

File created: 16-May-90 14:31:36 {DSK}<usr>local>lde>lispcore>sources>CMLSEQMAPPERS.;2

changes to: (VARS CMLSEQMAPPERSCOMS)

previous date: 1-Jun-87 11:21:23 {DSK}<usr>local>lde>lispcore>sources>CMLSEQMAPPERS.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

::
:: Copyright (c) 1986, 1987, 1990 by Venue & Xerox Corporation. All rights reserved.

(RPAQQ **CMLSEQMAPPERSCOMS**

```
((DECLARE%: EVAL@COMPILE DONTCOPY (FILES CMLSEQCOMMON))
 (FUNCTIONS %%FILL-SLICE %%MAP-FOR-EFFECT %%MAP-FOR-EFFECT-MULTIPLE %%MAP-FOR-EFFECT-SINGLE
  %%MAP-FOR-RESULT-MULTIPLE %%MAP-FOR-RESULT-SINGLE %%MIN-SEQUENCE-LENGTH CL:MAP)
 ;; For compatibility with old optimizers
 (FUNCTIONS %%MAP-SINGLE-FOR-EFFECT %%MAP-SINGLE-TO-LIST %%MAP-SINGLE-TO-SIMPLE %%MAP-TO-LIST
  %%MAP-TO-SIMPLE)
 (OPTIMIZERS CL:MAP)
 (FUNCTIONS %%SOME-MULTIPLE %%SOME-SINGLE %%EVERY-MULTIPLE %%EVERY-SINGLE %%NOTANY-MULTIPLE
  %%NOTANY-SINGLE %%NOTEVERY-MULTIPLE %%NOTEVERY-SINGLE CL:SOME CL:EVERY CL:NOTANY CL:NOTEVERY)
 ;; For compatibility with old optimizers
 (P (MOVD '%%SOME-SINGLE '%%SINGLE-SOME)
 (MOVD '%%EVERY-SINGLE '%%SINGLE-EVERY)
 (MOVD '%%NOTEVERY-SINGLE '%%SINGLE-NOTEVERY)
 (MOVD '%%NOTANY-SINGLE '%%SINGLE-NOTANY))
 (OPTIMIZERS CL:SOME CL:EVERY CL:NOTANY CL:NOTEVERY)
 (FUNCTIONS REDUCE-FROM-END REDUCE-FROM-START CL:REDUCE)
 (PROP FILETYPE CMLSEQMAPPERS)
 (DECLARE%: DONTVAL@LOAD DONTCOPY DOEVAL@COMPILE (LOCALVARS . T))))
```

(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD CMLSEQCOMMON)
)

(DEFMACRO **%%FILL-SLICE** (INDEX SLICE SEQUENCES)

```
`(CL:DO ((%SUBSLICE ,SLICE (CDR %%SUBSLICE))
 (%SUBSEQ ,SEQUENCES (CDR %%SUBSEQ))
 %%SEQUENCE)
 ((NULL %%SUBSEQ)
 ,SLICE)
 (SETQ %%SEQUENCE (CAR %%SUBSEQ))
 [RPLACA %%SUBSLICE (SEQ-DISPATCH %%SEQUENCE (PROG1 (CAR %%SEQUENCE)
 (RPLACA %%SUBSEQ (CDR %%SEQUENCE)))
 (CL:AREF %%SEQUENCE ,INDEX))])
```

(CL:DEFUN **%%MAP-FOR-EFFECT** (FUNCTION SEQUENCE &REST MORE-SEQUENCES)

```
(CL:IF (NULL MORE-SEQUENCES)
 (%MAP-FOR-EFFECT-SINGLE FUNCTION SEQUENCE)
 (%MAP-FOR-EFFECT-MULTIPLE FUNCTION (CONS SEQUENCE MORE-SEQUENCES))))
```

(CL:DEFUN **%%MAP-FOR-EFFECT-MULTIPLE** (FUNCTION SEQUENCES)

```
[LET [(MIN-LENGTH (%MIN-SEQUENCE-LENGTH SEQUENCES))
 (ELT-SLICE (CL:MAKE-LIST (CL:LENGTH SEQUENCES)
 (CL:DOTIMES (I MIN-LENGTH)
 (CL:APPLY FUNCTION (%FILL-SLICE I ELT-SLICE SEQUENCES)))))]
```

(CL:DEFUN **%%MAP-FOR-EFFECT-SINGLE** (FUNCTION SEQUENCE)

```
[SEQ-DISPATCH SEQUENCE (CL:DOLIST (ELT SEQUENCE)
 (CL:FUNCALL FUNCTION ELT))
 (CL:DOTIMES (I (VECTOR-LENGTH SEQUENCE))
 (CL:FUNCALL FUNCTION (CL:AREF SEQUENCE I)))]
```

(CL:DEFUN **%%MAP-FOR-RESULT-MULTIPLE** (RESULT-TYPE FUNCTION SEQUENCES)

```
[LET* ((MIN-LENGTH (%MIN-SEQUENCE-LENGTH SEQUENCES))
 (ELT-SLICE (CL:MAKE-LIST (CL:LENGTH SEQUENCES))
 (RESULT (MAKE-SEQUENCE-OF-TYPE RESULT-TYPE MIN-LENGTH)))
 (SEQ-DISPATCH RESULT (CL:DO ((SUBRESULT RESULT (CDR SUBRESULT))
 (INDEX 0 (CL:1+ INDEX)))
 ((EQL INDEX MIN-LENGTH)
 RESULT)
 (RPLACA SUBRESULT (CL:APPLY FUNCTION (%FILL-SLICE INDEX ELT-SLICE SEQUENCES)
 ))
 (CL:DO ((INDEX 0 (CL:1+ INDEX))
```

```

        ((EQL INDEX MIN-LENGTH)
         RESULT)
      (CL:SETF (CL:AREF RESULT INDEX)
              (CL:APPLY FUNCTION (%%FILL-SLICE INDEX ELT-SLICE SEQUENCES))))]]

```

```

(CL:DEFUN %%MAP-FOR-RESULT-SINGLE (RESULT-TYPE FUNCTION SEQUENCE)
  (LET* ((LENGTH (CL:LENGTH SEQUENCE))
         (RESULT (MAKE-SEQUENCE-OF-TYPE RESULT-TYPE LENGTH)))
    [SEQ-DISPATCH SEQUENCE [SEQ-DISPATCH RESULT (CL:DO ((SUBSEQ SEQUENCE (CDR SUBSEQ))
                                                         (SUBRESULT RESULT (CDR SUBRESULT)))
                                                         (NULL SUBSEQ))
                        (RPLACA SUBRESULT (CL:FUNCALL FUNCTION (CAR SUBSEQ))))]
      (CL:DO ((SUBSEQ SEQUENCE (CDR SUBSEQ))
             (INDEX 0 (CL:1+ INDEX)))
            ((NULL SUBSEQ))
            (CL:SETF (CL:AREF RESULT INDEX)
                    (CL:FUNCALL FUNCTION (CAR SUBSEQ))))]
    (SEQ-DISPATCH RESULT (CL:DO ((INDEX 0 (CL:1+ INDEX))
                                (SUBRESULT RESULT (CDR SUBRESULT)))
                                (EQL INDEX LENGTH))
                        (RPLACA SUBRESULT (CL:FUNCALL FUNCTION (CL:AREF SEQUENCE INDEX))))
      (CL:DO ((INDEX 0 (CL:1+ INDEX))
            ((EQL INDEX LENGTH))
            (CL:SETF (CL:AREF RESULT INDEX)
                    (CL:FUNCALL FUNCTION (CL:AREF SEQUENCE INDEX))))]
    RESULT))

```

```

(DEFMACRO %%MIN-SEQUENCE-LENGTH (SEQUENCES)
  `(CL:DO ([MIN-LENGTH (CL:LENGTH (CAR ,SEQUENCES)
                                (SUBSEQ (CDR ,SEQUENCES)
                                         (CDR SUBSEQ))
                                NEXT-LENGTH)
          ((NULL SUBSEQ))
          MIN-LENGTH)
    (SETQ NEXT-LENGTH (CL:LENGTH (CAR SUBSEQ)))
    (CL:IF (< NEXT-LENGTH MIN-LENGTH)
           (SETQ MIN-LENGTH NEXT-LENGTH)))

```

```

(CL:DEFUN CL:MAP (RESULT-TYPE FUNCTION SEQUENCE &REST MORE-SEQUENCES)
  "FUNCTION must take as many arguments as there are sequences provided. The result is a sequence such that
  element i is the result of applying FUNCTION to element i of each of the argument sequences."
  (CL:IF (NULL RESULT-TYPE)
         (CL:IF (NULL MORE-SEQUENCES)
                (%%MAP-FOR-EFFECT-SINGLE FUNCTION SEQUENCE)
                (%%MAP-FOR-EFFECT-MULTIPLE FUNCTION (CONS SEQUENCE MORE-SEQUENCES)))
         (CL:IF (NULL MORE-SEQUENCES)
                (%%MAP-FOR-RESULT-SINGLE RESULT-TYPE FUNCTION SEQUENCE)
                (%%MAP-FOR-RESULT-MULTIPLE RESULT-TYPE FUNCTION (CONS SEQUENCE MORE-SEQUENCES))))

```

:: For compatibility with old optimizers

```

(CL:DEFUN %%MAP-SINGLE-FOR-EFFECT (FUNCTION SEQUENCE)
  (%%MAP-FOR-EFFECT-SINGLE FUNCTION SEQUENCE))

```

```

(CL:DEFUN %%MAP-SINGLE-TO-LIST (FUNCTION SEQUENCE)
  (%%MAP-FOR-RESULT-SINGLE 'LIST FUNCTION SEQUENCE))

```

```

(CL:DEFUN %%MAP-SINGLE-TO-SIMPLE (RESULT-TYPE FUNCTION SEQUENCE)
  (%%MAP-FOR-RESULT-SINGLE RESULT-TYPE FUNCTION SEQUENCE))

```

```

(CL:DEFUN %%MAP-TO-LIST (FUNCTION SEQUENCE &REST MORE-SEQUENCES)
  (CL:IF (NULL MORE-SEQUENCES)
         (%%MAP-FOR-RESULT-SINGLE 'LIST FUNCTION SEQUENCE)
         (%%MAP-FOR-RESULT-MULTIPLE 'LIST FUNCTION (CONS SEQUENCE MORE-SEQUENCES))))

```

```

(CL:DEFUN %%MAP-TO-SIMPLE (RESULT-TYPE FUNCTION SEQUENCE &REST MORE-SEQUENCES)
  (CL:IF (NULL MORE-SEQUENCES)
         (%%MAP-FOR-RESULT-SINGLE RESULT-TYPE FUNCTION SEQUENCE)
         (%%MAP-FOR-RESULT-MULTIPLE RESULT-TYPE FUNCTION (CONS SEQUENCE MORE-SEQUENCES))))

```

```

(DEFOPTIMIZER CL:MAP (RESULT-TYPE FUNCTION FIRST-SEQUENCE &REST MORE-SEQUENCES)
  (CL:IF (AND (NULL MORE-SEQUENCES)
              (CL:CONSTANTP RESULT-TYPE))
        (CL:IF (NULL (EVAL RESULT-TYPE))
                (%%MAP-FOR-EFFECT-SINGLE ,FUNCTION ,FIRST-SEQUENCE)
                (%%MAP-FOR-RESULT-SINGLE ,RESULT-TYPE ,FUNCTION ,FIRST-SEQUENCE))
        'COMPILER:PASS))

```

```
(CL:DEFUN %%SOME-MULTIPLE (PREDICATE SEQUENCES)
  [LET [(MIN-LENGTH (%%MIN-SEQUENCE-LENGTH SEQUENCES))
        (ELT-SLICE (CL:MAKE-LIST (CL:LENGTH SEQUENCES)
                                  (CL:DO ((INDEX 0 (CL:1+ INDEX))
                                           PREDICATE-RESULT)
                                           ((EQL INDEX MIN-LENGTH))
                                           (SETQ PREDICATE-RESULT (CL:APPLY PREDICATE (%%FILL-SLICE INDEX ELT-SLICE SEQUENCES)))
                                           (CL:IF PREDICATE-RESULT (RETURN PREDICATE-RESULT)))))]])
```

```
(CL:DEFUN %%SOME-SINGLE (PREDICATE SEQUENCE)
  [LET ((LENGTH (CL:LENGTH SEQUENCE)))
    (SEQ-DISPATCH SEQUENCE (FORWARD-LIST-LOOP SEQUENCE 0 LENGTH (INDEX CURRENT PREDICATE-RESULT)
                                          NIL
                                          (SETQ PREDICATE-RESULT (CL:FUNCALL PREDICATE CURRENT))
                                          (CL:IF PREDICATE-RESULT (RETURN PREDICATE-RESULT)))
      (FORWARD-VECTOR-LOOP SEQUENCE 0 LENGTH (INDEX CURRENT PREDICATE-RESULT)
        NIL
        (SETQ PREDICATE-RESULT (CL:FUNCALL PREDICATE CURRENT))
        (CL:IF PREDICATE-RESULT (RETURN PREDICATE-RESULT)))]])
```

```
(CL:DEFUN %%EVERY-MULTIPLE (PREDICATE SEQUENCES)
  [LET [(MIN-LENGTH (%%MIN-SEQUENCE-LENGTH SEQUENCES))
        (ELT-SLICE (CL:MAKE-LIST (CL:LENGTH SEQUENCES)
                                  (CL:DOTIMES (INDEX MIN-LENGTH T)
                                                (CL:IF (NULL (CL:APPLY PREDICATE (%%FILL-SLICE INDEX ELT-SLICE SEQUENCES)))
                                                         (RETURN NIL)))))]])
```

```
(CL:DEFUN %%EVERY-SINGLE (PREDICATE FIRST-SEQUENCE)
  [SEQ-DISPATCH FIRST-SEQUENCE (CL:DOLIST (ELT FIRST-SEQUENCE T)
                                           (CL:IF (NULL (CL:FUNCALL PREDICATE ELT))
                                                  (RETURN NIL)))
    (CL:DOTIMES (INDEX (VECTOR-LENGTH FIRST-SEQUENCE)
                      T)
      (CL:IF (NULL (CL:FUNCALL PREDICATE (CL:AREF FIRST-SEQUENCE INDEX)))
              (RETURN NIL)))))]])
```

```
(CL:DEFUN %%NOTANY-MULTIPLE (PREDICATE SEQUENCES)
  [LET [(MIN-LENGTH (%%MIN-SEQUENCE-LENGTH SEQUENCES))
        (ELT-SLICE (CL:MAKE-LIST (CL:LENGTH SEQUENCES)
                                  (CL:DOTIMES (INDEX MIN-LENGTH T)
                                                (CL:IF (CL:APPLY PREDICATE (%%FILL-SLICE INDEX ELT-SLICE SEQUENCES))
                                                         (RETURN NIL)))))]])
```

```
(CL:DEFUN %%NOTANY-SINGLE (PREDICATE FIRST-SEQUENCE)
  [SEQ-DISPATCH FIRST-SEQUENCE (CL:DOLIST (ELT FIRST-SEQUENCE T)
                                           (CL:IF (CL:FUNCALL PREDICATE ELT)
                                                  (RETURN NIL)))
    (CL:DOTIMES (I (VECTOR-LENGTH FIRST-SEQUENCE)
                  T)
      (CL:IF (CL:FUNCALL PREDICATE (CL:AREF FIRST-SEQUENCE I))
              (RETURN NIL)))))]])
```

```
(CL:DEFUN %%NOTEVERY-MULTIPLE (PREDICATE SEQUENCES)
  [LET [(MIN-LENGTH (%%MIN-SEQUENCE-LENGTH SEQUENCES))
        (ELT-SLICE (CL:MAKE-LIST (CL:LENGTH SEQUENCES)
                                  (CL:DOTIMES (INDEX MIN-LENGTH)
                                                (CL:IF (NULL (CL:APPLY PREDICATE (%%FILL-SLICE INDEX ELT-SLICE SEQUENCES)))
                                                         (RETURN T)))))]])
```

```
(CL:DEFUN %%NOTEVERY-SINGLE (PREDICATE FIRST-SEQUENCE)
  [SEQ-DISPATCH FIRST-SEQUENCE (CL:DOLIST (ELT FIRST-SEQUENCE)
                                           (CL:IF (NULL (CL:FUNCALL PREDICATE ELT))
                                                  (RETURN T)))
    (CL:DOTIMES (I (VECTOR-LENGTH FIRST-SEQUENCE)
                  (CL:IF (NULL (CL:FUNCALL PREDICATE (CL:AREF FIRST-SEQUENCE I))
                          (RETURN T)))))]])
```

```
(CL:DEFUN CL:SOME (PREDICATE FIRST-SEQUENCE &REST MORE-SEQUENCES)
  "PREDICATE is applied to the elements with index 0 of the sequences, then possibly to those with index 1, and so on. SOME returns the first non-() value encountered, or () if the end of a sequence is reached."
  (CL:IF (NULL MORE-SEQUENCES)
    (%%SOME-SINGLE PREDICATE FIRST-SEQUENCE)
    (%%SOME-MULTIPLE PREDICATE (CONS FIRST-SEQUENCE MORE-SEQUENCES))))
```

```
(CL:DEFUN CL:EVERY (PREDICATE FIRST-SEQUENCE &REST MORE-SEQUENCES)
  "PREDICATE is applied to the elements with index 0 of the sequences, then possibly to those with index 1, and so on. EVERY returns () as soon as any invocation of PREDICATE returns (), or T if every invocation is
```

```

non-() ."
(CL:IF (NULL MORE-SEQUENCES)
  (%%EVERY-SINGLE PREDICATE FIRST-SEQUENCE)
  (%%EVERY-MULTIPLE PREDICATE (CONS FIRST-SEQUENCE MORE-SEQUENCES))))

```

```

(CL:DEFUN CL:NOTANY (PREDICATE FIRST-SEQUENCE &REST MORE-SEQUENCES)
  "PREDICATE is applied to the elements with index 0 of the sequences, then possibly to those with index 1, and
  so on. NOTANY returns () as soon as any invocation of PREDICATE returns a non-() value, or T if the end of a
  sequence is reached."
  (CL:IF (NULL MORE-SEQUENCES)
    (%%NOTANY-SINGLE PREDICATE FIRST-SEQUENCE)
    (%%NOTANY-MULTIPLE PREDICATE (CONS FIRST-SEQUENCE MORE-SEQUENCES))))

```

```

(CL:DEFUN CL:NOTEVERY (PREDICATE FIRST-SEQUENCE &REST MORE-SEQUENCES)
  "PREDICATE is applied to the elements with index 0 of the sequences, then possibly to those with index 1, and
  so on. NOTEVERY returns T as soon as any invocation of PREDICATE returns (), or () if every invocation is
  non-() ."
  (CL:IF (NULL MORE-SEQUENCES)
    (%%NOTEVERY-SINGLE PREDICATE FIRST-SEQUENCE)
    (%%NOTEVERY-MULTIPLE PREDICATE (CONS FIRST-SEQUENCE MORE-SEQUENCES))))

```

:: For compatibility with old optimizers

```

(MOVD '%%SOME-SINGLE '%%SINGLE-SOME)
(MOVD '%%EVERY-SINGLE '%%SINGLE-EVERY)
(MOVD '%%NOTEVERY-SINGLE '%%SINGLE-NOTEVERY)
(MOVD '%%NOTANY-SINGLE '%%SINGLE-NOTANY)

```

```

(DEFOPTIMIZER CL:SOME (PREDICATE SEQUENCE &REST MORE-SEQUENCES)
  (COND
    [(NULL MORE-SEQUENCES)
     \ (%%SOME-SINGLE ,PREDICATE ,SEQUENCE)
     (T 'COMPILER:PASS)])

```

```

(DEFOPTIMIZER CL:EVERY (PREDICATE SEQUENCE &REST MORE-SEQUENCES)
  (COND
    [(NULL MORE-SEQUENCES)
     \ (%%EVERY-SINGLE ,PREDICATE ,SEQUENCE)
     (T 'COMPILER:PASS)])

```

```

(DEFOPTIMIZER CL:NOTANY (PREDICATE SEQUENCE &REST MORE-SEQUENCES)
  (COND
    [(NULL MORE-SEQUENCES)
     \ (%%NOTANY-SINGLE ,PREDICATE ,SEQUENCE)
     (T 'COMPILER:PASS)])

```

```

(DEFOPTIMIZER CL:NOTEVERY (PREDICATE SEQUENCE &REST MORE-SEQUENCES)
  (COND
    [(NULL MORE-SEQUENCES)
     \ (%%NOTEVERY-SINGLE ,PREDICATE ,SEQUENCE)
     (T 'COMPILER:PASS)])

```

```

(CL:DEFUN REDUCE-FROM-END (FUNCTION SEQUENCE START END INITIAL-VALUE)
  "Backward reduction"
  [SEQ-DISPATCH SEQUENCE (BACKWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT (ACCUMULATOR INITIAL-VALUE)))
    ACCUMULATOR
    (SETQ ACCUMULATOR (CL:FUNCALL FUNCTION CURRENT ACCUMULATOR))]
  (BACKWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (ACCUMULATOR INITIAL-VALUE))
    ACCUMULATOR
    (SETQ ACCUMULATOR (CL:FUNCALL FUNCTION CURRENT ACCUMULATOR]))

```

```

(CL:DEFUN REDUCE-FROM-START (FUNCTION SEQUENCE START END INITIAL-VALUE)
  [SEQ-DISPATCH SEQUENCE (FORWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT (ACCUMULATOR INITIAL-VALUE)))
    ACCUMULATOR
    (SETQ ACCUMULATOR (CL:FUNCALL FUNCTION ACCUMULATOR CURRENT))]
  (FORWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (ACCUMULATOR INITIAL-VALUE))
    ACCUMULATOR
    (SETQ ACCUMULATOR (CL:FUNCALL FUNCTION ACCUMULATOR CURRENT]))

```

```

(CL:DEFUN CL:REDUCE (FUNCTION SEQUENCE &KEY (START 0)
  END FROM-END (INITIAL-VALUE NIL INITIAL-VALUE-P))
  [LET ((LENGTH (CL:LENGTH SEQUENCE)))
    (CL:IF (NULL END)
      (SETQ END LENGTH))
    (CHECK-SUBSEQ SEQUENCE START END LENGTH)

```

```
(CL:IF INITIAL-VALUE-P
  (CL:IF FROM-END
    (REDUCE-FROM-END FUNCTION SEQUENCE START END INITIAL-VALUE)
    (REDUCE-FROM-START FUNCTION SEQUENCE START END INITIAL-VALUE))
  (CASE (- END START)
    (0 (CL:FUNCALL FUNCTION))
    (1 (CL:ELT SEQUENCE START))
    (T (CL:IF FROM-END
      (REDUCE-FROM-END FUNCTION SEQUENCE START (CL:1- END)
        (CL:ELT SEQUENCE (CL:1- END)))
      (REDUCE-FROM-START FUNCTION SEQUENCE (CL:1+ START)
        END
        (CL:ELT SEQUENCE START))))))])

(PUTPROPS CMLSEQMAPPERS FILETYPE CL:COMPILE-FILE)

(DECLARE%: DONTEVAL@LOAD DONTCOPY DOEVAL@COMPILE

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)

(PUTPROPS CMLSEQMAPPERS COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990))
```

FUNCTION INDEX

%EVERY-MULTIPLE	3	%%MAP-SINGLE-TO-SIMPLE	2	CL:EVERY	3
%%EVERY-SINGLE	3	%%MAP-TO-LIST	2	CL:MAP	2
%%MAP-FOR-EFFECT	1	%%MAP-TO-SIMPLE	2	CL:NOTANY	4
%%MAP-FOR-EFFECT-MULTIPLE	1	%%NOTANY-MULTIPLE	3	CL:NOTEVERY	4
%%MAP-FOR-EFFECT-SINGLE	1	%%NOTANY-SINGLE	3	CL:REDUCE	4
%%MAP-FOR-RESULT-MULTIPLE	1	%%NOTEVERY-MULTIPLE	3	REDUCE-FROM-END	4
%%MAP-FOR-RESULT-SINGLE	2	%%NOTEVERY-SINGLE	3	REDUCE-FROM-START	4
%%MAP-SINGLE-FOR-EFFECT	2	%%SOME-MULTIPLE	3	CL:SOME	3
%%MAP-SINGLE-TO-LIST	2	%%SOME-SINGLE	3		

OPTIMIZER INDEX

CL:EVERY	4	CL:MAP	2	CL:NOTANY	4	CL:NOTEVERY	4	CL:SOME	4
----------------	---	--------------	---	-----------------	---	-------------------	---	---------------	---

MACRO INDEX

%%FILL-SLICE	1	%%MIN-SEQUENCE-LENGTH	2
--------------------	---	-----------------------------	---

PROPERTY INDEX

CMLSEQMAPPERS	5
---------------------	---
