

File created: 16-May-90 14:56:15 {DSK}<usr>local>lde>lispcore>sources>CMLWALK.;2

changes to: (VARS CMLWALKCOMS)

previous date: 17-Jun-87 17:43:58 {DSK}<usr>local>lde>lispcore>sources>CMLWALK.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

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

(RPAQQ **CMLWALKCOMS**

[(FUNCTIONS XCL:ONCE-ONLY)

; not a wonderful place for it, but CMLMACROS comes too early
; in the loadup.

(VARIABLES *WALK-FUNCTION* *WALK-FORM* *DECLARATIONS* *LEXICAL-VARIABLES* *ENVIRONMENT* *WALK-COPY*)
(FUNCTIONS WITH-NEW-CONTOUR NOTE-LEXICAL-BINDING NOTE-DECLARATION)
(FUNCTIONS VARIABLE-SPECIAL-P VARIABLE-LEXICAL-P GET-WALKER-TEMPLATE)
(FUNCTIONS WALK-FORM)
(FNS WALK-FORM-INTERNAL WALK-TEMPLATE WALK-TEMPLATE-HANDLE-REPEAT WALK-TEMPLATE-HANDLE-REPEAT-1
WALK-LIST WALK-RECONS)
(FUNCTIONS WALK-RELIST*)
(FNS WALK-DECLARATIONS WALK-ARGLIST WALK-LAMBDA)
(COMS (PROP WALKER-TEMPLATE CL:COMPILER-LET)
(FNS WALK-COMPILER-LET)
(PROP WALKER-TEMPLATE DECLARE)
(FNS WALK-UNEXPECTED-DECLARE)
(PROP WALKER-TEMPLATE LET PROG LET* PROG*)
(FNS WALK-LET WALK-LET* WALK-LET/LET*)
(PROP WALKER-TEMPLATE CL:TAGBODY)
(FNS WALK-TAGBODY)
(PROP WALKER-TEMPLATE FUNCTION CL:FUNCTION GO CL:IF CL:MULTIPLE-VALUE-CALL CL:MULTIPLE-VALUE-PROG1
PROGN CL:PROGV QUOTE CL:RETURN-FROM RETURN CL:SETQ CL:BLOCK CL:CATCH CL:EVAL-WHEN THE
CL:THROW CL:UNWIND-PROTECT LOAD-TIME-EVAL COND CL:UNWIND-PROTECT SETQ AND OR))

(COMS ;; for Interlisp

(PROP WALKER-TEMPLATE RPAQ? RPAQ XNLSETQ ERSETQ NLSETQ RESETVARS))
(PROP FILETYPE CMLWALK)
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVERS
(ADDVARS (NLAMA)
(NLAML)
(LAMA WALK-TAGBODY WALK-LET/LET* WALK-LET* WALK-LET WALK-UNEXPECTED-DECLARE
WALK-COMPILER-LET WALK-LAMBDA WALK-ARGLIST WALK-DECLARATIONS WALK-RECONS
WALK-TEMPLATE-HANDLE-REPEAT-1 WALK-TEMPLATE-HANDLE-REPEAT WALK-TEMPLATE
WALK-FORM-INTERNAL]))

(DEFMACRO **XCL:ONCE-ONLY** (XCL::VARS &BODY XCL::BODY)

:: ONCE-ONLY assures that the forms given as vars are evaluated in the proper order, once only. Used in the body of macro definitions. Taken from
:: Zeta Lisp.

```
[LET* [(XCL::GENSYM-VAR (CL:GENSYM))
(XCL::RUN-TIME-VARS (CL:GENSYM))
(XCL::RUN-TIME-VALS (CL:GENSYM))
(XCL::EXPAND-TIME-VAL-FORMS (FOR XCL::VAR IN XCL::VARS
COLLECT `(CL:IF (OR (CL:SYMBOLP ,XCL::VAR)
(CL:CONSTANTP ,XCL::VAR))
,XCL::VAR
(LET ((,XCL::GENSYM-VAR (CL:GENSYM)))
(CL:PUSH ,XCL::GENSYM-VAR ,XCL::RUN-TIME-VARS)
(CL:PUSH ,XCL::VAR ,XCL::RUN-TIME-VALS)
,XCL::GENSYM-VAR))])
` (LET* [,XCL::RUN-TIME-VARS ,XCL::RUN-TIME-VALS
(XCL::WRAPPED-BODY (LET ,(FOR XCL::VAR IN XCL::VARS AS XCL::EXPAND-TIME-VAL-FORM
IN XCL::EXPAND-TIME-VAL-FORMS COLLECT (LIST XCL::VAR
XCL::EXPAND-TIME-VAL-FORM
),@XCL::BODY]
` (LET ,(FOR XCL::RUN-TIME-VAR IN (CL:REVERSE XCL::RUN-TIME-VARS) AS XCL::RUN-TIME-VAL
IN (CL:REVERSE XCL::RUN-TIME-VALS) COLLECT (LIST XCL::RUN-TIME-VAR XCL::RUN-TIME-VAL)
),XCL::WRAPPED-BODY)])
```

:: not a wonderful place for it, but CMLMACROS comes too early in the loadup.

(CL:DEFVAR ***WALK-FUNCTION*** NIL
"the function being called on each sub-form in the code-walker")

(CL:DEFVAR ***WALK-FORM***

"When the first argument to the IF template in the code-walker is a list, it will be evaluated with *walk-form* bound to the form currently being walked.")

(CL:DEFVAR *DECLARATIONS* "a list of the declarations currently in effect while codewalking")

(CL:DEFVAR *LEXICAL-VARIABLES* NIL ; used in walker to hold list of lexical variables available)

(CL:DEFVAR *ENVIRONMENT* "while codewalking, this is the lexical environment as far as macros are concerned")

(CL:DEFVAR *WALK-COPY* "while walking, this is true if we are making a copy of the expression being walked")

(DEFMACRO WITH-NEW-CONTOUR (&BODY BODY)

;; WITH-NEW-CONTOUR is used to enter a new lexical binding contour which inherits from the existing one. Using WITH-NEW-CONTOUR is often
;; overkill: It would suffice for the walker to rebind *LEXICAL-VARIABLES* and *DECLARATIONS* when walking LET and rebind
;; *ENVIRONMENT* and *DECLARATIONS* when walking MACROLET etc. WITH-NEW-CONTOUR is much more convenient and just as correct.
;; *

`(LET ((*DECLARATIONS* NIL)
(*LEXICAL-VARIABLES* *LEXICAL-VARIABLES*)
(*ENVIRONMENT* *ENVIRONMENT*))
,@BODY))

(DEFMACRO NOTE-LEXICAL-BINDING (THING)

`(CL:PUSH ,THING *LEXICAL-VARIABLES*))

(DEFMACRO NOTE-DECLARATION (CL:DECLARATION)

`(CL:PUSH ,CL:DECLARATION *DECLARATIONS*))

(CL:DEFUN VARIABLE-SPECIAL-P (VAR)

(* Imm "27-May-86 15:42")

(OR (for DECL in *DECLARATIONS* do (AND (EQ (CAR DECL)
'CL:SPECIAL)
(FMEMB VAR (CDR DECL))
(RETURN T)))
(VARIABLE-GLOBALLY-SPECIAL-P VAR)))

(CL:DEFUN VARIABLE-LEXICAL-P (VAR)

(* Imm "11-Apr-86 10:59")

(AND (NOT (VARIABLE-SPECIAL-P VAR))
(CL:MEMBER VAR *LEXICAL-VARIABLES* :TEST (FUNCTION EQ))))

(CL:DEFUN GET-WALKER-TEMPLATE (X)

(* Imm "24-May-86 14:48")

(CL:IF (NOT (CL:SYMBOLP X))
'(CL:LAMBDA :REPEAT (:EVAL))
(GET X 'WALKER-TEMPLATE)))

(CL:DEFUN WALK-FORM (FORM &KEY ([:DECLARATIONS *DECLARATIONS*]
NIL)

([:LEXICAL-VARIABLES *LEXICAL-VARIABLES*]
NIL)
(:ENVIRONMENT *ENVIRONMENT*)
NIL)
(:WALK-FUNCTION *WALK-FUNCTION*)
(FUNCTION (CL:LAMBDA (X IGNORE)
IGNORE X)))
(:COPY *WALK-COPY*)
T))

"Walk FORM, expanding all macros, calling :WALK-FUNCTION on each subform :COPY is true (default), will return the expansion"

(WALK-FORM-INTERNAL FORM ' :EVAL))

(DEFINEQ

(WALK-FORM-INTERNAL

[CL:LAMBDA (FORM CONTEXT &AUX FN TEMPLATE WALK-NO-MORE-P NEWFORM)

(* Imm "24-May-86 20:28")

;; WALK-FORM-INTERNAL is the main driving function for the code walker. It takes a form and the current context and walks the form
;; calling itself or the appropriate template recursively.

(CL:MULTIPLE-VALUE-SETQ (NEWFORM WALK-NO-MORE-P)
(CL:FUNCALL *WALK-FUNCTION* FORM CONTEXT))
(COND

```
(WALK-NO-MORE-P NEWFORM)
(NOT (EQ FORM NEWFORM))
(WALK-FORM-INTERNAL NEWFORM CONTEXT))
(NOT (CL:CONSP FORM))
FORM)
(NOT (CL:SYMBOLP (CAR FORM)))
(WALK-TEMPLATE FORM '(:CALL :REPEAT (:EVAL))
CONTEXT))
((SETQ TEMPLATE (GET-WALKER-TEMPLATE (CAR FORM)))
(CL:IF (CL:SYMBOLP TEMPLATE)
(CL:FUNCALL TEMPLATE FORM CONTEXT)
(WALK-TEMPLATE FORM TEMPLATE CONTEXT)))
((NEQ FORM (SETQ FORM (CL:MACROEXPAND-1 FORM *ENVIRONMENT*)))
(WALK-FORM-INTERNAL FORM CONTEXT))
(T ;; Otherwise, walk the form as if its just a standard function call using a template for standard function call.
(WALK-TEMPLATE FORM '(:CALL :REPEAT (:EVAL))
CONTEXT])
```

(WALK-TEMPLATE

```
[CL:LAMBDA (FORM TEMPLATE CONTEXT) ; (* Imm "24-May-86 16:43")
(CL:IF (CL:ATOM TEMPLATE)
(CL:ECASE TEMPLATE
((CALL :CALL) (if (CL:CONSP FORM)
then (WALK-LAMBDA FORM NIL)
else FORM))
((QUOTE NIL PPE :ERROR) FORM)
(:EVAL EVAL :FUNCTION FUNCTION :TEST TEST :EFFECT EFFECT :RETURN RETURN) (WALK-FORM-INTERNAL
FORM
':EVAL))
((SET :SET) (WALK-FORM-INTERNAL FORM ':SET))
(CL:LAMBDA (WALK-LAMBDA FORM CONTEXT)))
(CASE (CAR TEMPLATE)
(CL:IF (LET ((*WALK-FORM* FORM))
(WALK-TEMPLATE FORM (COND
((CL:IF (LISTP (CL:SECOND TEMPLATE))
(CL:EVAL (CL:SECOND TEMPLATE))
(CL:FUNCALL (CL:SECOND TEMPLATE)
FORM))
(CL:THIRD TEMPLATE))
(T (CL:FOURTH TEMPLATE)))
CONTEXT)))
((REPEAT :REPEAT) (WALK-TEMPLATE-HANDLE-REPEAT FORM (CDR TEMPLATE)
(CL:NTHCDR (- (CL:LENGTH FORM)
(CL:LENGTH (CDDR TEMPLATE)))
FORM)
CONTEXT))
(T [COND
((CL:ATOM FORM)
FORM)
(T (WALK-RECONS FORM (WALK-TEMPLATE (CAR FORM)
(CAR TEMPLATE)
CONTEXT)
(WALK-TEMPLATE (CDR FORM)
(CDR TEMPLATE)
CONTEXT))))])])
```

(WALK-TEMPLATE-HANDLE-REPEAT

```
(CL:LAMBDA (FORM TEMPLATE STOP-FORM CONTEXT) ; (* Imm "11-Apr-86 12:05")
(CL:IF (EQ FORM STOP-FORM)
(WALK-TEMPLATE FORM (CDR TEMPLATE)
CONTEXT)
(WALK-TEMPLATE-HANDLE-REPEAT-1 FORM TEMPLATE (CAR TEMPLATE)
STOP-FORM CONTEXT)))
```

(WALK-TEMPLATE-HANDLE-REPEAT-1

```
[CL:LAMBDA (FORM TEMPLATE REPEAT-TEMPLATE STOP-FORM CONTEXT) ; (* Imm "24-May-86 16:43")
(COND
((NULL FORM)
NIL)
((EQ FORM STOP-FORM)
(CL:IF (NULL REPEAT-TEMPLATE)
(WALK-TEMPLATE STOP-FORM (CDR TEMPLATE)
CONTEXT)
(CL:ERROR "While handling repeat:
~%~%Ran into stop while still in repeat template.))))
((NULL REPEAT-TEMPLATE)
(WALK-TEMPLATE-HANDLE-REPEAT-1 FORM TEMPLATE (CAR TEMPLATE)
STOP-FORM CONTEXT))
(T (WALK-RECONS FORM (WALK-TEMPLATE (CAR FORM)
(CAR REPEAT-TEMPLATE)
CONTEXT)
(WALK-TEMPLATE-HANDLE-REPEAT-1 (CDR FORM)
TEMPLATE
```

(CDR REPEAT-TEMPLATE)
STOP-FORM CONTEXT])

(WALK-LIST

[LAMBDA (LIST FN)

(* Imm "24-May-86 16:43")
(* copy list walking each element)

(CL:IF LIST
(WALK-RECONS LIST (CL:FUNCALL FN (CAR LIST))
(WALK-LIST (CDR LIST)
FN)))])

(WALK-RECONS

(CL:LAMBDA (X CAR CDR)
(CL:IF *WALK-COPY*
(CL:IF (OR (NOT (EQ (CAR X)
CAR))
(NOT (EQ (CDR X)
CDR)))
(CONS CAR CDR)
X)
NIL)))

(* Imm "24-May-86 16:43")

(DEFMACRO WALK-RELIST*

(X FIRST &REST CL:REST)
(CL:IF CL:REST
(WALK-RECONS ,X ,FIRST (WALK-RELIST* (CDR ,X)
,@CL:REST))
FIRST))

(DEFINEQ

(WALK-DECLARATIONS

[CL:LAMBDA (BODY FN &OPTIONAL DOC-STRING-P DECLARATIONS &AUX (FORM (CAR BODY)))
(* Imm "18-Jun-86 14:35")
(* skips over declarations)

(COND
((AND (STRINGP FORM) (* might be a doc string *)
(CDR BODY) (* isn't the returned value *)
(NULL DOC-STRING-P) (* no doc string yet *)
(NULL DECLARATIONS)) (* no declarations yet *)
(WALK-RECONS BODY FORM (WALK-DECLARATIONS (CDR BODY)
FN T)))

((AND (LISTP FORM)
(EQ (CAR FORM)
'DECLARE)) (* Got a real declaration. Record it, look for more.
*)

(CL:DOLIST (CL:DECLARATION (CDR FORM))
(NOTE-DECLARATION CL:DECLARATION)
(CL:PUSH CL:DECLARATION DECLARATIONS))
(WALK-RECONS BODY FORM (WALK-DECLARATIONS (CDR BODY)
FN DOC-STRING-P DECLARATIONS)))

([AND (CL:CONSP FORM)
(NULL (GET-WALKER-TEMPLATE (CAR FORM)))
(NOT (EQ FORM (SETQ FORM (CL:MACROEXPAND-1 FORM *ENVIRONMENT*])

(* * When we macroexpanded this form we got something else back.
Maybe this is a macro which expanded into a declare? Recurse to find out.)

(WALK-DECLARATIONS (CONS FORM (CDR BODY))
FN DOC-STRING-P DECLARATIONS))

(T

(* Now that we have walked and recorded the declarations, call the function our caller provided to expand the body.
We call that function rather than passing the real-body back, because we are RECONSING up the new body.)

(CL:FUNCALL FN BODY])

(WALK-ARGLIST

[CL:LAMBDA (ARGLIST CONTEXT &OPTIONAL DESTRUCTURINGP &AUX ARG) (* Imm "24-May-86 16:44")

(COND
(NULL ARGLIST)
NIL)
[(CL:SYMBOLP (CL:SETQ ARG (CAR ARGLIST)))
(OR (CL:MEMBER ARG CL:LAMBDA-LIST-KEYWORDS :TEST (FUNCTION EQ))
(NOTE-LEXICAL-BINDING ARG))
(WALK-RECONS ARGLIST ARG (WALK-ARGLIST (CDR ARGLIST)
CONTEXT
(AND DESTRUCTURINGP (NOT (CL:MEMBER ARG CL:LAMBDA-LIST-KEYWORDS
:TEST (FUNCTION EQ))

(CL:CONSP ARG)
(PROG1 (CL:IF DESTRUCTURINGP
(WALK-ARGLIST ARG CONTEXT DESTRUCTURINGP)

```

(WALK-RECONS ARGLIST (WALK-RELIST* ARG (CAR ARG)
                        (WALK-FORM-INTERNAL (CADR ARG)
                                              ' :EVAL)
                        (CDDR ARG))
 (WALK-ARGLIST (CDR ARGLIST)
               CONTEXT NIL))
(CL:IF (CL:SYMBOLP (CAR ARG))
      (NOTE-LEXICAL-BINDING (CAR ARG))
      (NOTE-LEXICAL-BINDING (CADR ARG)))
(OR (NULL (CDDR ARG))
    (NOT (CL:SYMBOLP (CADDR ARG)))
    (NOTE-LEXICAL-BINDING ARG)))
(T (CL:ERROR "Can't understand something in the arglist ~S" ARGLIST])

```

(WALK-LAMBDA

```

[CL:LAMBDA (FORM CONTEXT) (* Imm "24-May-86 16:44")
  (WITH-NEW-CONTOUR (LET* [(ARGLIST (CADR FORM))
                           (BODY (CDDR FORM))
                           (WALKED-ARGLIST NIL)
                           (WALKED-BODY (WALK-DECLARATIONS BODY (FUNCTION (CL:LAMBDA
                                                                              (REAL-BODY)
                                                                              (CL:SETQ WALKED-ARGLIST
                                                                              (WALK-ARGLIST
                                                                              ARGLIST
                                                                              CONTEXT))
                                                                              (WALK-TEMPLATE
                                                                              REAL-BODY
                                                                              ' (:REPEAT (:EVAL))
                                                                              CONTEXT]
                                                                              (WALK-RELIST* FORM (CAR FORM)
                                                                              WALKED-ARGLIST WALKED-BODY]))

```

)

(PUTPROPS **CL:COMPILER-LET WALKER-TEMPLATE** WALK-COMPILER-LET)

(DEFINEQ

(WALK-COMPILER-LET

```

[CL:LAMBDA (FORM CONTEXT) (* gbn "7-Aug-86 18:21")
  ; bind the variables, but then return the COMPILER-LET
  (LET [(VARS (CL:MAPCAR (FUNCTION (LAMBDA (X)
                                     (CL:IF (CL:CONSP X)
                                             (CAR X)
                                             X)
                                     (CADR FORM)))
                        (VALS (CL:MAPCAR (FUNCTION (CL:LAMBDA (X)
                                                         (CL:IF (CL:CONSP X)
                                                             (CL:EVAL (CADR X)
                                                             NIL)))
                                     (CADR FORM]
                        (CL:PROGV VARS VALS
                                (WALK-TEMPLATE FORM '(NIL NIL :REPEAT (:EVAL)
                                :RETURN)
                                CONTEXT))])

```

)

(PUTPROPS **DECLARE WALKER-TEMPLATE** WALK-UNEXPECTED-DECLARE)

(DEFINEQ

(WALK-UNEXPECTED-DECLARE

```

[CL:LAMBDA (FORM CONTEXT) (* Imm "24-May-86 22:27")
  (DECLARE (IGNORE CONTEXT))
  (CL:WARN "Encountered declare ~S in a place where a declare was not expected." FORM)
  FORM))

```

)

(PUTPROPS **LET WALKER-TEMPLATE** WALK-LET)

(PUTPROPS **PROG WALKER-TEMPLATE** WALK-LET)

(PUTPROPS **LET* WALKER-TEMPLATE** WALK-LET*)

(PUTPROPS **PROG* WALKER-TEMPLATE** WALK-LET*)

(DEFINEQ

(WALK-LET

```

[CL:LAMBDA (FORM CONTEXT)
  (WALK-LET/LET* FORM CONTEXT NIL))

```

(WALK-LET*

(CL:LAMBDA (FORM CONTEXT)
(WALK-LET/LET* FORM CONTEXT T))

(WALK-LET/LET*

[CL:LAMBDA
(FORM CONTEXT SEQUENTIALP)
(LET ((OLD-DECLARATIONS *DECLARATIONS*)
(OLD-LEXICAL-VARIABLES *LEXICAL-VARIABLES*))
(WITH-NEW-CONTOUR
(LET* [(LET/LET* (CAR FORM))
(BINDINGS (CADR FORM))
(BODY (CDDR FORM))
WALKED-BINDINGS
(WALKED-BODY (WALK-DECLARATIONS
BODY
(FUNCTION (CL:LAMBDA
(REAL-BODY)
[CL:SETQ WALKED-BINDINGS
(WALK-LIST BINDINGS
(FUNCTION (LAMBDA (BINDING)
(CL:IF (CL:SYMBOLP BINDING)
(PROG1 BINDING (NOTE-LEXICAL-BINDING
BINDING))
(PROG1
(LET ((*DECLARATIONS*
OLD-DECLARATIONS)
(*LEXICAL-VARIABLES*
(CL:IF SEQUENTIALP
LEXICAL-VARIABLES
OLD-LEXICAL-VARIABLES)))
(WALK-RELIST*
BINDING
(CAR BINDING)
(WALK-FORM-INTERNAL
(CADR BINDING)
CONTEXT)
(CDDR BINDING)))
(NOTE-LEXICAL-BINDING (CAR BINDING)))
]]

(WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
CONTEXT]
(WALK-RELIST* FORM LET/LET* WALKED-BINDINGS WALKED-BODY))

)
(PUTPROPS CL:TAGBODY WALKER-TEMPLATE WALK-TAGBODY)
(DEFINEQ

(WALK-TAGBODY

[CL:LAMBDA (FORM CONTEXT)
(WALK-RECONS FORM (CAR FORM))
(WALK-LIST (CDR FORM)
(FUNCTION (LAMBDA (X)
(WALK-FORM-INTERNAL X (CL:IF (CL:SYMBOLP X)
'QUOTE
CONTEXT)))

)
(PUTPROPS FUNCTION WALKER-TEMPLATE (NIL :CALL))
(PUTPROPS CL:FUNCTION WALKER-TEMPLATE (NIL :CALL))
(PUTPROPS GO WALKER-TEMPLATE (NIL NIL))
(PUTPROPS CL:IF WALKER-TEMPLATE (NIL :TEST :RETURN :RETURN))
(PUTPROPS CL:MULTIPLE-VALUE-CALL WALKER-TEMPLATE (NIL :EVAL :REPEAT (:EVAL)))
(PUTPROPS CL:MULTIPLE-VALUE-PROG1 WALKER-TEMPLATE (NIL :RETURN :REPEAT (:EVAL)))
(PUTPROPS PROGN WALKER-TEMPLATE (NIL :REPEAT (:EVAL)))
(PUTPROPS CL:PROGV WALKER-TEMPLATE (NIL :EVAL :EVAL :REPEAT (:EVAL)))
(PUTPROPS QUOTE WALKER-TEMPLATE (NIL QUOTE))
(PUTPROPS CL:RETURN-FROM WALKER-TEMPLATE (NIL NIL :EVAL))
(PUTPROPS RETURN WALKER-TEMPLATE (NIL :EVAL))
(PUTPROPS CL:SETQ WALKER-TEMPLATE (NIL :REPEAT (:SET :EVAL)))
(PUTPROPS CL:BLOCK WALKER-TEMPLATE (NIL NIL :REPEAT (:EVAL)))

```
(PUTPROPS CL:CATCH WALKER-TEMPLATE (NIL :EVAL :REPEAT (:EVAL)))
(PUTPROPS CL:EVAL-WHEN WALKER-TEMPLATE (NIL NIL :REPEAT (:EVAL)))
(PUTPROPS THE WALKER-TEMPLATE (NIL NIL :EVAL))
(PUTPROPS CL:THROW WALKER-TEMPLATE (NIL :EVAL :EVAL))
(PUTPROPS CL:UNWIND-PROTECT WALKER-TEMPLATE (NIL :EVAL :REPEAT (:EVAL)))
(PUTPROPS LOAD-TIME-EVAL WALKER-TEMPLATE (NIL :EVAL))
(PUTPROPS COND WALKER-TEMPLATE [NIL :REPEAT ((:REPEAT (:EVAL))
(PUTPROPS CL:UNWIND-PROTECT WALKER-TEMPLATE (NIL :EVAL :REPEAT (:EVAL)))
(PUTPROPS SETQ WALKER-TEMPLATE (NIL :SET :EVAL))
(PUTPROPS AND WALKER-TEMPLATE (NIL :REPEAT (:EVAL)))
(PUTPROPS OR WALKER-TEMPLATE (NIL :REPEAT (:EVAL)))
```

;; for Interlisp

```
(PUTPROPS RPAQ? WALKER-TEMPLATE (NIL :SET :EVAL))
(PUTPROPS RPAQ WALKER-TEMPLATE (NIL :SET :EVAL))
(PUTPROPS XNLSETQ WALKER-TEMPLATE (NIL :REPEAT (:EVAL)))
(PUTPROPS ERSETQ WALKER-TEMPLATE (NIL :REPEAT (:EVAL)))
(PUTPROPS NLSETQ WALKER-TEMPLATE (NIL :REPEAT (:EVAL)))
(PUTPROPS RESETVARS WALKER-TEMPLATE WALK-LET)
(PUTPROPS CMLWALK FILETYPE :COMPILE-FILE)
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
(ADDTOVAR NLAMA )
(ADDTOVAR NLAML )
(ADDTOVAR LAMA WALK-TAGBODY WALK-LET/LET* WALK-LET* WALK-LET WALK-UNEXPECTED-DECLARE WALK-COMPILER-LET
WALK-LAMBDA WALK-ARGLIST WALK-DECLARATIONS WALK-RECONS WALK-TEMPLATE-HANDLE-REPEAT-1
WALK-TEMPLATE-HANDLE-REPEAT WALK-TEMPLATE WALK-FORM-INTERNAL)
)
```

(RPAQQ **CMLWALKCOMS**

```
[ (FUNCTIONS XCL:ONCE-ONLY)
; not a wonderful place for it, but CMLMACROS comes too early
; in the loadup.
```

```
(VARIABLES *WALK-FUNCTION* *WALK-FORM* *DECLARATIONS* *LEXICAL-VARIABLES* *ENVIRONMENT* *WALK-COPY*)
(FUNCTIONS WITH-NEW-CONTOUR NOTE-LEXICAL-BINDING NOTE-DECLARATION)
(FUNCTIONS VARIABLE-SPECIAL-P VARIABLE-LEXICAL-P GET-WALKER-TEMPLATE)
(FUNCTIONS WALK-FORM)
(FNS WALK-FORM-INTERNAL WALK-TEMPLATE WALK-TEMPLATE-HANDLE-REPEAT WALK-TEMPLATE-HANDLE-REPEAT-1
WALK-LIST WALK-RECONS)
(FUNCTIONS WALK-RELIST*)
(FNS WALK-DECLARATIONS WALK-ARGLIST WALK-LAMBDA)
(COMS (PROP WALKER-TEMPLATE CL:COMPILER-LET)
(FNS WALK-COMPILER-LET)
(PROP WALKER-TEMPLATE DECLARE)
(FNS WALK-UNEXPECTED-DECLARE)
(PROP WALKER-TEMPLATE LET PROG LET* PROG*)
(FNS WALK-LET WALK-LET* WALK-LET/LET*)
(PROP WALKER-TEMPLATE CL:TAGBODY)
(FNS WALK-TAGBODY)
(PROP WALKER-TEMPLATE FUNCTION CL:FUNCTION GO CL:IF CL:MULTIPLE-VALUE-CALL CL:MULTIPLE-VALUE-PROG1
PROGN CL:PROGV QUOTE CL:RETURN-FROM RETURN CL:SETQ CL:BLOCK CL:CATCH CL:EVAL-WHEN THE
CL:THROW CL:UNWIND-PROTECT LOAD-TIME-EVAL COND CL:UNWIND-PROTECT SETQ AND OR))
```

(COMS ;; for Interlisp

```
(PROP WALKER-TEMPLATE RPAQ? RPAQ XNLSETQ ERSETQ NLSETQ RESETVARS))
(PROP FILETYPE CMLWALK)
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
(ADDVARS (NLAMA)
(NLAML)
(LAMA WALK-TAGBODY WALK-LET/LET* WALK-LET* WALK-LET WALK-UNEXPECTED-DECLARE
WALK-COMPILER-LET WALK-LAMBDA WALK-ARGLIST WALK-DECLARATIONS WALK-RECONS
WALK-TEMPLATE-HANDLE-REPEAT-1 WALK-TEMPLATE-HANDLE-REPEAT]))
```

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR **NLAMA**)

{MEDLEY}<sources>CMLWALK.;1

Page 8

(ADDTOVAR **NLAML**)

(ADDTOVAR **LAMA** WALK-TAGBODY WALK-LET/LET* WALK-LET* WALK-LET WALK-UNEXPECTED-DECLARE WALK-COMPILER-LET
WALK-LAMBDA WALK-ARGLIST WALK-DECLARATIONS WALK-RECONS WALK-TEMPLATE-HANDLE-REPEAT-1
WALK-TEMPLATE-HANDLE-REPEAT)
)

(PUTPROPS **CMLWALK COPYRIGHT** ("Venue & Xerox Corporation" 1986 1987 1990))

FUNCTION INDEX

GET-WALKER-TEMPLATE	2	WALK-FORM-INTERNAL	2	WALK-TAGBODY	6
VARIABLE-LEXICAL-P	2	WALK-LAMBDA	5	WALK-TEMPLATE	3
VARIABLE-SPECIAL-P	2	WALK-LET	5	WALK-TEMPLATE-HANDLE-REPEAT	3
WALK-ARGLIST	4	WALK-LET*	5	WALK-TEMPLATE-HANDLE-REPEAT-1	3
WALK-COMPILER-LET	5	WALK-LET/LET*	6	WALK-UNEXPECTED-DECLARE	5
WALK-DECLARATIONS	4	WALK-LIST	4		
WALK-FORM	2	WALK-RECONS	4		

PROPERTY INDEX

AND	7	CL:FUNCTION	6	PROG	5	CL:SETQ	6
CL:BLOCK	6	GO	6	PROG*	5	SETQ	7
CL:CATCH	7	CL:IF	6	PROGN	6	CL:TAGBODY	6
CMLWALK	7	LET	5	CL:PROGV	6	THE	7
CL:COMPILER-LET	5	LET*	5	QUOTE	6	CL:THROW	7
COND	7	LOAD-TIME-EVAL	7	RESETVARS	7	CL:UNWIND-PROTECT	7
DECLARE	5	CL:MULTIPLE-VALUE-CALL	6	RETURN	6	XNLSETQ	7
ERSETQ	7	CL:MULTIPLE-VALUE-PROG1	6	CL:RETURN-FROM	6		
CL:EVAL-WHEN	7	NLSETQ	7	RPAQ	7		
FUNCTION	6	OR	7	RPAQ?	7		

VARIABLE INDEX

DECLARATIONS	2	*LEXICAL-VARIABLES*	2	*WALK-FORM*	1
ENVIRONMENT	2	*WALK-COPY*	2	*WALK-FUNCTION*	1

MACRO INDEX

NOTE-DECLARATION	2	XCL:ONCE-ONLY	1	WITH-NEW-CONTOUR	2
NOTE-LEXICAL-BINDING	2	WALK-RELIST*	4		
