:1+ SIZE]
          (VECTOR-LENGTH SEQUENCE)))


(CL:DEFUN CL:MAKE-SEQUENCE (TYPE LENGTH &KEY (INITIAL-ELEMENT NIL INITIAL-ELEMENT-P))
   "Make a sequnce of the specified type"
   (CL:IF (EQ TYPE 'LIST)
       (CL:MAKE-LIST LENGTH :INITIAL-ELEMENT INITIAL-ELEMENT)
       (LET ((VECTOR (MAKE-SEQUENCE-OF-TYPE TYPE LENGTH)))
            (CL:IF INITIAL-ELEMENT-P (FILL-VECTOR-SUBSEQ VECTOR 0 LENGTH INITIAL-ELEMENT))
```

```
           VECTOR)))


(CL:DEFUN CL:NREVERSE (SEQUENCE)
   "Returns a sequence of the same elements in reverse order (the argument is destroyed)."
   [SEQ-DISPATCH SEQUENCE [LET ((REST SEQUENCE)
                                LIST-HEAD RESULT)
                               (CL:LOOP (CL:IF (NOT (CL:CONSP (SETQ LIST-HEAD REST)))
                                               (RETURN RESULT))
                                      (SETQ REST (CDR REST))
                                      (SETQ RESULT (RPLACD LIST-HEAD RESULT]
          (LET ((LENGTH (VECTOR-LENGTH SEQUENCE)))
               (CL:DO ((LEFT-INDEX 0 (CL:1+ LEFT-INDEX))
                       (RIGHT-INDEX (CL:1- LENGTH)
                              (CL:1- RIGHT-INDEX))
                       (HALF-LENGTH (LRSH LENGTH 1)))
                      ((EQL LEFT-INDEX HALF-LENGTH)
                       SEQUENCE)
                   (CL:ROTATEF (CL:AREF SEQUENCE LEFT-INDEX)
                          (CL:AREF SEQUENCE RIGHT-INDEX)))])


(CL:DEFUN CL:REVERSE (SEQUENCE)
   "Returns a new sequence containing the same elements but in reverse order."
   [SEQ-DISPATCH SEQUENCE [LET ((REST SEQUENCE)
                                RESULT)
                               (CL:LOOP (CL:IF (NOT (CL:CONSP REST))
                                               (RETURN RESULT))
                                      (CL:PUSH (CAR REST)
                                             RESULT)
                                      (SETQ REST (CDR REST]
          (LET ((LENGTH (VECTOR-LENGTH SEQUENCE)))
               (CL:DO ((RESULT (MAKE-VECTOR LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE SEQUENCE)))
                       (FORWARD-INDEX 0 (CL:1+ FORWARD-INDEX))
                       (BACKWARD-INDEX (CL:1- LENGTH)
                              (CL:1- BACKWARD-INDEX)))
                      ((EQL FORWARD-INDEX LENGTH)
                       RESULT)
                   (CL:SETF (CL:AREF RESULT FORWARD-INDEX)
                          (CL:AREF SEQUENCE BACKWARD-INDEX)))])


(CL:DEFUN CL:SUBSEQ (SEQUENCE START &OPTIONAL END)
   [LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (SEQ-DISPATCH SEQUENCE (FORWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT COPY TAIL)
                                      COPY
                                      (COLLECT-ITEM CURRENT COPY TAIL))
               (LET [(COPY (MAKE-VECTOR (- END START)
                                  :ELEMENT-TYPE
                                  (CL:ARRAY-ELEMENT-TYPE SEQUENCE]
                    (COPY-VECTOR-SUBSEQ SEQUENCE START END COPY 0])


(CL:DEFUN %%SETELT (SEQUENCE INDEX NEWVAL)
   (CL:IF (NOT (< -1 INDEX (CL:LENGTH SEQUENCE)))
       (CL:ERROR 'INDEX-BOUNDS-ERROR :NAME SEQUENCE :INDEX INDEX))
   (SEQ-DISPATCH SEQUENCE (CL:SETF (CL:NTH INDEX SEQUENCE)
                                 NEWVAL)
          (CL:SETF (CL:AREF SEQUENCE INDEX)
                 NEWVAL)))


(CL:DEFUN MAKE-SEQUENCE-OF-TYPE (TYPE LENGTH)
   [LET ((BROAD-TYPE (TYPE-SPECIFIER TYPE))
         TYPE-LENGTH)
        (CL:IF (EQ BROAD-TYPE 'LIST)
            (CL:MAKE-LIST LENGTH)
            [LET [(ELEMENT-TYPE (CASE BROAD-TYPE
                                    ((CL:SIMPLE-STRING STRING)
                                       (SETQ TYPE-LENGTH (AND (CL:CONSP TYPE)
                                                              (CL:SECOND TYPE)))
                                       'CL:STRING-CHAR)
                                    ((CL:SIMPLE-BIT-VECTOR CL:BIT-VECTOR)
                                       (SETQ TYPE-LENGTH (AND (CL:CONSP TYPE)
                                                              (CL:SECOND TYPE)))
                                       'BIT)
                                    (CL:SIMPLE-VECTOR
                                       (SETQ TYPE-LENGTH (AND (CL:CONSP TYPE)
                                                              (CL:SECOND TYPE)))
                                       T)
                                    ((CL:ARRAY CL:VECTOR CL:SIMPLE-ARRAY)
                                       (CL:IF (CL:CONSP TYPE)
                                           (LET ((ELT-TYPE (CADR TYPE)))
                                                (SETQ TYPE-LENGTH (CL:THIRD TYPE))
                                                (CL:IF (CL:CONSP TYPE-LENGTH)
                                                    (SETQ TYPE-LENGTH (CAR TYPE-LENGTH)))
                                                (CL:IF [AND ELT-TYPE (NOT (EQ ELT-TYPE 'CL:*]
                                                    ELT-TYPE
                                                    T))
                                           T)))]
                 (CL:IF (AND (CL:INTEGERP TYPE-LENGTH)
                             (NOT (EQUAL TYPE-LENGTH LENGTH)))
                        (CL:ERROR "~D is not the length of type ~s" LENGTH TYPE))
                 (CL:IF ELEMENT-TYPE
                     (MAKE-VECTOR LENGTH :ELEMENT-TYPE ELEMENT-TYPE)
                     (LET ((EXPANDER (CL::TYPE-EXPANDER BROAD-TYPE)))
                          (CL:IF EXPANDER
                              (MAKE-SEQUENCE-OF-TYPE (CL::TYPE-EXPAND TYPE EXPANDER)
                                     LENGTH)
                              (CL:ERROR "~S is a bad type specifier for sequences." TYPE))))])])

(CL:DEFSETF CL:ELT %%SETELT)

(CL:DEFSETF CL:SUBSEQ (SEQUENCE START &OPTIONAL END) (NEW-SEQUENCE)
   `(PROGN (CL:REPLACE ,SEQUENCE ,NEW-SEQUENCE :START1 ,START :END1 ,END)
           ,NEW-SEQUENCE))

(PUTPROPS CMLSEQBASICS FILETYPE CL:COMPILE-FILE)

(DECLARE%: EVAL@COMPILE DONTCOPY DONTEVAL@LOAD

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)

(PUTPROPS CMLSEQBASICS COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990 1991 1993))
```

{MEDLEY}<CLTL2>CMLSEQBASICS.;1   28-Jun-2024 18:34:02
-- Listed on 30-Jun-2024 13:12:08 --

FUNCTION INDEX

SETF INDEX

PROPERTY INDEX
