File created: 20-Aug-2021 20:44:50 **{DSK}<Users>kaplan>Local>medley3.5>git-medley>sources>CMLSTRING.;2**

previous date: 16-May-90 14:45:59 {DSK}<Users>kaplan>Local>medley3.5>git-medley>sources>CMLSTRING.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
;;
;; Copyright (c) 1985-1987, 1990, 2021 by Venue & Xerox Corporation.

(RPAQQ CMLSTRINGCOMS
       (;; run-time support
        (FUNCTIONS CL::SIMPLE-STRING= CL::SIMPLE-STRING-EQUAL)
        (FUNCTIONS %%STRING-BASE-COMPARE %%STRING-BASE-COMPARE-EQUAL %%STRING-UPCASE %%STRING-DOWNCASE)
        ;; User entry points
        (FUNCTIONS CL:MAKE-STRING CL:NSTRING-CAPITALIZE CL:NSTRING-DOWNCASE CL:NSTRING-UPCASE STRING
               CL:STRING-CAPITALIZE CL:STRING-DOWNCASE STRING-EQUAL CL:STRING-GREATERP CL:STRING-LEFT-TRIM
               CL:STRING-LESSP CL:STRING-NOT-EQUAL CL:STRING-NOT-GREATERP CL:STRING-NOT-LESSP
               CL:STRING-RIGHT-TRIM CL:STRING-TRIM CL:STRING-UPCASE CL:STRING/= CL:STRING< CL:STRING<=
               CL:STRING= CL:STRING> CL:STRING>=)
        (OPTIMIZERS CL:STRING= STRING-EQUAL)
        ;; Internal macros
        (DECLARE%: DONTCOPY DOEVAL@COMPILE (FUNCTIONS WITH-ONE-STRING WITH-ONE-STRING-ONLY WITH-STRING
                                                  WITH-TWO-UNPACKED-STRINGS %%UNPACK-STRING %%ADJUST-FOR-OFFSET
                                                  %%CHECK-BOUNDS %%PARSE-STRING-ARGS %%STRING-LENGTH))
        ;; Compiler options
        (PROP FILETYPE CMLSTRING)
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY (LOCALVARS . T))))

;; run-time support

(CL:DEFUN CL::SIMPLE-STRING= (STRING1 STRING2)
   [LET ((END1 (%%STRING-LENGTH STRING1))
         (END2 (%%STRING-LENGTH STRING2)))
        (CL:IF (EQ END1 END2)
            (LET (BASE1 BASE2 OFFSET1 OFFSET2 TYPENUMBER1 TYPENUMBER2)
                 (%%UNPACK-STRING STRING1 BASE1 OFFSET1 TYPENUMBER1)
                 (%%UNPACK-STRING STRING2 BASE2 OFFSET2 TYPENUMBER2)
                 (CL:IF (NOT (EQ 0 OFFSET1))
                     (SETQ END1 (+ END1 OFFSET1)))
                 (CL:IF (NOT (EQ 0 OFFSET2))
                     (SETQ END2 (+ END2 OFFSET2)))
                 (EQ END1 (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 OFFSET1 END1 OFFSET2
                                 END2))))])

(CL:DEFUN CL::SIMPLE-STRING-EQUAL (STRING1 STRING2)
   [LET ((END1 (%%STRING-LENGTH STRING1))
         (END2 (%%STRING-LENGTH STRING2)))
        (CL:IF (EQ END1 END2)
            (LET (BASE1 BASE2 OFFSET1 OFFSET2 TYPENUMBER1 TYPENUMBER2)
                 (%%UNPACK-STRING STRING1 BASE1 OFFSET1 TYPENUMBER1)
                 (%%UNPACK-STRING STRING2 BASE2 OFFSET2 TYPENUMBER2)
                 (CL:IF (NOT (EQ 0 OFFSET1))
                     (SETQ END1 (+ END1 OFFSET1)))
                 (CL:IF (NOT (EQ 0 OFFSET2))
                     (SETQ END2 (+ END2 OFFSET2)))
                 (EQ END1 (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 OFFSET1 END1
                                 OFFSET2 END2))))])

(CL:DEFUN %%STRING-BASE-COMPARE (BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2 END2)
    ;; Return index into base1 of first inequality
    ;; Can use eq for character comparisons because they are immediate datatypes. Can use eq for numeric equality since Indices are always in the
    ;; fixnum range
    (CL:IF (EQ START1 START2)
        (CL:DO ((INDEX START1 (CL:1+ INDEX))
                (ENDINDEX (MIN END1 END2)))
               ([OR (EQ INDEX ENDINDEX)
                    (NOT (EQ (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                             (%%ARRAY-READ BASE2 TYPENUMBER2 INDEX]
                INDEX))
        (CL:DO [(INDEX1 START1 (CL:1+ INDEX1))
                (INDEX2 START2 (CL:1+ INDEX2))
                (ENDINDEX (MIN END1 (+ START1 (- END2 START2]
               ([OR (EQ INDEX1 ENDINDEX)
```

```
                    (NOT (EQ (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX1)
                             (%%ARRAY-READ BASE2 TYPENUMBER2 INDEX2]
                INDEX1))))
```

(CL:DEFUN **%%STRING-BASE-COMPARE-EQUAL** (BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2 END2)

;; Return index into base1 of first case insensitive inequality

;; Can use eq for character comparisons because they are immediate datatypes.

;; Char-upcase has been expanded out and simplified below.

```
(CL:IF (EQ START1 START2)
    (CL:DO ((INDEX START1 (CL:1+ INDEX))
            (ENDINDEX (MIN END1 END2)))
           ([OR (EQ INDEX ENDINDEX)
                (NOT (EQ (%%CHAR-UPCASE-CODE (\LOLOC (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)))
                         (%%CHAR-UPCASE-CODE (\LOLOC (%%ARRAY-READ BASE2 TYPENUMBER2 INDEX]
            INDEX))
    (CL:DO [(INDEX1 START1 (CL:1+ INDEX1))
            (INDEX2 START2 (CL:1+ INDEX2))
            (ENDINDEX (MIN END1 (+ START1 (- END2 START2]
           ([OR (EQ INDEX1 ENDINDEX)
                (NOT (EQ (%%CHAR-UPCASE-CODE (\LOLOC (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX1)))
                         (%%CHAR-UPCASE-CODE (\LOLOC (%%ARRAY-READ BASE2 TYPENUMBER2 INDEX2]
            INDEX1))))
```

(CL:DEFUN **%%STRING-UPCASE** (**STRING** START END)

;; Assumes string is a string. Start and end define a subsequence. Destructively upcases string and returns it

```
(LET ((BASE (%%ARRAY-BASE STRING))
      (OFFSET (%%ARRAY-OFFSET STRING))
      (TYPENUMBER (%%ARRAY-TYPE-NUMBER STRING)))
     (%%ADJUST-FOR-OFFSET START END OFFSET)
     (CL:DO ((INDEX START (CL:1+ INDEX)))
            ((EQ INDEX END)
             STRING)
         (%%ARRAY-WRITE (CL:CHAR-UPCASE (%%ARRAY-READ BASE TYPENUMBER INDEX))
                BASE TYPENUMBER INDEX))))
```

(CL:DEFUN **%%STRING-DOWNCASE** (**STRING** START END)

;; Assumes string is a string. Start and end define a subsequence. Destructively downcases string and returns it

```
(LET ((BASE (%%ARRAY-BASE STRING))
      (OFFSET (%%ARRAY-OFFSET STRING))
      (TYPENUMBER (%%ARRAY-TYPE-NUMBER STRING)))
     (%%ADJUST-FOR-OFFSET START END OFFSET)
     (CL:DO ((INDEX START (CL:1+ INDEX)))
            ((EQ INDEX END)
             STRING)
         (%%ARRAY-WRITE (CL:CHAR-DOWNCASE (%%ARRAY-READ BASE TYPENUMBER INDEX))
                BASE TYPENUMBER INDEX))))
```

;; User entry points

(CL:DEFUN **CL:MAKE-STRING** (SIZE &KEY (INITIAL-ELEMENT NIL INITIAL-ELEMENT-P)
FATP)

```
   "Makes a simple string"
   (LET ((STRING (MAKE-VECTOR SIZE :ELEMENT-TYPE 'CL:STRING-CHAR :FATP FATP)))
        (CL:IF INITIAL-ELEMENT-P (FILL-ARRAY STRING INITIAL-ELEMENT))
        STRING))
```

(CL:DEFUN **CL:NSTRING-CAPITALIZE** (**STRING** &KEY START END)

```
   "Given a string, returns it with the first letter of every word in uppercase and all other letters in
   lowercase. A word is defined to be a sequence of alphanumeric characters delimited by non-alphanumeric
   characters"
   [WITH-ONE-STRING-ONLY STRING START END (CL:DO ((INDEX START (CL:1+ INDEX))
                                                  (ALPHA-P NIL)
                                                  (WAS-ALPHA-P NIL ALPHA-P)
                                                  CHAR)
                                                 ((EQ INDEX END)
                                                  STRING)
                                             (SETQ CHAR (CL:CHAR STRING INDEX))
                                             (SETQ ALPHA-P (CL:ALPHANUMERICP CHAR))
                                             (CL:SETF (CL:CHAR STRING INDEX)
                                                    (CL:IF (AND ALPHA-P (NOT WAS-ALPHA-P))
                                                        (CL:CHAR-UPCASE CHAR)
                                                        (CL:CHAR-DOWNCASE CHAR))))])
```

(CL:DEFUN **CL:NSTRING-DOWNCASE** (**STRING** &KEY START END)

```
   "Given a string, returns that string with all uppercase alphabetic characters converted to lowercase."
   (WITH-ONE-STRING-ONLY STRING START END (%%STRING-DOWNCASE STRING START END)))
```

```
(CL:DEFUN CL:NSTRING-UPCASE (STRING &KEY START END)
   "Given a string, returns that string with all lower case alphabetic characters converted to uppercase."
   (WITH-ONE-STRING-ONLY STRING START END (%%STRING-UPCASE STRING START END)))


(CL:DEFUN STRING (X)
   "Coerces X into a string. If X is a string, X is returned. If X is a symbol, X's pname is returned. If X is a
   character then a one element string containing that character is returned. If X cannot be coerced into a
   string, an error occurs."
   (CL:TYPECASE X
       (STRING X)
       (CL:SYMBOL (CL:SYMBOL-NAME X))
       (CL:CHARACTER (CL:MAKE-STRING 1 :INITIAL-ELEMENT X))
       (CL:OTHERWISE (CL:ERROR "~S cannot be coerced into a string" X))))


(CL:DEFUN CL:STRING-CAPITALIZE (STRING &KEY START END)
   "Given a string, returns a new string that is a copy of it with the first letter of every word in uppercase
   and all other letters in lowercase. A word is defined to be a sequence of alphanumeric characters delimited
   by non-alphanumeric characters"
   (WITH-ONE-STRING STRING START END (LET ((NEW-STRING (CL:MAKE-STRING SLEN)))
                                       (CL:DOTIMES (INDEX START)
                                           (CL:SETF (CL:SCHAR NEW-STRING INDEX)
                                                    (CL:CHAR STRING INDEX)))
                                       (CL:DO ((INDEX START (CL:1+ INDEX))
                                               (ALPHA-P NIL)
                                               (WAS-ALPHA-P NIL ALPHA-P)
                                               CHAR)
                                              ((EQ INDEX END))
                                           (SETQ CHAR (CL:CHAR STRING INDEX))
                                           (SETQ ALPHA-P (CL:ALPHANUMERICP CHAR))
                                           (CL:SETF (CL:SCHAR NEW-STRING INDEX)
                                                    (CL:IF (AND ALPHA-P (NOT WAS-ALPHA-P))
                                                           (CL:CHAR-UPCASE CHAR)
                                                           (CL:CHAR-DOWNCASE CHAR))))
                                       (CL:DO ((INDEX END (CL:1+ INDEX)))
                                              ((EQ INDEX SLEN))
                                           (CL:SETF (CL:SCHAR NEW-STRING INDEX)
                                                    (CL:CHAR STRING INDEX)))
                                       NEW-STRING)))


(CL:DEFUN CL:STRING-DOWNCASE (STRING &KEY START END)
   "Given a string, returns a new string that is a copy of it with all uppercase case alphabetic characters
   converted to lowercase."
   (WITH-ONE-STRING STRING START END (%%STRING-DOWNCASE (COPY-VECTOR STRING (CL:MAKE-STRING SLEN))
                                                        START END)))


(CL:DEFUN STRING-EQUAL (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Compare two strings for case insensitive equality"
   (CL:IF (OR START1 END1 START2 END2)
       [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
              (CL:IF (EQ SLEN1 SLEN2)
                  (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                         (EQ END1 (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1
                                                               END1 START2 END2))))]
       (CL::SIMPLE-STRING-EQUAL STRING1 STRING2)))


(CL:DEFUN CL:STRING-GREATERP (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Case insensitive version of STRING>"
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1
                                                            START2 END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ REL-INDEX SLEN2)
                           (CL:IF (> SLEN1 SLEN2)
                                  (- INDEX OFFSET1)))
                          ((EQ INDEX END1)
                           NIL)
                          ((CL:CHAR-GREATERP (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                  (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1])


(CL:DEFUN CL:STRING-LEFT-TRIM (CHAR-BAG STRING)
   "Trim only on left"
   (WITH-STRING STRING (LET [(LEFT-END (CL:DO ((INDEX 0 (CL:1+ INDEX)))
                                              ((OR (EQ INDEX SLEN)
                                                   (NOT (CL:FIND (CL:CHAR STRING INDEX)
                                                                 CHAR-BAG)))
                                               INDEX))]
                         (CL:SUBSEQ STRING LEFT-END SLEN))))
```

```
(CL:DEFUN CL:STRING-LESSP (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Case insensitive version of STRING<"
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1
                                    START2 END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ INDEX END1)
                           (CL:IF (< SLEN1 SLEN2)
                                  (- INDEX OFFSET1)))
                          ((EQ (- INDEX START1)
                               SLEN2)
                           NIL)
                          ((CL:CHAR-LESSP (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                  (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1)])

(CL:DEFUN CL:STRING-NOT-EQUAL (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Compare two string for case insensitive equality"
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET ((INDEX (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1
                                   START2 END2)))
                      (CL:IF (AND (EQ INDEX END1)
                                  (EQ SLEN1 SLEN2))
                          NIL
                          (- INDEX OFFSET1))])

(CL:DEFUN CL:STRING-NOT-GREATERP (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Case insensitive version of STRING<="
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1
                                    START2 END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ INDEX END1)
                           (- INDEX OFFSET1))
                          ((EQ (- INDEX START1)
                               SLEN2)
                           NIL)
                          ((CL:CHAR-NOT-GREATERP (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                  (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1])

(CL:DEFUN CL:STRING-NOT-LESSP (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Case insensitive version of STRING>="
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE-EQUAL BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1
                                    START2 END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ REL-INDEX SLEN2)
                           (- INDEX OFFSET1))
                          ((EQ INDEX END1)
                           NIL)
                          ((CL:CHAR-NOT-LESSP (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                  (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1])

(CL:DEFUN CL:STRING-RIGHT-TRIM (CHAR-BAG STRING)
   "Trim only on right"
   (WITH-STRING STRING (LET [(RIGHT-END (CL:DO ((INDEX (CL:1- SLEN)
                                                       (CL:1- INDEX)))
                                               ((OR (< INDEX 0)
                                                    (NOT (CL:FIND (CL:CHAR STRING INDEX)
                                                                  CHAR-BAG)))
                                                (CL:1+ INDEX)))]
                             (CL:SUBSEQ STRING 0 RIGHT-END))))

(CL:DEFUN CL:STRING-TRIM (CHAR-BAG STRING)
   ;; Given a set of characters (a list or string) and a string, returns a copy of the string with the characters in the set removed from both ends.
   (WITH-STRING STRING (LET* [(LEFT-END (CL:DO ((INDEX 0 (CL:1+ INDEX)))
                                               ((OR (EQ INDEX SLEN)
                                                    (NOT (CL:FIND (CL:CHAR STRING INDEX)
                                                                  CHAR-BAG)))
                                                INDEX)))
```

```
                              (RIGHT-END (CL:DO ((INDEX (CL:1- SLEN)
                                                        (CL:1- INDEX)))
                                                 ((OR (< INDEX LEFT-END)
                                                      (NOT (CL:FIND (CL:CHAR STRING INDEX)
                                                                    CHAR-BAG)))
                                                  (CL:1+ INDEX)))]
                                         (CL:SUBSEQ STRING LEFT-END RIGHT-END))))


(CL:DEFUN CL:STRING-UPCASE (STRING &KEY START END)
   "Given a string, returns a new string that is a copy of it with all lower case alphabetic characters
   converted to uppercase."
   (WITH-ONE-STRING STRING START END (%%STRING-UPCASE (COPY-VECTOR STRING (CL:MAKE-STRING SLEN))
                                                      START END)))


(CL:DEFUN CL:STRING/= (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Compare two strings for case sensitive inequality"
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET ((INDEX (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2
                                                     END2)))
                      (CL:IF (AND (EQ INDEX END1)
                                  (EQ SLEN1 SLEN2))
                          NIL
                          (- INDEX OFFSET1))])


(CL:DEFUN CL:STRING< (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "A string A is less than a string B if in the first position in which they differ the character of A is less
   than the corresponding character of B according to char< or if string A is a proper prefix of string B (of
   shorter length and matching in all the characters of A). Returns either NIL or an index into STRING1"
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2
                                                      END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ INDEX END1)
                           (CL:IF (< SLEN1 SLEN2)
                                  (- INDEX OFFSET1)))
                          ((EQ (- INDEX START1)
                               SLEN2)
                           NIL)
                          ((CL:CHAR< (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                     (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1)])


(CL:DEFUN CL:STRING<= (STRING1 STRING2 &KEY START1 END1 START2 END2)
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2
                                                      END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ INDEX END1)
                           (- INDEX OFFSET1))
                          ((EQ (- INDEX START1)
                               SLEN2)
                           NIL)
                          ((CL:CHAR<= (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                      (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1)])


(CL:DEFUN CL:STRING= (STRING1 STRING2 &KEY START1 END1 START2 END2)
   "Compare two strings for case sensitive equality"
   (CL:IF (OR START1 END1 START2 END2)
       [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
              (CL:IF (EQ SLEN1 SLEN2)
                  (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                         (EQ END1 (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1
                                                         START2 END2))))]
       (CL::SIMPLE-STRING= STRING1 STRING2)))


(CL:DEFUN CL:STRING> (STRING1 STRING2 &KEY START1 END1 START2 END2)
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2
                                                      END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ REL-INDEX SLEN2)
                           (CL:IF (> SLEN1 SLEN2)
                                  (- INDEX OFFSET1)))
```

```
                        ((EQ INDEX END1)
                         NIL)
                        ((CL:CHAR> (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                         (- INDEX OFFSET1])


(CL:DEFUN CL:STRING>= (STRING1 STRING2 &KEY START1 END1 START2 END2)
   [%%PARSE-STRING-ARGS STRING1 STRING2 START1 END1 START2 END2
          (WITH-TWO-UNPACKED-STRINGS STRING1 STRING2 START1 END1 START2 END2
                 (LET* ((INDEX (%%STRING-BASE-COMPARE BASE1 TYPENUMBER1 BASE2 TYPENUMBER2 START1 END1 START2
                                    END2))
                        (REL-INDEX (- INDEX START1)))
                       (COND
                          ((EQ REL-INDEX SLEN2)
                           (- INDEX OFFSET1))
                          ((EQ INDEX END1)
                           NIL)
                          ((CL:CHAR>= (%%ARRAY-READ BASE1 TYPENUMBER1 INDEX)
                                  (%%ARRAY-READ BASE2 TYPENUMBER2 (+ START2 REL-INDEX)))
                           (- INDEX OFFSET1])


(DEFOPTIMIZER CL:STRING= (STRING1 STRING2 &REST OPTIONS)
                         (CL:IF OPTIONS
                             'COMPILER:PASS
                             `(CL::SIMPLE-STRING= ,STRING1 ,STRING2)))


(DEFOPTIMIZER STRING-EQUAL (STRING1 STRING2 &REST OPTIONS)
                           (CL:IF OPTIONS
                               'COMPILER:PASS
                               `(CL::SIMPLE-STRING-EQUAL ,STRING1 ,STRING2)))
```

;; Internal macros

```
(DECLARE%: DONTCOPY DOEVAL@COMPILE


(DEFMACRO WITH-ONE-STRING (STRING START END &REST FORMS)
   "WITH-ONE-STRING is used to set up string operations. The keywords are parsed, and STRING is coerced into a
   string. SLEN is bound to the string length"
   `(LET [(SLEN (VECTOR-LENGTH (SETQ ,STRING (STRING ,STRING]
         (%%CHECK-BOUNDS ,START ,END SLEN)
         ,@FORMS))


(DEFMACRO WITH-ONE-STRING-ONLY (STRING START END &REST FORMS)
   ;; Like WITH-ONE-STRING but only strings allowed
   `(PROGN (CL:IF (NOT (CL:STRINGP ,STRING))
               (CL:ERROR 'CONDITIONS:SIMPLE-TYPE-ERROR :EXPECTED-TYPE 'STRING :CULPRIT ,STRING))
           (LET [(SLEN (VECTOR-LENGTH ,STRING]
                (%%CHECK-BOUNDS ,START ,END SLEN)
                ,@FORMS)))


(DEFMACRO WITH-STRING (STRING &REST FORMS)
   ;; WITH-STRING is like WITH-ONE-STRING, but doesn't process keywords
   `(LET [(SLEN (VECTOR-LENGTH (SETQ ,STRING (STRING ,STRING]
         ,@FORMS))


(DEFMACRO WITH-TWO-UNPACKED-STRINGS (STRING1 STRING2 START1 END1 START2 END2 &REST FORMS)
   ;; Used to set up string comparison operations. String1 and string2 are unpacked and start1, end1, start2, end2 are adjusted for non-zero offsets.
   ;; Base1 and base2, typenumber1, typenumber2 , offset1 and offset2 are bound to the appropriate unpacked quantities
   `(LET (BASE1 BASE2 OFFSET1 OFFSET2 TYPENUMBER1 TYPENUMBER2)
         (%%UNPACK-STRING ,STRING1 BASE1 OFFSET1 TYPENUMBER1)
         (%%UNPACK-STRING ,STRING2 BASE2 OFFSET2 TYPENUMBER2)
         (%%ADJUST-FOR-OFFSET ,START1 ,END1 OFFSET1)
         (%%ADJUST-FOR-OFFSET ,START2 ,END2 OFFSET2)
         ,@FORMS))


(DEFMACRO %%UNPACK-STRING (OBJECT BASE OFFSET TYPENUMBER &OPTIONAL LENGTH)
   `[COND
       [(CL:SYMBOLP ,OBJECT)
        (SETQ ,BASE (fetch (LITATOM PNAMEBASE) of ,OBJECT))
        (SETQ ,OFFSET 1)
        (SETQ ,TYPENUMBER (CL:IF (fetch (LITATOM FATPNAMEP) of ,OBJECT)
                              %%FAT-CHAR-TYPENUMBER
                              %%THIN-CHAR-TYPENUMBER))
        ,@(CL:IF LENGTH
              `[(SETQ ,LENGTH (fetch (LITATOM PNAMELENGTH) of ,OBJECT])]
       (T [COND
```

```
            [(%%ONED-ARRAY-P ,OBJECT)
             (SETQ ,BASE (fetch (ARRAY-HEADER BASE) of ,OBJECT))
             (SETQ ,OFFSET (fetch (ARRAY-HEADER OFFSET) of ,OBJECT))
             (SETQ ,TYPENUMBER (fetch (ARRAY-HEADER TYPE-NUMBER) of ,OBJECT]
            (T (SETQ ,BASE (%%ARRAY-BASE ,OBJECT))
               (SETQ ,OFFSET (%%ARRAY-OFFSET ,OBJECT))
               (SETQ ,TYPENUMBER (%%ARRAY-TYPE-NUMBER ,OBJECT]
          ,@(CL:IF LENGTH
                `[(SETQ ,LENGTH (fetch (ARRAY-HEADER FILL-POINTER) of ,OBJECT])])

(DEFMACRO %%ADJUST-FOR-OFFSET (START END OFFSET)
   `(CL:WHEN (NOT (EQ 0 ,OFFSET))
        (SETQ ,START (+ ,START ,OFFSET))
        (SETQ ,END (+ ,END ,OFFSET))))

(DEFMACRO %%CHECK-BOUNDS (START END LENGTH)
   `[PROGN [COND
              ((NULL ,END)
               (SETQ ,END ,LENGTH))
              ((> ,END ,LENGTH)
               (CL:ERROR "End out of bounds: ~S" ,END]
           (COND
              ((NULL ,START)
               (SETQ ,START 0))
              ((NOT (<= 0 ,START ,END))
               (CL:ERROR "Improper substring bounds: ~s ~s" ,START ,END])

(DEFMACRO %%PARSE-STRING-ARGS (STRING1 STRING2 START1 END1 START2 END2 &REST FORMS)
    ;; Used to set up string comparison operations. The keywords are defaulted, bounds are checked and  Slen1 and Slen1 are bound to substring
    ;; lengths"
   `(LET [(SLEN1 (%%STRING-LENGTH ,STRING1))
          (SLEN2 (%%STRING-LENGTH ,STRING2]
         (%%CHECK-BOUNDS ,START1 ,END1 SLEN1)
         (%%CHECK-BOUNDS ,START2 ,END2 SLEN2)
         (SETQ SLEN1 (- ,END1 ,START1))
         (SETQ SLEN2 (- ,END2 ,START2))
        ,@FORMS))

(DEFMACRO %%STRING-LENGTH (STRING)
   `(COND
       ((%%STRINGP ,STRING)
        (fetch (ARRAY-HEADER FILL-POINTER) of ,STRING))
       ((CL:SYMBOLP ,STRING)
        (fetch (LITATOM PNAMELENGTH) of ,STRING))
       [(CL:CHARACTERP ,STRING)
        (VECTOR-LENGTH (SETQ ,STRING (STRING ,STRING]
       (T (CL:ERROR 'XCL:TYPE-MISMATCH :EXPECTED-TYPE '(OR STRING CL:SYMBOL CL:CHARACTER)
                 :NAME
                 ,STRING :VALUE ,STRING :MESSAGE "a string, symbol or character"))))
)

;; Compiler options

(PUTPROPS CMLSTRING FILETYPE CL:COMPILE-FILE)

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)

(PUTPROPS CMLSTRING COPYRIGHT ("Venue & Xerox Corporation" 1985 1986 1987 1990 2021))
```

FUNCTION INDEX

MACRO INDEX

OPTIMIZER INDEX

PROPERTY INDEX