File created: 8-Jun-90 14:57:22 **{PELE:MV:ENVOS}<LISPCORE>SOURCES>CMLPARSE.;3**

changes to: (FUNCTIONS ANALYZE)

previous date: 16-May-90 14:14:58 {PELE:MV:ENVOS}<LISPCORE>SOURCES>CMLPARSE.;2

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

;;


```
(RPAQQ CMLPARSECOMS
       (;; Parsing bodies and argument lists
        (VARIABLES %%ARG-COUNT %%MIN-ARGS %%UNBOUNDED-ARG-COUNT %%LET-LIST %%KEYWORD-TESTS %%ENV-ARG-USED
               %%CTX-ARG-USED %%ENV-ARG-NAME %%CTX-ARG-NAME)
        (VARIABLES *DEFAULT-DEFAULT* *KEY-FINDER*)
        (FUNCTIONS PARSE-BODY)
        (FUNCTIONS PARSE-DEFMACRO ANALYZE ANALYZE-AUX ANALYZE-KEY ANALYZE-PARAMETER CHECK-PARAMETER-NAME
               PUSH-KEYWORD-BINDING ANALYZE-REST RECURSIVELY-ANALYZE DEFMACRO-ARG-TEST)
        ;; Testing the argument-list parsing
        (VARIABLES ANALYZE-TESTS)
        ;; Runtime support functions
        (FUNCTIONS KEYWORD-TEST FIND-KEYWORD)
        ;; Arrange to use the correct compiler
        (PROP FILETYPE CMLPARSE)))

;; Parsing bodies and argument lists

(CL:DEFVAR %%ARG-COUNT NIL)

(CL:DEFVAR %%MIN-ARGS NIL)

(CL:DEFVAR %%UNBOUNDED-ARG-COUNT 0)

(CL:DEFVAR %%LET-LIST NIL)

(CL:DEFVAR %%KEYWORD-TESTS NIL)

(CL:DEFVAR %%ENV-ARG-USED NIL)

(CL:DEFVAR %%CTX-ARG-USED NIL)

(CL:DEFVAR %%ENV-ARG-NAME NIL)

(CL:DEFVAR %%CTX-ARG-NAME NIL)

(CL:DEFVAR *DEFAULT-DEFAULT* NIL)

(CL:DEFVAR *KEY-FINDER* NIL)

(CL:DEFUN PARSE-BODY (XCL::BODY XCL::ENVIRONMENT &OPTIONAL (XCL::DOC-STRING-ALLOWED T))
    ;; CDR down the list of forms in BODY, looking for declarations and documentation strings, until we hit either the end of the BODY or a form that is
    ;; neither of these.  We expand macros in our search for declarations and doc-strings, but only until we find a form we don't understand.
    ;;
    ;; Return three values:
    ;;   1) The remainder of the BODY, after declarations and doc-strings,
    ;;   2) A list of the declarations found,
    ;;   3) The first documentation string found, or NIL if none are present.
    (LET ((XCL::TAIL XCL::BODY)
          (XCL::DECLS NIL)
          (XCL::DOC NIL))
         (CL:LOOP (CL:WHEN (NULL XCL::TAIL)
```

```
                    (RETURN))
               [LET ((XCL::FORM (CAR XCL::TAIL)))
                    (COND
                       ((AND (CL:STRINGP XCL::FORM)
                             (CDR XCL::TAIL))                    ; Be careful about strings at the end of BODY... They aren't
                                                                 ; doc-strings!
                        (CL:IF (AND (NOT XCL::DOC)
                                    XCL::DOC-STRING-ALLOWED)
                               (CL:SETQ XCL::DOC XCL::FORM)))
                       ([OR (CL:ATOM XCL::FORM)
                            (NOT (CL:SYMBOLP (CAR XCL::FORM]
                        (RETURN))
                       ((EQ (CAR XCL::FORM)
                            'DECLARE)
                        (CL:PUSH XCL::FORM XCL::DECLS))
                       ((EQ (CAR XCL::FORM)
                            COMMENTFLG)                          ; Ignore Interlisp comments.
                        NIL)
                       ((CL:SPECIAL-FORM-P (CAR XCL::FORM))
                        (RETURN))
                       (T (LET [(XCL::RESULT (CONDITIONS:RESTART-CASE (CL:MACROEXPAND XCL::FORM XCL::ENVIRONMENT
                                                                      )
                                                                      (CONDITIONS:CONTINUE NIL :REPORT
                                                                             (CL:LAMBDA (STREAM)
                                                                                    (LET ((*PRINT-LEVEL* 3)
                                                                                          (*PRINT-LENGTH* 3))
                                                                                         (CL:FORMAT STREAM "Assume that ~S does
                                                                                                not expand into a declaration."
                                                                                                XCL::FORM)))
                                                                      XCL::FORM]
                                     (CL:IF (AND (CL:CONSP XCL::RESULT)
                                                 (EQ (CAR XCL::RESULT)
                                                     'DECLARE))
                                            (CL:PUSH XCL::RESULT XCL::DECLS)
                                            (RETURN))]
                    (CL:POP XCL::TAIL))
             (CL:VALUES XCL::TAIL (CL:REVERSE XCL::DECLS)
                    XCL::DOC)))

(CL:DEFUN PARSE-DEFMACRO [ARGUMENT-LIST WHOLE-EXPRESSION MACRO-BODY ERROR-LOCATION ENVIRONMENT &KEY
                                [PATH `(CDR ,WHOLE-EXPRESSION]
                                ((:ENVIRONMENT %%ENV-ARG-NAME))
                                ((:CONTEXT %%CTX-ARG-NAME))
                                (ERROR-STRING NIL)
                                (DOC-STRING-ALLOWED T)
                                ((:DEFAULT-DEFAULT *DEFAULT-DEFAULT*)
                                 NIL)
                                ((:KEY-FINDER *KEY-FINDER*)
                                 'FIND-KEYWORD)
                                (REMOVE-COMMENTS (AND (EQ (CAR ARGUMENT-LIST)
                                                          '&WHOLE)
                                                      (EQ (CADR ARGUMENT-LIST)
                                                          '%%ORIGINAL-DEFINITION]
   (DECLARE (CL:SPECIAL %%CTX-ARG-NAME %%ENV-ARG-NAME *KEY-FINDER* *DEFAULT-DEFAULT*))
   ;; "Parse-Defmacro provides a clean interface to ANALYZE for use by macros and macro-like forms that must parse some form according to a
   ;; defmacro-like argument list.
   ;;
   ;; -- ARGUMENT-LIST is the argument-list to be used for parsing.
   ;; -- WHOLE-EXPRESSION is the variable which is bound to the entire macro-call, or NIL if &whole is illegal.
   ;; -- MACRO-BODY is the code that will be executed in the scope of the argument-list.
   ;; -- ERROR-LOCATION is the name of the function being worked on, for use in error messages.
   ;; -- ENVIRONMENT is an environment in which PARSE-DEFMACRO may macroexpand the WHOLE-EXPRESSION, looking for declarations.
   ;; -- PATH is an access expression for getting to the object to be parsed, which defaults to the CDR of WHOLE.
   ;; -- :ENVIRONMENT is the place where the macroexpansion environment may be found. If not supplied, then no &environment arg is allowed.
   ;; -- :CONTEXT is the place where the macroexpansion compiler context may be found.  If not supplied, then no &context arg is allowed.
   ;; -- :ERROR-STRING is used as the first argument to ERROR if an incorrect number of arguments are supplied.  The additional arguments to
   ;; ERROR are ERRLOC and the number of arguments supplied.  If ERROR-STRING is not supplied, then no argument count error checking is done.
   ;; -- :DOC-STRING-ALLOWED indicates whether a doc-string should be parsed out of the body.
   ;; -- :DEFAULT-DEFAULT is the default value for unsupplied arguments, which defaults to NIL.
   ;; -- :KEY-FINDER the function used to do keyword lookup. It defaults to a function that does the right thing.  If you supply your own, it should take
   ;; two arguments, the keyword to be found and a list in which to find it, and return either a list of one element, the value of the given keyword, or NIL,
   ;; if the keyword is not present.
   ;; -- :REMOVE-COMMENTS should be non-NIL iff comments should be stripped from the macro-call before processing.  The default is set up as a
   ;; horrible hack to allow macros created by DEFDEFINER to get this feature.
   ;;
   ;; The first value returned is a LET* form which binds things and then evaluates the specified CODE.
   ;; The second value is a list of ignore declarations for the WHOLE and ENVIRONMENT vars, if appropriate.
```

```
;; The third value is the documentation string, if DOC-STRING-ALLOWED and one is present, and NIL otherwise.

;; The fourth and fifth values are the minimum and maximum number of arguments allowed, in case you care about that kind of thing. The fifth value
;; is NIL if there is no upper limit.

(CL:MULTIPLE-VALUE-BIND (BODY LOCAL-DECS DOC)
    (PARSE-BODY MACRO-BODY ENVIRONMENT DOC-STRING-ALLOWED)
  [LET ((%%ARG-COUNT 0)
        (%%MIN-ARGS 0)
        (%%UNBOUNDED-ARG-COUNT NIL)
        (%%LET-LIST NIL)
        (%%KEYWORD-TESTS NIL)
        (%%ENV-ARG-USED NIL)
        (%%CTX-ARG-USED NIL))
       (ANALYZE ARGUMENT-LIST (CL:IF REMOVE-COMMENTS
                                  `(REMOVE-COMMENTS ,PATH)
                                 PATH)
              ERROR-LOCATION WHOLE-EXPRESSION)
       (LET [(ARG-TEST (CL:IF ERROR-STRING (DEFMACRO-ARG-TEST PATH)))
             (BODY `(LET* ,(REVERSE %%LET-LIST)
                          ,@LOCAL-DECS
                          ,@%%KEYWORD-TESTS
                          ,@BODY]
            (CL:VALUES (CL:IF ARG-TEST
                              `(CL:IF ,ARG-TEST
                                   (CL:ERROR ,ERROR-STRING ',ERROR-LOCATION (CL:LENGTH ,PATH))
                                   ,BODY)
                               BODY)
                       `[,@(CL:UNLESS (OR ARG-TEST ARGUMENT-LIST)
                              `[(DECLARE (IGNORE ,WHOLE-EXPRESSION])
                         ,@(CL:WHEN (AND %%ENV-ARG-NAME (NOT %%ENV-ARG-USED))
                              `[(DECLARE (IGNORE ,%%ENV-ARG-NAME])
                         ,@(CL:WHEN (AND %%CTX-ARG-NAME (NOT %%CTX-ARG-USED))
                              `[(DECLARE (IGNORE ,%%CTX-ARG-NAME])]
                       DOC %%MIN-ARGS (CL:IF %%UNBOUNDED-ARG-COUNT
                                             NIL
                                             %%ARG-COUNT)]))


(CL:DEFUN ANALYZE (ARGLIST PATH ERRLOC WHOLE)

;; ANALYZE is implemented as a finite-state machine that steps through the legal parts of an arglist in order: required, optional, rest, key, and aux.
;; The results are accumulated in a set of special variables: %let-list, %arg-count, %min-args, %unbounded-arg-count, %keyword-tests,
;; %ctx-arg-used and %env-arg-used.  It reads the special variables %env-arg-name and %ctx-arg-name.

  (CL:ASSERT (CL:LISTP ARGLIST)
         NIL "The argument list %"~S%" was not a list." ARGLIST)
  (CL:UNLESS (OR (CL:ATOM PATH)
                 (NULL ARGLIST))                                    ; Eliminate a common subexpression
                                                                    ; (OR ...(NULL ARGLIST))  is added for solution of
                                                                    ; AR#7337(Edited by TT : 8-June-90)

      (LET ((NEW-PATH (GENSYM)))
           (CL:PUSH `(,NEW-PATH ,PATH)
                 %%LET-LIST)
           (CL:SETQ PATH NEW-PATH)))
  (CL:DO ((ARGS ARGLIST (CDR ARGS))
          (OPTIONALP NIL)
          A)
         ((CL:ATOM ARGS)
          (CL:UNLESS (NULL ARGS)                                    ; If the variable-list is dotted, treat it as a &rest argument and
                                                                    ; return.
              (CL:SETQ %%UNBOUNDED-ARG-COUNT T)
              (CL:PUSH `(,ARGS ,PATH)
                    %%LET-LIST)))
      (CL:SETQ A (CAR ARGS))
      (CASE A
          ((&WHOLE) (COND
                       ((AND WHOLE (CL:CONSP (CDR ARGS)))
                        (ANALYZE-PARAMETER (CADR ARGS)
                               WHOLE ERRLOC)
                        (SETQ %%UNBOUNDED-ARG-COUNT T)
                        (SETQ ARGS (CDR ARGS))                      ; Only one CDR here; the other one is done by the DO loop,
                                                                    ; above.

                        )
                       (T (CL:ERROR "Illegal or ill-formed &whole arg in ~S." ERRLOC))))
          ((&ENVIRONMENT) (COND
                             ((AND %%ENV-ARG-NAME (CL:CONSP (CDR ARGS))
                                   (CL:SYMBOLP (CADR ARGS)))
                              (CHECK-PARAMETER-NAME (CADR ARGS)
                                    ERRLOC)
                              (CL:PUSH `(,(CADR ARGS)
                                         ,%%ENV-ARG-NAME)
                                    %%LET-LIST)
                              (CL:SETQ %%ENV-ARG-USED T)
                              (CL:SETQ ARGS (CDR ARGS)))
                             (T (CL:ERROR "Illegal or ill-formed &environment arg in ~S." ERRLOC))))
          ((&CONTEXT) (COND
                         ((AND %%CTX-ARG-NAME (CL:CONSP (CDR ARGS))
```

```
                              (CL:SYMBOLP (CADR ARGS)))
                             (CHECK-PARAMETER-NAME (CADR ARGS)
                                    ERRLOC)
                             (CL:PUSH `(,(CADR ARGS)
                                        ,%%CTX-ARG-NAME)
                                    %%LET-LIST)
                             (CL:SETQ %%CTX-ARG-USED T)
                             (CL:SETQ ARGS (CDR ARGS)))
                            (T (CL:ERROR "Illegal or ill-formed &context arg in ~S." ERRLOC))))
          ((&OPTIONAL)
             (AND OPTIONALP (CL:CERROR "Ignore it." "Redundant &optional flag in varlist of ~S." ERRLOC))
             (CL:SETQ OPTIONALP T))
          ((&REST &BODY) (RETURN (ANALYZE-REST (CAR ARGS)
                                       (CDR ARGS)
                                      PATH ERRLOC WHOLE)))
          ((&KEY) (LET ((KEYWORD-ARGS-VAR (GENSYM)))
                       (CL:SETQ %%UNBOUNDED-ARG-COUNT T)
                       (CL:PUSH `(,KEYWORD-ARGS-VAR ,PATH)
                             %%LET-LIST)
                       (RETURN (ANALYZE-KEY (CDR ARGS)
                                    KEYWORD-ARGS-VAR ERRLOC))))
          ((&ALLOW-OTHER-KEYS) (CL:CERROR "Ignore it." "Stray &ALLOW-OTHER-KEYS in arglist of ~S." ERRLOC))
          ((&AUX) (RETURN (ANALYZE-AUX (CDR ARGS)
                                ERRLOC)))

          ;; It's actually a parameter!

          (CL:OTHERWISE
             (COND
                ;; It's an optional argument.

                [OPTIONALP (CL:SETQ %%ARG-COUNT (CL:1+ %%ARG-COUNT))
                       (COND
                          ;; The normal case, a simple variable.

                          ((CL:SYMBOLP A)
                           (CHECK-PARAMETER-NAME A ERRLOC)
                           (CL:PUSH `[,A (COND
                                             (,PATH (CAR ,PATH))
                                             (T ,*DEFAULT-DEFAULT*]
                                  %%LET-LIST))

                          ;; A buggy case.

                          ((CL:ATOM A)
                           (CL:CERROR "Ignore this item." "Non-symbol variable name in ~S." ERRLOC))

                          ;; The defaulting case:  (var [default [svar]])

                          (T (ANALYZE-PARAMETER (CAR A)
                                    `[COND
                                        (,PATH (CAR ,PATH))
                                        (T ,(COND
                                               ((CDR A)         ; Was a default value specified?
                                                (CADR A))
                                               (T *DEFAULT-DEFAULT*]
                                      ERRLOC)
                             (CL:WHEN (NOT (NULL (CDDR A)))
                                 (CHECK-PARAMETER-NAME (CADDR A)
                                        ERRLOC)
                                 (CL:PUSH `[,(CADDR A)
                                             (NOT (NULL ,PATH]
                                        %%LET-LIST))]

                ;; It's a required argument.

                (T (CL:SETQ %%MIN-ARGS (CL:1+ %%MIN-ARGS))
                   (CL:SETQ %%ARG-COUNT (CL:1+ %%ARG-COUNT))
                   (ANALYZE-PARAMETER A `(CAR ,PATH)
                          ERRLOC)))

             ;; After each real parameter, we need to advance PATH by CDRing.  In many cases, though, we can eliminate a common
             ;; subexpression.

             (CL:IF (OR (CL:ATOM (CDR ARGS))
                        (CL:ATOM (CDDR ARGS)))
                 [CL:SETQ PATH `(CDR ,PATH]
                 (LET ((NEW-PATH (GENSYM)))
                      (CL:PUSH `(,NEW-PATH (CDR ,PATH))
                             %%LET-LIST)
                      (CL:SETQ PATH NEW-PATH)))))))
```

```
(CL:DEFUN ANALYZE-AUX (ARGLIST ERRLOC)
   ;; Analyze stuff following &aux.
   (CL:DO ((ARGS ARGLIST (CDR ARGS)))
          ((NULL ARGS))
       (COND
          ((CL:ATOM ARGS)
           (CL:CERROR "Ignore the illegal terminator." "Dotted arglist after &AUX in ~S." ERRLOC)
           (RETURN NIL))
```

```
          ((CL:SYMBOLP (CAR ARGS))
           (CHECK-PARAMETER-NAME (CAR ARGS)
                  ERRLOC)
           (CL:PUSH `(,(CAR ARGS)
                      NIL)
                  %%LET-LIST))
          ((CL:ATOM (CAR ARGS))
           (CL:ERROR "Non-symbolic &AUX parameter %"~S%" in arglist of ~S." (CAR ARGS)
                  ERRLOC))
          ((CL:SYMBOLP (CAAR ARGS))
           (CHECK-PARAMETER-NAME (CAAR ARGS)
                  ERRLOC)
           (CL:PUSH `(,(CAAR ARGS)
                      ,(CADAR ARGS))
                  %%LET-LIST))
          (T (CL:ERROR "Non-symbolic &AUX parameter %"~S%" in arglist of ~S." (CAAR ARGS)
                    ERRLOC)))))
```

(CL:DEFUN **ANALYZE-KEY** (ARGLIST RESTVAR ERRLOC)

;; Handle analysis of keywords, perhaps with destructuring over the keyword variable. Assumes the remainder of the calling form has already been ;; bound to the variable passed in as RESTVAR.

```
(LET ((TEMP (GENSYM))
      (CHECK-KEYWORDS T)
      (KEYWORDS-SEEN NIL))
     (CL:PUSH TEMP %%LET-LIST)                              ; TEMP will be used for each keyword as a temporary piece of
                                                            ; storage; see PUSH-KEYWORD-BINDING.
     (CL:DO ((ARGS ARGLIST (CDR ARGS))
             A K SP-VAR TEMP1)
            ((CL:ATOM ARGS)
             (CL:IF (NULL ARGS)
                 NIL
                 (CL:CERROR "Ignore the illegal terminator." "Dotted arglist after &key in ~S." ERRLOC)))
        (SETQ A (CAR ARGS))
        (COND
           ((EQ A '&ALLOW-OTHER-KEYS)
            (SETQ CHECK-KEYWORDS NIL))
           ((EQ A '&AUX)
            (RETURN (ANALYZE-AUX (CDR ARGS)
                           ERRLOC)))
           [(EQ A '&ENVIRONMENT)
            (COND
               ((AND %%ENV-ARG-NAME (CL:CONSP (CDR ARGS))
                     (CL:SYMBOLP (CADR ARGS)))
                (CHECK-PARAMETER-NAME (CADR ARGS)
                       ERRLOC)
                (CL:PUSH `(,(CADR ARGS)
                           ,%%ENV-ARG-NAME)
                       %%LET-LIST)
                (CL:SETQ %%ENV-ARG-USED T)
                (CL:SETQ ARGS (CDR ARGS)))
               (T (CL:ERROR "Illegal or ill-formed &environment arg in ~S." ERRLOC]
           [(EQ A '&CONTEXT)
            (COND
               ((AND %%CTX-ARG-NAME (CL:CONSP (CDR ARGS))
                     (CL:SYMBOLP (CADR ARGS)))
                (CHECK-PARAMETER-NAME (CADR ARGS)
                       ERRLOC)
                (CL:PUSH `(,(CADR ARGS)
                           ,%%CTX-ARG-NAME)
                       %%LET-LIST)
                (CL:SETQ %%CTX-ARG-USED T)
                (CL:SETQ ARGS (CDR ARGS)))
               (T (CL:ERROR "Illegal or ill-formed &context arg in ~S." ERRLOC]
           ((CL:SYMBOLP A)                                  ; Just a top-level variable. Make matching keyword.
            (SETQ K (MAKE-KEYWORD A))
            (PUSH-KEYWORD-BINDING A K NIL NIL RESTVAR TEMP ERRLOC)
            (CL:PUSH K KEYWORDS-SEEN))
           ((CL:ATOM A)                                     ; Filter out error that might choke defmacro.
            (CL:CERROR "Ignore this item." "~S -- non-symbol variable name in arglist of ~S." A ERRLOC))
           ((CL:SYMBOLP (CAR A))                            ; Deal with the common case:
                                                            ; (var [init [svar]])
            (SETQ K (MAKE-KEYWORD (CAR A)))
            (SETQ SP-VAR (CADDR A))
            (PUSH-KEYWORD-BINDING (CAR A)
                   K
                   (CADR A)
                   SP-VAR RESTVAR TEMP ERRLOC)
            (CL:PUSH K KEYWORDS-SEEN))
           ((OR (CL:ATOM (CAR A))
                (NOT (CL:KEYWORDP (CAAR A)))
                (CL:ATOM (CDAR A)))                         ; Filter out more error cases that might kill defmacro.
            (CL:CERROR "Ignore this item." "~S -- ill-formed keyword arg in ~S." (CAR A)
                   ERRLOC))
           ((CL:SYMBOLP (CADR (CAR A)))                     ; Next case is
                                                            ; ((:key var) [init [supplied-p]]).
```

```
               (SETQ K (CAAR A))
               (CL:UNLESS (CL:KEYWORDP K)
                      (CL:ERROR "%"~S%" should be a keyword, in arglist of ~S." K ERRLOC))
               (SETQ SP-VAR (CADDR A))
               (PUSH-KEYWORD-BINDING (CADR (CAR A))
                      K
                      (CADR A)
                      SP-VAR RESTVAR TEMP ERRLOC)
               (CL:PUSH K KEYWORDS-SEEN))
              (T                                                    ; Same case, but must destructure the 'variable'.
                 (SETQ K (CAAR A))
                 (CL:UNLESS (CL:KEYWORDP K)
                        (CL:ERROR "%"~S%" should be a keyword, in arglist of ~S." K ERRLOC))
                 (SETQ TEMP1 (GENSYM))
                 (SETQ SP-VAR (CADDR A))
                 (PUSH-KEYWORD-BINDING TEMP1 K (CADR A)
                        SP-VAR RESTVAR TEMP ERRLOC)
                 (CL:PUSH K KEYWORDS-SEEN)
                 (RECURSIVELY-ANALYZE (CADAR A)
                        TEMP1 ERRLOC NIL))))
        (CL:WHEN CHECK-KEYWORDS
            (CL:PUSH `(KEYWORD-TEST ,RESTVAR ',KEYWORDS-SEEN)
                  %%KEYWORD-TESTS))))
```

(CL:DEFUN **ANALYZE-PARAMETER** (PARAM PATH ERRLOC)

;;; We are given a single parameter and the path for getting to its value. The parameter may ask us to destructure the value. Arrange for the parameter ;;; to get its value.

```
   [COND
      ((CL:SYMBOLP PARAM)                                           ; The simple, normal case.
       (CHECK-PARAMETER-NAME PARAM ERRLOC)
       (CL:PUSH `(,PARAM ,PATH)
             %%LET-LIST))
      ((CL:ATOM PARAM)                                              ; Not so good.
       (CL:CERROR "Ignore this item." "Non-symbol variable name %"~S%" in ~S." PARAM ERRLOC))
      (T                                                            ; The destructuring case.
         (LET ((NEW-WHOLE (GENSYM)))
              (CL:PUSH `(,NEW-WHOLE ,PATH)
                    %%LET-LIST)
              (RECURSIVELY-ANALYZE PARAM NEW-WHOLE ERRLOC NEW-WHOLE])
```

(CL:DEFUN **CHECK-PARAMETER-NAME** (NAME ERRLOC)

```
   (CL:ASSERT (CL:SYMBOLP NAME)
          NIL "CHECK-PARAMETER-NAME should only be called with a symbol!")
   (COND
      ((NULL NAME)
       (CL:CERROR "Try to continue.  Good luck!" "NIL used as a parameter name in ~S" ERRLOC))
      ((CL:KEYWORDP NAME)
       (CL:CERROR "Use it anyway.  This is UGLY..." "The keyword ~S was used as a parameter name in ~S" NAME
             ERRLOC))
      ((MEMBER NAME CL:LAMBDA-LIST-KEYWORDS :TEST 'EQ)
       (CL:CERROR "Use it anyway.  This is UGLY..." "The lambda-list keyword ~S was used as a parameter name in
             ~S" NAME ERRLOC))))
```

(CL:DEFUN **PUSH-KEYWORD-BINDING** (VARIABLE CL:KEYWORD DEFAULT SUPPLIED-P-VAR REST-VAR TEMP-VAR ERRLOC)

```
   (CHECK-PARAMETER-NAME VARIABLE ERRLOC)
   (CL:UNLESS (CL:SYMBOLP SUPPLIED-P-VAR)
          (CL:ERROR "Non-symbolic supplied-p parameter %"~S%" found in arglist of ~S." SUPPLIED-P-VAR ERRLOC))
   (CL:PUSH `[,VARIABLE (COND
                           ((CL:SETQ ,TEMP-VAR (,*KEY-FINDER* ',CL:KEYWORD ,REST-VAR))
                            (CAR ,TEMP-VAR))
                           (T ,(OR DEFAULT *DEFAULT-DEFAULT*]
         %%LET-LIST)
   (CL:WHEN (NOT (NULL SUPPLIED-P-VAR))
       (CHECK-PARAMETER-NAME SUPPLIED-P-VAR ERRLOC)
       (CL:PUSH `[,SUPPLIED-P-VAR (NOT (NULL ,TEMP-VAR]
             %%LET-LIST)))
```

(CL:DEFUN **ANALYZE-REST** (CL:KEYWORD ARGLIST PATH ERRLOC WHOLE)

;;; This is complicated by the ''implicit PARSE-BODY'' convention. If a &body keyword is followed by a symbol, then it's just a normal &rest. If it's ;;; followed by a list of length one, then it's just like &rest using the CAR of that list. Otherwise, it's a list of length either 2 or 3: (body decls [doc]). The ;;; tail of the macro-call arguments is passed to PARSE-BODY along with the current lexical environment (as from &environment) and a ;;; doc-string-allowed-p argument of T iff the ''doc'' was specified (that is, the list after &body was of length three). PARSE-BODY returns three values ;;; that are then matched against ''body'', ''decls'', and ''doc'' respectively. Those three values can, in turn, be destructured, but it's not likely to be useful ;;; in any but the ''body'' case.

```
   (CL:WHEN (CL:ATOM ARGLIST)
         (CL:ERROR "Bad ~S arg in ~S." CL:KEYWORD ERRLOC))
   (SETQ %%UNBOUNDED-ARG-COUNT T)
   [LET ((REST-ARG (CAR ARGLIST)))
        (COND
```

```
           ((OR (CL:ATOM REST-ARG)
                (EQ CL:KEYWORD '&REST)
                (AND (EQ CL:KEYWORD '&BODY)
                     (CL:CONSP REST-ARG)
                     (NULL (CDR REST-ARG))
                     (PROGN (SETQ REST-ARG (CAR REST-ARG))
                            T)))                                    ; The non-parsing case of &rest or &body.
            (ANALYZE-PARAMETER REST-ARG PATH ERRLOC))
           [(AND (CL:CONSP REST-ARG)
                 (> (CL:LENGTH REST-ARG)
                    1))                                             ; Fancy case:
                                                                    ; (body-var decls-var [doc-var])
                                                                    ; an implicit call to PARSE-BODY.
            (CL:UNLESS %%ENV-ARG-NAME (CL:ERROR "The parsing version of &body is not allowed when no lexical
                                               environment is available."))
            (LET ((BODY (CL:FIRST REST-ARG))
                  (DECLS (CL:SECOND REST-ARG))
                  (DOC (CL:THIRD REST-ARG))
                  (PARSE-BODY-RESULT (GENSYM)))
                 (SETQ REST-ARG NIL)                                ; This makes &key illegal.
                 (CL:PUSH `[,PARSE-BODY-RESULT (CL:MULTIPLE-VALUE-LIST (PARSE-BODY ,PATH ,%%ENV-ARG-NAME
                                                                                    ,(NOT (NULL DOC]
                          %%LET-LIST)
                 (ANALYZE-PARAMETER BODY `(CL:FIRST ,PARSE-BODY-RESULT)
                          ERRLOC)
                 (ANALYZE-PARAMETER DECLS `(CL:SECOND ,PARSE-BODY-RESULT)
                          ERRLOC)
                 (CL:WHEN DOC
                     (ANALYZE-PARAMETER DOC `(CL:THIRD ,PARSE-BODY-RESULT)
                              ERRLOC))]
           (T (CL:ERROR "Bad &rest or &body arg in ~S." ERRLOC)))
```

;; Handle any arguments after &rest or &body.

```
        (CL:DO ((MORE (CDR ARGLIST)
                      (CDR MORE)))
               ((CL:ATOM MORE)
                (CL:IF (NULL MORE)
                       NIL
                       (CL:CERROR "Ignore the illegal terminator." "Dotted arglist terminator after &rest arg in
                                  ~S." ERRLOC)))
            (CASE (CAR MORE)
                ((&KEY) (CL:IF (NULL REST-ARG)
                               (CL:CERROR "Ignore the keywords." "The parsing version of &body was mixed with &key
                                          in arglist of ~S." ERRLOC)
                               (RETURN (ANALYZE-KEY (CDR MORE)
                                                    REST-ARG ERRLOC))))
                ((&AUX) (RETURN (ANALYZE-AUX (CDR MORE)
                                             ERRLOC)))
                ((&ALLOW-OTHER-KEYS) (CL:CERROR "Ignore it." "Stray &ALLOW-OTHER-KEYS in arglist of ~S." ERRLOC)
)
                ((&WHOLE) (COND
                              ((AND WHOLE (CL:CONSP (CDR MORE))
                                    (CL:SYMBOLP (CADR MORE)))
                               (CL:PUSH `(,(CADR MORE)
                                          ,WHOLE)
                                        %%LET-LIST)
                               (SETQ MORE (CDR MORE)))
                              (T (CL:ERROR "Ill-formed or illegal &whole arg in ~S." ERRLOC))))
                ((&ENVIRONMENT) (COND
                                    ((AND %%ENV-ARG-NAME (CL:CONSP (CDR MORE))
                                          (CL:SYMBOLP (CADR MORE)))
                                     (CL:PUSH `(,(CADR MORE)
                                                ,%%ENV-ARG-NAME)
                                              %%LET-LIST)
                                     (SETQ %%ENV-ARG-USED T)
                                     (SETQ MORE (CDR MORE)))
                                    (T (CL:ERROR "Ill-formed or illegal &environment arg in ~S." ERRLOC))))
                ((&CONTEXT) (COND
                                ((AND %%CTX-ARG-NAME (CL:CONSP (CDR MORE))
                                      (CL:SYMBOLP (CADR MORE)))
                                 (CL:PUSH `(,(CADR MORE)
                                            ,%%CTX-ARG-NAME)
                                          %%LET-LIST)
                                 (SETQ %%CTX-ARG-USED T)
                                 (SETQ MORE (CDR MORE)))
                                (T (CL:ERROR "Ill-formed or illegal &context arg in ~S." ERRLOC))))
                (CL:OTHERWISE (CL:CERROR "Ignore it." "Stray parameter %"~S%" found in arglist of ~S."
                                         (CAR MORE)
                                         ERRLOC)))))])
```

(CL:DEFUN **RECURSIVELY-ANALYZE** (ARGLIST PATH ERRLOC WHOLE)

;; Make a recursive call on ANALYZE, being careful to shield the data-structures of outer calls and to make certain constructs illegal. The bindings of ;; MIN-ARGS, ARG-COUNT, and UNBOUNDED-ARG-COUNT are for shielding and those of ENV-ARG-NAME and CTX-ARG-NAME are to disallow ;; &environment and &context respectively.

```
(LET ((%%MIN-ARGS 0)
```

```
         (%%ARG-COUNT 0)
         (%%UNBOUNDED-ARG-COUNT NIL)
         (%%ENV-ARG-NAME NIL)
         (%%CTX-ARG-NAME NIL))
        (ANALYZE ARGLIST PATH ERRLOC WHOLE)))
```

(CL:DEFUN **DEFMACRO-ARG-TEST** (ARGS)

;; Return a form which tests whether an illegal number of arguments have been supplied. Args is a form which evaluates to the list of arguments.

```
[COND
   ((AND (ZEROP %%MIN-ARGS)
         %%UNBOUNDED-ARG-COUNT)
    NIL)
   [(ZEROP %%MIN-ARGS)
    `(> (CL:LENGTH ,ARGS)
       ,%%ARG-COUNT]
   [%%UNBOUNDED-ARG-COUNT `(< (CL:LENGTH ,ARGS)
                              ,%%MIN-ARGS]
   [(= %%MIN-ARGS %%ARG-COUNT)
    `(CL:/= (CL:LENGTH ,ARGS)
            ,%%MIN-ARGS]
   (T `(OR (> (CL:LENGTH ,ARGS)
              ,%%ARG-COUNT)
           (< (CL:LENGTH ,ARGS)
              ,%%MIN-ARGS])
```

;; Testing the argument-list parsing

(CL:DEFVAR **ANALYZE-TESTS**

```
   '((CL:MULTIPLE-VALUE-LIST (PARSE-DEFMACRO '((&WHOLE HEAD MOUTH &OPTIONAL EYE1 (EYE2 7 EYE2-P))
                                               ([FIN1 LENGTH1 &KEY ONE (TWO 8)
                                                      ((:THREE TROIS)
                                                       3 TRES-P)
                                                      ((:FOUR (QUATRE QUATRO))
                                                       '(4 4]
                                                &OPTIONAL
                                                ((FIN2 LENGTH2)
                                                 9 FL2-P))
                                               TAIL &REST (FOO BAR BAZ)
                                               &ENVIRONMENT ENV)
                                    'WHOLE-ARG
                                    '((CODE))
                                    'ERRLOC :ENVIRONMENT '*ENV* :ERROR-STRING "Ack!"))
     '((&WHOLE HEAD MOUTH EYE1 EYE2)
       ((FIN1 LENGTH1)
        (FIN2 LENGTH2))
       TAIL)
     '((&WHOLE HEAD MOUTH &OPTIONAL EYE1 (EYE2 7 EYE2-P))
       ([FIN1 LENGTH1 &KEY ONE (TWO 8)
              ((:THREE TROIS)
               3 TRES-P)
              ((:FOUR (QUATRE QUATRO))
               '(4 4]
        &OPTIONAL
        ((FIN2 LENGTH2)
         9 FL2-P))
       TAIL &REST (FOO BAR BAZ)
       &ENVIRONMENT ENV)))
```

;; Runtime support functions

(CL:DEFUN **KEYWORD-TEST** (ARGS KEYS)

;; Signal an error unless

;; -- one of the keywords on ARGS is :ALLOW-OTHER-KEYS and it has a non-NIL value, or

;; -- all of the keywords on ARGS are also on KEYS.

;; Note that we should search ARGS by CDDR and KEYS by CDR.

```
(LET ((EXTRA-KEY-FOUND NIL)
      (ALLOW-OTHER-KEYS-P NIL))
     [FOR TAIL ON ARGS BY (CDDR TAIL) DO (CL:WHEN (EQ (CAR TAIL)
                                                      :ALLOW-OTHER-KEYS)
                                                  (SETQ ALLOW-OTHER-KEYS-P (CADR TAIL)))
                                         (CL:UNLESS (CL:MEMBER (CAR TAIL)
                                                               KEYS :TEST #'EQ)
                                                    (CL:SETQ EXTRA-KEY-FOUND (CAR TAIL)))]
     (CL:WHEN (AND EXTRA-KEY-FOUND (NOT ALLOW-OTHER-KEYS-P))
            (CL:ERROR "Extraneous keyword %"~S%" given." EXTRA-KEY-FOUND))))
```

(CL:DEFUN **FIND-KEYWORD** (CL:KEYWORD KEYLIST)

;; If keyword is present in the keylist, return a list of its argument. Else, return NIL.

```
   (CL:DO ((L KEYLIST (CDDR L)))
          ((CL:ENDP L)
           NIL)
       (CL:WHEN (CL:ENDP (CDR L))
           (CL:CERROR "Stick a NIL on the end and go on." "Unpaired item in keyword portion of macro call.")
           (RPLACD L (LIST NIL)))
       (CL:WHEN (EQ (CAR L)
                    CL:KEYWORD)
           (RETURN (LIST (CADR L))))))
```

;; Arrange to use the correct compiler

```
(PUTPROPS CMLPARSE FILETYPE CL:COMPILE-FILE)

(PUTPROPS CMLPARSE COPYRIGHT ("Venue & Xerox Corporation" 1986 1988 1990))
```

FUNCTION INDEX

VARIABLE INDEX

PROPERTY INDEX