File created: 18-Oct-93 12:11:00 **{Pele:mv:envos}<LispCore>Sources>CLTL2>CMLFORMAT.;2**

previous date: 25-Oct-91 16:43:17 {Pele:mv:envos}<LispCore>Sources>CLTL2>CMLFORMAT.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
;;
;; Copyright (c) 1986, 1987, 1988, 1989, 1990, 1991, 1993 by Venue & Xerox Corporation.  All rights reserved.

(RPAQQ CMLFORMATCOMS
       (;; The FORMAT facility
        (STRUCTURES FORMAT-ERROR)
        (FUNCTIONS MAKE-DISPATCH-VECTOR SCALE-EXPONENT SCALE-EXPT-AUX)
        (FUNCTIONS FORMAT-ERROR)
        (VARIABLES *DIGIT-STRING* *DIGITS*)
        (FUNCTIONS FLONUM-TO-STRING FORMAT-WITH-CONTROL-STRING FORMAT-STRINGIFY-OUTPUT POP-FORMAT-ARG
               WITH-FORMAT-PARAMETERS NEXTCHAR FORMAT-PEEK FORMAT-FIND-CHAR)
        (FUNCTIONS FORMAT-GET-PARAMETER PARSE-FORMAT-OPERATION FORMAT-FIND-COMMAND CL:FORMAT FORMAT-MAYBE-PPRINT
               SUB-FORMAT)
        ;; Top-level entries in *FORMAT-DISPATCH-TABLE*
        (FUNCTIONS FORMAT-PRINT-BINARY FORMAT-PRINT-OCTAL FORMAT-PRINT-HEXADECIMAL FORMAT-PRINT-RADIX
               FORMAT-FIXED FORMAT-EXPONENTIAL FORMAT-GENERAL-FLOAT FORMAT-PRINC FORMAT-PRINT-CHARACTER
               FORMAT-PLURAL FORMAT-PRIN1 FORMAT-TAB FORMAT-TERPRI FORMAT-FRESHLINE FORMAT-SKIP-ARGUMENTS
               FORMAT-PAGE FORMAT-TILDE FORMAT-DOLLARS FORMAT-INDIRECTION FORMAT-ESCAPE FORMAT-SEMICOLON-ERROR
               FORMAT-CONDITION FORMAT-ITERATION FORMAT-JUSTIFICATION FORMAT-CAPITALIZATION FORMAT-NEWLINE
               FORMAT-JUST-WRITE FORMAT-PPRINT-NEWLINE FORMAT-PPRINT-INDENT FORMAT-CALL-FUNCTION)
        ;; Direct support for top-level entries
        (FUNCTIONS FORMAT-ROUND-COLUMNS FORMAT-EAT-WHITESPACE FORMAT-PRINT-NAMED-CHARACTER FORMAT-ADD-COMMAS
               FORMAT-WRITE-FIELD FORMAT-PRINT-NUMBER FORMAT-PRINT-SMALL-CARDINAL FORMAT-PRINT-CARDINAL
               FORMAT-PRINT-CARDINAL-AUX FORMAT-PRINT-ORDINAL FORMAT-PRINT-OLD-ROMAN FORMAT-PRINT-ROMAN
               FORMAT-PRINT-DECIMAL FORMAT-PRINT-RADIX-AUX FORMAT-FIXED-AUX FORMAT-EXPONENT-MARKER
               FORMAT-EXP-AUX FORMAT-GENERAL-AUX FORMAT-UNTAGGED-CONDITION FORMAT-FUNNY-CONDITION
               FORMAT-BOOLEAN-CONDITION FORMAT-DO-ITERATION FORMAT-GET-TRAILING-SEGMENTS FORMAT-GET-SEGMENTS
               FORMAT-PPRINT-LOGICAL-BLOCK FORMAT-CHECK-JUSTIFY)
        (FUNCTIONS CHARPOS WHITESPACE-CHAR-P MAKE-PAD-SEGS FORMAT-LOGICAL-FILL)
        (FUNCTIONS NAME-ARRAY)
        (VARIABLES *FORMAT-ARGUMENTS* *OUTER-FORMAT-ARGUMENTS* *FORMAT-CONTROL-STRING* *FORMAT-DISPATCH-TABLE*
               *FORMAT-INDEX* *FORMAT-LENGTH* *FORMAT-ORIGINAL-ARGUMENTS* *FORMAT-LOGICAL-BLOCK*
               *FORMAT-JUSTIFICATION* *FORMAT-INCOMPATIBLE-JUSTIFICATION* *FORMAT-COLON-ITERATION* CARDINAL-ONES
               CARDINAL-TENS CARDINAL-TEENS CARDINAL-PERIODS ORDINAL-ONES ORDINAL-TENS)
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA)
                                                                            (NLAML)
                                                                            (LAMA)))
        ;; Arrange to use the correct compiler.
        (PROP FILETYPE CMLFORMAT)))

;; The FORMAT facility

(DEFINE-CONDITION FORMAT-ERROR (CL:ERROR)
   (ARGS)
   [:REPORT (CL:LAMBDA (CONDITION *STANDARD-OUTPUT*)
                  (CL:FORMAT T "~%%~:{~@?~%%~}" (FORMAT-ERROR-ARGS CONDITION])

(DEFMACRO MAKE-DISPATCH-VECTOR (&BODY ENTRIES)
    ;; Hairy dispatch-table initialization macro. Takes a list of two-element lists (<character> <function-object>) and returns a vector char-code-limit
    ;; elements in length, where the Ith element is the function associated with the character with char-code I. If the character is case-convertible, it must
    ;; be given in only one case however, an entry in the vector will be made for both.
    [LET ((ENTRIES (CL:MAPCAN #'[CL:LAMBDA (X)
                                    (LET [(LOWER (CL:CHAR-DOWNCASE (CAR X)))
                                          (UPPER (CL:CHAR-UPCASE (CAR X]
                                         (CL:IF (CL:CHAR= LOWER UPPER)
                                             (LIST X)
                                             (LIST (CONS UPPER (CDR X))
                                                   (CONS LOWER (CDR X))))]
                          ENTRIES)))
          (CL:DO ([ENTRIES (SORT ENTRIES #'(CL:LAMBDA (X Y)
                                               (CL:CHAR< (CAR X)
                                                      (CAR Y]
                  (CHARIDX 0 (CL:1+ CHARIDX))
                  (COMTAB NIL (CONS (CL:IF ENTRIES
                                    (CL:IF (= (CL:CHAR-CODE (CAAR ENTRIES))
                                              CHARIDX)
                                        (CADR (pop ENTRIES))
                                        NIL)
                                    NIL)
```

```
                                    COMTAB)))
                [(= CHARIDX 256)
                 (CL:IF ENTRIES (CL:ERROR "Garbage in dispatch vector - ~S" ENTRIES))
                 `(CL:MAKE-ARRAY '(256)
                         :ELEMENT-TYPE T :INITIAL-CONTENTS ',(CL:NREVERSE COMTAB))])

(CL:DEFUN SCALE-EXPONENT (X)
   (SCALE-EXPT-AUX X 0.0 1.0 10.0 0.1 (CONSTANT (CL:LOG 2.0 10.0))))

(CL:DEFUN SCALE-EXPT-AUX (X ZERO ONE TEN ONE-TENTH LOG10-OF-2)
   (CL:MULTIPLE-VALUE-BIND (SIG EXPONENT)
       (CL:DECODE-FLOAT X)
     (DECLARE (IGNORE SIG))
     (CL:IF (= X ZERO)
         (CL:VALUES ZERO 1)
         [LET* [(E (ROUND (CL:* EXPONENT LOG10-OF-2)))
                (NEWX (CL:IF (MINUSP E)
                          (CL:* X TEN (CL:EXPT TEN (- -1 E)))
                          (/ X TEN (CL:EXPT TEN (CL:1- E)))))]
               (CL:DO ((D TEN (CL:* D TEN))
                       (Y NEWX (/ NEWX D))
                       (E E (CL:1+ E)))
                      [(< Y ONE)
                       (CL:DO ((M TEN (CL:* M TEN))
                               (Z Y (CL:* Z M))
                               (E E (CL:1- E)))
                              ((>= Z ONE-TENTH)
                               (CL:VALUES (/ X (CL:EXPT 10 E))
                                      E)))])])))

(CL:DEFUN FORMAT-ERROR (COMPLAINT &REST FORMAT-ARGS)
   [CL:ERROR 'FORMAT-ERROR :ARGS (LIST (LIST "~?~%~S~%~V@T^" COMPLAINT FORMAT-ARGS *FORMAT-CONTROL-STRING*
                                       (CL:1+ *FORMAT-INDEX*])

(CL:DEFVAR *DIGIT-STRING* (CL:MAKE-ARRAY 50 :ELEMENT-TYPE 'CL:STRING-CHAR :FILL-POINTER 0 :ADJUSTABLE T))

(CL:DEFCONSTANT *DIGITS* "0123456789")

(CL:DEFUN FLONUM-TO-STRING (X &OPTIONAL WIDTH DECPLACES SCALE FMIN)
   ;; Returns FIVE values: a string of digits with one decimal point, the string's length, T if the point is at the front, T if the point is at the end,  the index
   ;; of the point in the string
   (CL:IF (ZEROP X)
       (CL:VALUES "." 1 T T)
       [LET* ((REALDP (COND
                          (DECPLACES (CL:IF FMIN
                                            (MAX DECPLACES FMIN)
                                            DECPLACES))
                          (FMIN)))
              [ROUND (COND
                         [REALDP                                    ; Foo! Compute rounding place based on size of number and
                                                                    ; scale factor
                                 (MIN 9 (+ (DIGITSBDP X)
                                           REALDP
                                           (OR SCALE 0]
                         (WIDTH (MAX 1 (MIN 9 (CL:1- WIDTH]
               MANTSTR INTEXP)
              (CL:MULTIPLE-VALUE-SETQ (MANTSTR INTEXP)
                    (FLTSTR X ROUND))
              (CL:IF SCALE (CL:INCF INTEXP SCALE))
              ;; OK, now copy the digit string into *digit-string* with the decimal point set appropriately
              (CL:MACROLET [(STRPUT (C)
                                    `(CL:VECTOR-PUSH-EXTEND ,C *DIGIT-STRING*]
                     (LET* ((DIGITS (CL:LENGTH MANTSTR))
                            (INDEX -1)
                            (POINTPLACE (+ DIGITS INTEXP))
                            DECPNT)
                           ;; MANTSTR may have more digits than necessary; prune off its zeros. Doing this will lose if X is zero.
                           (IF (NOT (ZEROP X))
                               THEN (WHILE (AND (CL:PLUSP DIGITS)
                                                (CL:CHAR= (CL:CHAR MANTSTR (CL:1- DIGITS))
                                                          #\0))
                                           DO (CL:DECF DIGITS)
                                              (CL:INCF INTEXP)))
                           (CL:SETF (CL:FILL-POINTER *DIGIT-STRING*)
                                    0)
                           [COND
                              ((NOT (CL:PLUSP POINTPLACE))          ; .<digits>
                               (STRPUT #\.)
```

```
                                (CL:DOTIMES (I (- POINTPLACE))
                                    (STRPUT #\0))
                                (CL:DOTIMES (I DIGITS)
                                    (STRPUT (CL:CHAR MANTSTR I)))
                                (SETQ DECPNT 0))
                               ((MINUSP INTEXP)                    ; <digits>.<digits>
                                (CL:DOTIMES (I POINTPLACE)
                                    (STRPUT (CL:CHAR MANTSTR (CL:INCF INDEX))))
                                (STRPUT #\.)
                                (CL:DOTIMES (I (- INTEXP))
                                    (STRPUT (CL:CHAR MANTSTR (CL:INCF INDEX))))
                                (SETQ DECPNT (+ DIGITS INTEXP)))
                               (T                                  ; <digits>00.
                                  (CL:DOTIMES (I DIGITS)
                                      (STRPUT (CL:CHAR MANTSTR I)))
                                  (CL:DOTIMES (I INTEXP)
                                      (STRPUT #\0))
                                  (STRPUT #\.)
                                  (SETQ DECPNT (+ DIGITS INTEXP]
                            (SETQ DIGITS (CL:1- (CL:LENGTH *DIGIT-STRING*)))
                            (IF DECPLACES
                                THEN ;; Need extra 0s to get enough decimal places
                                     (CL:DOTIMES (I (- DECPLACES (- DIGITS DECPNT)))
                                         (STRPUT #\0)
                                         (CL:INCF DIGITS)))
                            (CL:VALUES *DIGIT-STRING* (CL:1+ DIGITS)
                                  (= DECPNT 0)
                                  (= DECPNT DIGITS)
                                 DECPNT]))
```

(DEFMACRO **FORMAT-WITH-CONTROL-STRING** (CONTROL-STRING &BODY FORMS)

;; This macro establishes the correct environment for processing an indirect control string. CONTROL-STRING is the string to process, and FORMS
;; are the forms to do the processing. They invariably will involve a call to SUB-FORMAT. CONTROL-STRING is guaranteed to be evaluated exactly
;; once.

```
`[LET ((STRING ,CONTROL-STRING))
      (CONDITION-CASE (LET ((*FORMAT-CONTROL-STRING* STRING)
                            (*FORMAT-LENGTH* (CL:LENGTH STRING))
                            (*FORMAT-INDEX* 0))
                       ,@FORMS)
              (FORMAT-ERROR (C)
                     (CL:ERROR 'FORMAT-ERROR :ARGS (CONS (LIST "While processing indirect control
                                                               string~%%~S~%%~V@T^" *FORMAT-CONTROL-STRING*
                                                               (CL:1+ *FORMAT-INDEX*))
                                                         (FORMAT-ERROR-ARGS C])
```

(DEFMACRO **FORMAT-STRINGIFY-OUTPUT** (&BODY FORMS)

;; This macro collects output to the standard output stream in a string. It used to try to avoid consing new string streams if possible.

```
`(CL:WITH-OUTPUT-TO-STRING (*STANDARD-OUTPUT*)
        ,@FORMS))
```