

File created: 17-May-90 11:10:05 {DSK}<usr>local>lde>lispcore>sources>SEdit-LISTS.;2

changes to: (IL:VARS IL:SEdit-LISTSCOMS)

previous date: 14-Jun-88 21:42:26 {DSK}<usr>local>lde>lispcore>sources>SEdit-LISTS.;1

Read Table: XCL

Package: SEdit

Format: XCCS

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

```
(IL:RPAQQ IL:SEdit-LISTSCOMS
  ((IL:PROP IL:FILETYPE IL:SEdit-LISTS)
   (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:SEdit-LISTS)
   (IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FILES IL:SEdit-DECLS)
    (IL:LOCALVARS . T))
   (IL:VARIABLES *FORMAT-ALIAS-DEPTH-LIMIT* *WRAP-PARENS* INTERNAL-WRAPPERS)
   (IL:VARS (LIST-PARSE-INFO ' (QUOTE PARSE--QUOTE IL:BQUOTE PARSE--QUOTE IL:\\, PARSE--QUOTE IL:\\,@
    PARSE--QUOTE IL:\\,. PARSE--QUOTE FUNCTION PARSE--QUOTE IL:*
    PARSE--COMMENT))
    (CLISP-INDENT-WORDS ' (IL:THEN IL:|then| IL:ELSE IL:|else| IL:OF IL:|of| IL:WITH IL:|with|
    IL:IN IL:|in| IL:INSTRING IL:|instring| IL:FROM IL:|from| IL:ON
    IL:|on| IL:TO IL:|to| IL:BY IL:|by| IL:OLD IL:|old| IL:INSIDE
    IL:|inside| IL:OUTOF IL:|outof|))
    (CLISP-PROGRAM-WORDS ' (IL:THEN IL:|then| IL:ELSE IL:|else| IL:DO IL:|do| IL:COLLECT
    IL:|collect| IL:JOIN IL:|join| IL:SUM IL:|sum| IL:COUNT
    IL:|count| IL:ALWAYS IL:|always| IL:NEVER IL:|never| IL:THEREIS
    IL:|thereis| IL:LARGEST IL:|largest| IL:SMALLEST IL:|smallest|)))
   (IL:FNS ASSIGN-FORMAT-CLISP ASSIGN-FORMAT-DOTLIST ASSIGN-FORMAT-LIST ASSIGN-FORMAT-QUOTE
    BACKSPACE-LIST BACKSPACE-QUOTE CFV-CLISP CFV-DOTLIST CFV-LIST CFV-QUOTE CLOSE-LIST
    COMPUTE-POINT-POSITION-LIST COPY-STRUCTURE-LIST COPY-STRUCTURE-QUOTE CREATE-NULL-LIST
    CREATE-QUOTED-GAP DELETE-LIST DELETE-QUOTE DOT-THIS-LIST GET-LIST-FORMAT INITIALIZE-LISTS
    INSERT-LIST INSERT-NULL-LIST INSERT-QUOTED-GAP LINEARIZE-CLISP LINEARIZE-DOTLIST
    LINEARIZE-LIST LINEARIZE-QUOTE NEXT-NODE-TYPE OUTPUT-CR-OR-SPACE
    PARENTHESE-CURRENT-SELECTION PARSE--LIST PARSE--LIST-INTERNAL PARSE--QUOTE REPLACE-LIST
    REPLACE-QUOTE SET-LIST-FORMAT SET-POINT-LIST SET-POINT-QUOTE SET-SELECTION-LIST
    SET-SELECTION-QUOTE STRINGIFY-LIST STRINGIFY-QUOTE SUBNODE-CHANGED-LIST SUBNODE-CHANGED-QUOTE
    UNDO-LIST-REPLACE UNDO-REPLACE-QUOTE)))

(IL:PUTPROPS IL:SEdit-LISTS IL:FILETYPE :COMPILE-FILE)

(IL:PUTPROPS IL:SEdit-LISTS IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE IL:SEdit
(:USE IL:LISP IL:XCL))))

(IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE

(IL:FILESLOAD IL:SEdit-DECLS)

(IL:DECLARE\ IL:DOEVAL@COMPILE IL:DONTCOPY

(IL:LOCALVARS . T)
)
)

(DEFGLOBALVAR *FORMAT-ALIAS-DEPTH-LIMIT* 10)

(DEFPARAMETER *WRAP-PARENS* NIL
  "Determines whether closing parens wrap to next line if they don't fit.")

(DEFGLOBALVAR INTERNAL-WRAPPERS

;;; this list pretty-prints badly because of itself. see parse--list-internal.

' (IL:BQUOTE IL:\\, IL:\\,@ IL:\\,.)

(IL:RPAQQ LIST-PARSE-INFO (QUOTE PARSE--QUOTE IL:BQUOTE PARSE--QUOTE IL:\\, PARSE--QUOTE IL:\\,@ PARSE--QUOTE
IL:\\,. PARSE--QUOTE FUNCTION PARSE--QUOTE IL:* PARSE--COMMENT))

(IL:RPAQQ CLISP-INDENT-WORDS (IL:THEN IL:|then| IL:ELSE IL:|else| IL:OF IL:|of| IL:WITH IL:|with| IL:IN IL:|in|
IL:INSTRING IL:|instring| IL:FROM IL:|from| IL:ON IL:|on| IL:TO IL:|to|
IL:BY IL:|by| IL:OLD IL:|old| IL:INSIDE IL:|inside| IL:OUTOF IL:|outof|))

(IL:RPAQQ CLISP-PROGRAM-WORDS (IL:THEN IL:|then| IL:ELSE IL:|else| IL:DO IL:|do| IL:COLLECT IL:|collect|
IL:JOIN IL:|join| IL:SUM IL:|sum| IL:COUNT IL:|count| IL:ALWAYS
IL:|always| IL:NEVER IL:|never| IL:THEREIS IL:|thereis| IL:LARGEST
IL:|largest| IL:SMALLEST IL:|smallest|))

(IL:DEFINEQ

(ASSIGN-FORMAT-CLISP
  (IL:LAMBDA (NODE CONTEXT)
    ; Edited 16-Jul-87 08:32 by DCB
```

;; in a clisp expression, the car is a clispword and determines the type of the clisp expression. for example, for would set the type to be FORWARD. in a clisp expression, each clisp word of the same type as the car should be set as a keyword, and all other subnodes should be set normally. (note that this way, "if" won't get set as a keyword if it appears as an atom directly in a for-loop list.)

;; note that we must keep the clisp type in the Unassigned field of the clisp list's node, since the clisp linearize method depends on it.

```
(LET* ((SUBNODES (CDR (IL:FETCh SUB-NODES IL:OF NODE)))
      (CLISP-TYPE (CAR (IL:GETPROP (IL:FETCh STRUCTURE IL:OF (CAR SUBNODES))
                                'IL:CLISPWORD))))
  (SET-FORMAT (CAR SUBNODES)
              CONTEXT :KEYWORD)
  (IL:FOR SUBNODE IL:IN (CDR SUBNODES)
    IL:DO (SET-FORMAT SUBNODE CONTEXT (IF (EQ CLISP-TYPE (CAR (IL:LISTP (IL:GETPROP (IL:FETCh STRUCTURE
                                                                                       IL:OF SUBNODE)
                                                                                       'IL:CLISPWORD))))
                                          :KEYWORD
                                          NIL))))))
```

**(ASSIGN-FORMAT-DOTLIST**

```
(IL:LAMBDA (NODE CONTEXT) ; Edited 7-Jul-87 12:51 by DCB
```

;; in a dotted list, all sublists should be set as data lists and other types should not be set specially.

```
(IL:FOR SUBNODE IL:IN (CDR (IL:FETCh SUB-NODES IL:OF NODE)) IL:DO (SET-FORMAT SUBNODE CONTEXT (GET-LIST-FORMAT
                                                                                       :DATA))))
```

**(ASSIGN-FORMAT-LIST**

```
(IL:LAMBDA (NODE CONTEXT FORMAT) ; Edited 1-Sep-87 18:41 by drc:
```

;; Determine this list's ListFormat, and propagate the appropriate formats to its subnodes

```
(WHEN (NOT (IL:TYPE? LIST-FORMAT FORMAT))
  ;; if we weren't given one, see if we recognize the CAR -- if not, use the default format
  (LET ((LIST-CAR (CAR (IL:FETCh STRUCTURE IL:OF NODE)))
        (IL:SETQ FORMAT (IF (NOT (IL:LITATOM LIST-CAR))
                            (GET-LIST-FORMAT :DEFAULT)
                            (OR (GET-LIST-FORMAT LIST-CAR)
                                (AND (IL:LISTP (IL:SETQ LIST-CAR (IL:GETPROP LIST-CAR 'IL:CLISPWORD)))
                                    (IL:MEMB (CAR LIST-CAR)
                                             '(IL:IFWORD IL:FORWORD IL:RECORDTRAN))
                                             (GET-LIST-FORMAT :CLISP))
                                (GET-LIST-FORMAT :DEFAULT)))))))
```

;; Stash the ListFormat for cvf.list and linearize.list

```
(IL:REPLACE UNASSIGNED IL:OF NODE IL:WITH FORMAT)
```

;; Non-standard ListFormats provide their own SetFormat method -- use it.

```
(COND ((IL:FETCh NON-STANDARD? IL:OF FORMAT)
      (FUNCALL (IL:FETCh SET-FORMAT-LIST IL:OF FORMAT)
              NODE CONTEXT))
  (T ;; Otherwise, we do the work
    (LET* ((FORMATS (IL:FETCh LIST-FORMATS IL:OF FORMAT))
          (LAST-FORMAT (CAR FORMATS))
          (SUBNODES (CDR (IL:FETCh SUB-NODES IL:OF NODE)))
          (LAST-SUBNODE SUBNODES))
      ;; Find the last non-comment subnode
      (IL:FOR P IL:ON SUBNODES IL:WHEN (NOT (EQ (IL:FETCh NODE-TYPE IL:OF (CAR P))
                                                TYPE-COMMENT))
        IL:DO (IL:SETQ LAST-SUBNODE P))
      (IL:WHILE SUBNODES IL:DO (LET* ((SUBNODE (CAR SUBNODES))
                                     (SUBFORMAT-NAME (AND (IL:NEQ (IL:FETCh NODE-TYPE IL:OF SUBNODE)
                                                                    TYPE-COMMENT)
                                                           (IF (AND (EQ SUBNODES LAST-SUBNODE)
                                                                (NULL (CDDR FORMATS)))
                                                               LAST-FORMAT
                                                               (CAR (IL:SETQ FORMATS
                                                                    (OR (CDR FORMATS)
                                                                        FORMATS)))))))
                                     (SET-FORMAT SUBNODE CONTEXT (CASE SUBFORMAT-NAME
                                                                    ((NIL :KEYWORD) SUBFORMAT-NAME)
                                                                    (:RECURSIVE FORMAT)
                                                                    (OTHERWISE (GET-LIST-FORMAT
                                                                    SUBFORMAT-NAME))))))
        ))))
      (IL:SETQ SUBNODES (CDR SUBNODES))))))
```

**(ASSIGN-FORMAT-QUOTE**

```
(IL:LAMBDA (NODE CONTEXT FORMAT) ; Edited 7-Jul-87 12:51 by DCB
```

;; assigns the format for a quoted subnode. Normal quotes assume the subnode is data, other types (e.g., backquote) assume the subnode is a form.



```

      (IL:FETCH INLINE-WIDTH IL:OF SUBNODE)))
    (IL:SETQ IWIDTH NIL))
  (WHEN (AND (NOT FIRST-SUBNODE)
             (EQ (IL:FETCH FORMAT IL:OF SUBNODE)
                 :KEYWORD))
        ;; indentable keywords are indented by the base indentation, except for the first keyword of the expression. other keywords are
        ;; only indented by the width of the left parenthesis
    (COND
      ((IL:MEMB (CDR (IL:GETPROP (IL:FETCH STRUCTURE IL:OF SUBNODE)
                                'IL:CLISPPWORD))
                CLISP-INDENT-WORDS)
        (IL:SETQ INDENT (IL:FETCH INDENT-BASE IL:OF ENVIRONMENT)))
      (T (IL:SETQ INDENT PAREN-WIDTH)
         (IL:SETQ IWIDTH NIL)))
    (IL:SETQ PWIDTH (IL:IMAX PWIDTH (IL:IPLUS (IL:FETCH PREFERRED-WIDTH IL:OF SUBNODE)
                                              INDENT)))
    (WHEN (EQ (IL:FETCH FORMAT IL:OF SUBNODE)
              :KEYWORD)
          ;; the subnodes following a keyword are indented by the keyword's indentation plus its width plus a blank
      (IL:SETQ INDENT (IL:IPLUS INDENT (IL:FETCH INLINE-WIDTH IL:OF SUBNODE)
                                    SPACE-WIDTH)))
    (IL:SETQ FIRST-SUBNODE NIL)
    IL:FINALLY (IL:REPLACE INLINE-WIDTH IL:OF X IL:WITH (AND IWIDTH (IL:ILESSP IWIDTH (IL:FETCH MAX-WIDTH
                                                                                               IL:OF ENVIRONMENT))
                                                         (IL:IPLUS IWIDTH PAREN-WIDTH)))
              (IL:REPLACE PREFERRED-WIDTH IL:OF X IL:WITH PWIDTH))))

```

**(CFV-DOTLIST**

```
(IL:LAMBDA (X ENVIRONMENT)
```

; Edited 7-Jul-87 12:52 by DCB

;;; compute the width estimates for a dotted list

```

(LET ((PAREN-WIDTH (IL:FETCH WIDTH IL:OF (IL:FETCH LPAREN-STRING IL:OF ENVIRONMENT)))
      (SPACE-WIDTH (IL:CHARWIDTH (IL:CHARCODE IL:SPACE)
                                  (IL:FETCH DEFAULT-FONT IL:OF ENVIRONMENT)))
      (SUBNODES (CDR (IL:FETCH SUB-NODES IL:OF X)))
      (NUMBER-OF-SUBNODES (CAR (IL:FETCH SUB-NODES IL:OF X))))
  (COND
    ((EQ 0 NUMBER-OF-SUBNODES)
     ;; empty lists are boring
     (IL:SETQ PAREN-WIDTH (IL:ITIMES PAREN-WIDTH 2))
     (IL:REPLACE INLINE-WIDTH IL:OF X IL:WITH PAREN-WIDTH)
     (IL:REPLACE PREFERRED-WIDTH IL:OF X IL:WITH PAREN-WIDTH))
    (T (LET ((WIDTH-OF-DOT (IF (EQ (IL:FETCH NODE-TYPE IL:OF X)
                                   TYPE-DOTLIST)
                              (IL:IPLUS (IL:FETCH WIDTH IL:OF (IL:FETCH DOT-STRING IL:OF ENVIRONMENT))
                                          SPACE-WIDTH)
                              0)))
        ;; a list can go inline if all of its subnodes can
        (IL:REPLACE INLINE-WIDTH IL:OF X IL:WITH (AND (IL:FOR SUBNODE IL:IN SUBNODES
                                                         IL:ALWAYS (IL:ATOM (IL:FETCH STRUCTURE
                                                                                               IL:OF SUBNODE)))
                                                       (IL:IPLUS PAREN-WIDTH WIDTH-OF-DOT
                                                            (IL:ITIMES (IL:SUB1 NUMBER-OF-SUBNODES)
                                                                SPACE-WIDTH)
                                                            (IL:FOR SUBNODE IL:IN SUBNODES
                                                                 IL:SUM (IL:FETCH INLINE-WIDTH
                                                                 IL:OF SUBNODE))
                                                            PAREN-WIDTH)))
          ;; forget the closing paren if it can't go inline, since the last line may be short
          (IL:REPLACE PREFERRED-WIDTH IL:OF X IL:WITH (IL:BIND (MAX IL:_ 0) IL:FOR SUBNODE IL:IN SUBNODES
                                                                IL:DO (IL:SETQ MAX (IL:IMAX MAX
                                                                                               (IL:FETCH
                                                                                               PREFERRED-WIDTH
                                                                                               IL:OF SUBNODE))))
                                                                IL:FINALLY (RETURN (IL:IPLUS MAX PAREN-WIDTH))))
          ))))

```

**(CFV-LIST**

```
(IL:LAMBDA (NODE ENVIRONMENT)
```

; Edited 31-Aug-87 16:06 by drc:

;;; Compute the format values of a list, driven by its ListFormat.

```

(LET
  ((INFO (IL:FETCH UNASSIGNED IL:OF NODE)))
  (COND
    ((IL:FETCH NON-STANDARD? IL:OF INFO)
     ;; Non-standard ListFormats specify their own CFV method
     (FUNCALL (IL:FETCH CFVLIST IL:OF INFO)

```



```

(CADR SEPR-INFO)
(CDDR SEPR-INFO)))
(SET-INDENT (IL:SETQ SET-INDENT? T)
(CDR SEPR-INFO))
(FROM-INDENT (IL:SETQ INDENT-BASE INDENT)
(CDR SEPR-INFO))
(BREAK (IL:SETQ BREAK? T)
(CDR SEPR-INFO))
(IL:SHOULDNT "Bad List Format"))))
(IL:SETQ X (IF BREAK?
(IL:IMIN (IL:IPLUS SEPR-INFO INDENT-BASE)
(IL:IPLUS X SPACE-WIDTH))
(IL:IPLUS X SPACE-WIDTH)))
(WHEN SET-INDENT? (IL:SETQ INDENT X))))
;; Now that we think we know where this subnode will start, check its effect on the overall width
(IL:SETQ PWIDTH (IL:IMAX PWIDTH (IL:IPLUS X (IL:FETCH PREFERRED-WIDTH IL:OF SUBNODE))))
(LET ((SUB-IWIDTH (IL:FETCH INLINE-WIDTH IL:OF SUBNODE))
(SUB-PWIDTH (IL:FETCH PREFERRED-WIDTH IL:OF SUBNODE)))
(COND
(SUB-IWIDTH (IL:SETQ X (IL:IPLUS X SUB-IWIDTH))
(WHEN IWIDTH
(IL:SETQ IWIDTH (IL:IPLUS IWIDTH SUB-IWIDTH)))
(T (IL:SETQ IWIDTH NIL))))
(IL:SETQ PREV-TYPE NEXT-TYPE)))
(IL:SETQ SUBNODES (CDR SUBNODES)))
(IL:REPLACE INLINE-WIDTH IL:OF NODE IL:WITH (AND IWIDTH (IL:ILESSP IWIDTH (IL:FETCH MAX-WIDTH
IL:OF ENVIRONMENT))
IWIDTH))
(IL:REPLACE PREFERRED-WIDTH IL:OF NODE IL:WITH (IL:IPLUS PWIDTH TWO-PARENS))))))

```

**(CFV-QUOTE**

```
(IL:LAMBDA (X ENVIRONMENT FORMAT)
```

; Edited 7-Jul-87 12:53 by DCB

;;; compute the width estimates for a quoted structure. very straightforward

```

(LET ((QUOTE-WIDTH (IL:FETCH WIDTH IL:OF (IL:FETCH UNASSIGNED IL:OF X))
(SUBNODE (CADR (IL:FETCH SUB-NODES IL:OF X))))
(IL:REPLACE INLINE-WIDTH IL:OF X IL:WITH (AND (IL:FETCH INLINE-WIDTH IL:OF SUBNODE)
(IL:IPLUS QUOTE-WIDTH (IL:FETCH INLINE-WIDTH IL:OF SUBNODE)
)))
(IL:REPLACE PREFERRED-WIDTH IL:OF X IL:WITH (IL:IPLUS QUOTE-WIDTH (IL:FETCH PREFERRED-WIDTH IL:OF SUBNODE)
))))))

```

**(CLOSE-LIST**

```
(IL:LAMBDA (CONTEXT CHARCODE)
```

; Edited 22-Dec-87 09:03 by DCB

;;; implements the close paren command (skips to the end of this list)

```

(LET ((PNODE))
(WHEN (IL:FMEMB (TYPE-OF-INPUT CONTEXT)
' (ATOM STRUCTURE))
(CLOSE-OPEN-NODE CONTEXT)
(IL:BIND NODE IL:_ (IL:FETCH POINT-NODE IL:OF (IL:FETCH CARET-POINT IL:OF CONTEXT))
IL:FIRST (WHEN (TYPEP NODE 'EDIT-SELECTION)
(IL:SETQ NODE (IL:FETCH SELECT-NODE IL:OF NODE)))
IL:WHILE (AND NODE (NOT (IL:MEMB (IL:FETCH NAME IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
' (LIST DOTLIST CLISP))))
IL:DO
;; climb up looking for the nearest enclosing list-type structure
(IL:SETQ NODE (IL:FETCH SUPER-NODE IL:OF NODE))
IL:FINALLY (COND
(NODE ;; ask the list to put this point after itself
(SET-POINT (IL:FETCH CARET-POINT IL:OF CONTEXT)
CONTEXT NODE NIL T)
(SELECT-NODE CONTEXT NODE))
(T ;; we're not in a list (pretty unusual) so there's no obvious place to put the point
(SET-POINT-NOWHERE (IL:FETCH CARET-POINT IL:OF CONTEXT))
(FORMAT (GET-PROMPT-WINDOW CONTEXT)
"~%No enclosing list."))))
;; must return non-NIL if command executed
T))))

```

**(COMPUTE-POINT-POSITION-LIST**

```
(IL:LAMBDA (POINT)
```

; Edited 17-Nov-87 11:29 by DCB

;;; implement the ComputePointPosition method for a list, form, clisp, lambda, etc.

```

(LET ((NODE (IL:FETCH POINT-NODE IL:OF POINT))
SUBNODE ITEM)
(COND

```

```

(EQ 0 (IL:FETCH POINT-INDEX IL:OF POINT))
;; before the first element -- right after the opening paren, which we assume is the first item in the linear form
(IL:REPLACE POINT-X IL:OF POINT IL:WITH (IL:IPLUS (IL:FETCH START-X IL:OF NODE)
                                             (IL:FETCH WIDTH IL:OF (CAR (IL:FETCH LINEAR-FORM
                                                                    IL:OF NODE))))))
(IL:REPLACE POINT-LINE IL:OF POINT IL:WITH (IL:FETCH FIRST-LINE IL:OF NODE)))
(T
  ;; find the subnode it will follow
  (IL:SETQ SUBNODE (SUBNODE (IL:FETCH POINT-INDEX IL:OF POINT)
                            NODE))
  (COND
    ((EQ (IL:FETCH NODE-TYPE IL:OF SUBNODE)
         TYPE-COMMENT)
     (IL:REPLACE POINT-LINE IL:OF POINT IL:WITH (CAR (IL:FETCH NEXT-LINE
                                                         IL:OF (IL:FETCH LAST-LINE IL:OF SUBNODE))))))
    ((IL:REPLACE POINT-X IL:OF POINT IL:WITH (IL:IMAX (IL:IDIFFERENCE (IL:FETCH INDENT
                                                                        IL:OF (IL:FETCH POINT-LINE
                                                                IL:OF POINT))
                                                                6)
                                                         (IL:FETCH START-X IL:OF NODE))))))
    (T (IL:REPLACE POINT-LINE IL:OF POINT IL:WITH (IL:FETCH LAST-LINE IL:OF SUBNODE))
      (IL:SETQ ITEM (CADR (IL:FETCH LINEAR-THREAD IL:OF SUBNODE)))
      (IL:REPLACE POINT-X IL:OF POINT IL:WITH (IL:IPLUS (IL:FETCH START-X IL:OF SUBNODE)
                                                         (IL:FETCH ACTUAL-LENGTH IL:OF SUBNODE))
        (COND
          ((IL:SMALLP ITEM)
           ;; it's followed by space -- put the caret in the middle
           (IL:IMIN (IL:HALF ITEM)
                    6))
          ((IL:TYPE? LINE-START ITEM)
           ;; it's the last thing on the line -- put the caret a little
           ;; ways after it
           6))
        (T
         ;; it's followed by something else -- presumably the close paren -- so put
         ;; the caret immediately after it
         0)))))))))

```

**(COPY-STRUCTURE-LIST**

(IL:LAMBDA (NODE) ; Edited 17-Nov-87 11:29 by DCB

;; the CopyStructure method for lists, forms, clisp expressions, etc.

```

(IL:REPLACE STRUCTURE IL:OF NODE IL:WITH (IL:FOR SUBNODE IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE))
                                           IL:COLLECT (IL:FETCH STRUCTURE IL:OF SUBNODE)))
(WHEN (EQ (IL:FETCH NODE-TYPE IL:OF NODE)
          TYPE-DOTLIST)
  (LET ((TAIL (TAIL (IL:NTH (IL:FETCH STRUCTURE IL:OF NODE)
                           (IL:SUB1 (CAR (IL:FETCH SUB-NODES IL:OF NODE))))))
        (RPLACD TAIL (CADR TAIL))))))

```

**(COPY-STRUCTURE-QUOTE**

(IL:LAMBDA (NODE) ; Edited 17-Nov-87 11:29 by DCB

;; the CopyStructure method for quoted structures

```

(IL:REPLACE STRUCTURE IL:OF NODE IL:WITH (LIST (CAR (IL:FETCH STRUCTURE IL:OF NODE))
                                                (IL:FETCH STRUCTURE IL:OF (SUBNODE 1 NODE))))))

```

**(CREATE-NULL-LIST**

(IL:LAMBDA (CONTEXT) ; Edited 6-Apr-88 16:27 by woz

;;; creates a new node describing an empty list

```

(LET* ((WIDTH (IL:ITIMES 2 (IL:CHARWIDTH (IL:CHARCODE IL:\ ()
                                           (IL:|fetch| DEFAULT-FONT IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT))))))
       (NODE (IL:|create| EDIT-NODE
                        NODE-TYPE IL:_ TYPE-LIST
                        STRUCTURE IL:_ NIL
                        SUB-NODES IL:_ (LIST 0)
                        INLINE-WIDTH IL:_ WIDTH
                        PREFERRED-WIDTH IL:_ WIDTH)))
  (IL:|replace| LINEAR-FORM IL:|of| NODE IL:|with| (CREATE-WEAK-LINK NODE)
               NODE))

```

**(CREATE-QUOTED-GAP**

(IL:LAMBDA (GAP CONTEXT QUOTE-TYPE) ; Edited 6-Apr-88 16:28 by woz

;;; cons a quoted gap, and the node to represent it

```

(LET* ((GAP-NODE (CREATE-GAP-NODE GAP))

```

```
(QUOTE-NODE (IL:create| EDIT-NODE
              NODE-TYPE IL:_ TYPE-QUOTE
              STRUCTURE IL:_ (LIST (QUOTE-WRAPPER QUOTE-TYPE)
                                   GAP)
              SUB-NODES IL:_ (LIST 1 GAP-NODE)
              UNASSIGNED IL:_ (IL:LISTGET (IL:fetch| QUOTE-STRING IL:of| (IL:fetch| ENVIRONMENT
                                   IL:of| CONTEXT))
                                QUOTE-TYPE)))
(IL:replace| SUPER-NODE IL:of| GAP-NODE IL:with| QUOTE-NODE)
(IL:replace| SUB-NODE-INDEX IL:of| GAP-NODE IL:with| 1)
(IL:replace| LINEAR-FORM IL:of| QUOTE-NODE IL:with| (CREATE-WEAK-LINK QUOTE-NODE))
(NOTE-CHANGE QUOTE-NODE CONTEXT)
(QUOTE-NODE))
```

**(DELETE-LIST**

```
(IL:LAMBDA (NODE CONTEXT START END SET-POINT?) ; Edited 17-Nov-87 11:29 by DCB
  ;; the Delete method for lists and related animals
  (WHEN (IL:TYPE? EDIT-NODE START)
    (IL:SETQ START (IL:FETCH SUB-NODE-INDEX IL:OF START)))
  (REPLACE-LIST NODE CONTEXT START (OR END START)
    NIL SET-POINT?)
  T))
```

**(DELETE-QUOTE**

```
(IL:LAMBDA (NODE CONTEXT START END SET-POINT?) ; Edited 7-Jul-87 12:53 by DCB
```

;;; replace node to be delete with a gap. the backspace method will let a quoted gap be deleted.

```
(IF (OR (IL:NEQ (OR (IL:SMALLP START)
                    (IL:FETCH SUB-NODE-INDEX IL:OF START))
        1)
      (AND END (IL:NEQ END 1)))
  (IL:SHOULDNT "bad index in delete.quote")
  (LET ((SUBNODE (SUBNODE 1 NODE))
        (GAP-NODE (CREATE-GAP-NODE BASIC-GAP)))
    (REPLACE-NODE CONTEXT SUBNODE GAP-NODE)
    (WHEN SET-POINT?
      (SET-SELECTION-ME (IL:FETCH SELECTION IL:OF CONTEXT)
                        CONTEXT GAP-NODE)
      (PENDING-DELETE SET-POINT? (IL:FETCH SELECTION IL:OF CONTEXT))))
  T)))
```

**(DOT-THIS-LIST**

```
(IL:LAMBDA (CONTEXT) ; Edited 7-Jul-87 12:53 by DCB
```

;;; implements the dot command: make this a dotted list

```
(LET* ((POINT (IL:FETCH CARET-POINT IL:OF CONTEXT))
       (NODE (IL:FETCH POINT-NODE IL:OF POINT))
       (INDEX (IL:FETCH POINT-INDEX IL:OF POINT))
       (NUM-SUBNODES (CAR (IL:FETCH SUB-NODES IL:OF NODE)))
       (GAP-NODE)
       (COND
        ((AND (IL:IGREATERP INDEX 0)
              (IL:IGEQ INDEX (IL:SUB1 NUM-SUBNODES)))
         (WHEN (EQ INDEX NUM-SUBNODES) ; at end of list. add dotted gap
           (IL:SETQ GAP-NODE (CREATE-GAP-NODE BASIC-GAP))
           (INSERT POINT CONTEXT GAP-NODE)
           (SELECT-SEGMENT (IL:FETCH SELECTION IL:OF CONTEXT)
                           CONTEXT NODE GAP-NODE GAP-NODE)
           (PENDING-DELETE POINT (IL:FETCH SELECTION IL:OF CONTEXT)))
         ; just dot contents of this list
         (LET ((TAIL (IL:NTH (IL:FETCH STRUCTURE IL:OF NODE)
                             INDEX)))
              (RPLACD TAIL (CADR TAIL)))
           (IL:REPLACE NODE-TYPE IL:OF NODE IL:WITH TYPE-DOTLIST)
           (NOTE-CHANGE NODE CONTEXT)
           (WHEN (IL:NEQ INDEX NUM-SUBNODES) ; if dotted existing list, set point before dot
             (SET-SELECTION-NOWHERE (IL:FETCH SELECTION IL:OF CONTEXT))
             (SET-POINT POINT CONTEXT NODE INDEX T (SUBNODE INDEX NODE)
                          'STRUCTURE T)))
         T)
        ;; waste selection to avoid pending delete inconsistency
        (SET-SELECTION-NOWHERE (IL:FETCH SELECTION IL:OF CONTEXT))))))
```

**(GET-LIST-FORMAT**

```
(IL:LAMBDA (FN) ; Edited 1-Sep-87 18:45 by drc:
```

;;; return the internal list format for forms whose CAR is FN, or NIL.

;;; we loop down aliases to \*FORMAT-ALIAS-DEPTH-LIMIT\*.

```
(DO ((FORMAT (GETHASH FN LIST-FORMATS-TABLE)
            (GETHASH FORMAT LIST-FORMATS-TABLE))
```



```

(DEPTH 0 (1+ DEPTH))
(= DEPTH *FORMAT-ALIAS-DEPTH-LIMIT*)
(CERROR "forget ~S's list format" "aliases for ~S too deep (possibly circular)" FN)
(SET-LIST-FORMAT FN 'NIL))
(ETYPESCASE FORMAT
 (NULL (RETURN 'NIL))
 (LIST-FORMAT (RETURN FORMAT))
 (SYMBOL ))))

```

**(INITIALIZE-LISTS**

```

(IL:LAMBDA NIL ; Edited 7-Jul-87 12:53 by DCB
 (IL:SETQ TYPES (LIST* (IL:SETQ TYPE-LIST (IL:CREATE EDIT-NODE-TYPE
 NAME IL:_ 'LIST
 ASSIGN-FORMAT IL:_ 'ASSIGN-FORMAT-LIST
 COMPUTE-FORMAT-VALUES IL:_ 'CFV-LIST
 LINEARIZE IL:_ 'LINEARIZE-LIST
 SUB-NODE-CHANGED IL:_ 'SUBNODE-CHANGED-LIST
 COMPUTE-POINT-POSITION IL:_ 'COMPUTE-POINT-POSITION-LIST
 COMPUTE-SELECTION-POSITION IL:_
 'COMPUTE-SELECTION-POSITION-DEFAULT
 SET-POINT IL:_ 'SET-POINT-LIST
 SET-SELECTION IL:_ 'SET-SELECTION-LIST
 GROW-SELECTION IL:_ 'GROW-SELECTION-DEFAULT
 SELECT-SEGMENT IL:_ 'SELECT-SEGMENT-DEFAULT
 INSERT IL:_ 'INSERT-LIST
 DELETE IL:_ 'DELETE-LIST
 COPY-STRUCTURE IL:_ 'COPY-STRUCTURE-LIST
 COPY-SELECTION IL:_ 'COPY-SELECTION-DEFAULT
 STRINGIFY IL:_ 'STRINGIFY-LIST
 BACK-SPACE IL:_ 'BACKSPACE-LIST))
 (IL:SETQ TYPE-DOTLIST (IL:CREATE EDIT-NODE-TYPE IL:USING TYPE-LIST NAME IL:_
 'DOTLIST ASSIGN-FORMAT IL:_
 'ASSIGN-FORMAT-DOTLIST
 COMPUTE-FORMAT-VALUES IL:_
 'CFV-DOTLIST LINEARIZE IL:_
 'LINEARIZE-DOTLIST))
 (IL:SETQ TYPE-QUOTE (IL:CREATE EDIT-NODE-TYPE IL:USING TYPE-ROOT NAME IL:_ 'QUOTE
 ASSIGN-FORMAT IL:_
 'ASSIGN-FORMAT-QUOTE
 COMPUTE-FORMAT-VALUES IL:_
 'CFV-QUOTE LINEARIZE IL:_
 'LINEARIZE-QUOTE SUB-NODE-CHANGED
 IL:_ 'SUBNODE-CHANGED-QUOTE
 SET-POINT IL:_ 'SET-POINT-QUOTE
 SET-SELECTION IL:_
 'SET-SELECTION-QUOTE
 GROW-SELECTION IL:_
 'GROW-SELECTION-DEFAULT INSERT
 IL:_ 'REPLACE-QUOTE DELETE IL:_
 'DELETE-QUOTE COPY-STRUCTURE IL:_
 'COPY-STRUCTURE-QUOTE
 COPY-SELECTION IL:_
 'COPY-SELECTION-DEFAULT STRINGIFY
 IL:_ 'STRINGIFY-QUOTE BACK-SPACE
 IL:_ 'BACKSPACE-QUOTE))
 TYPES))
 (RESET-FORMATS)))

```

**(INSERT-LIST**

```

(IL:LAMBDA (NODE CONTEXT WHERE SUBNODES POINT) ; Edited 17-Jul-87 10:04 by DCB

```

;;; the Insert method for lists and related animals

```

(LET (START END)
 (COND
 ((IL:TYPE? EDIT-SELECTION WHERE)
 (IL:SETQ START (IL:FETCH SELECT-START IL:OF WHERE))
 (IL:SETQ END (OR (IL:FETCH SELECT-END IL:OF WHERE)
 START)))
 ((IL:TYPE? EDIT-POINT WHERE)
 (IL:SETQ END (IL:FETCH POINT-INDEX IL:OF WHERE))
 (IL:SETQ START (IL:ADD1 END)))
 (T (IL:SETQ START (IL:FETCH SUB-NODE-INDEX IL:OF WHERE))
 (IL:SETQ END START)))
 (REPLACE-LIST NODE CONTEXT START END SUBNODES POINT))))

```

**(INSERT-NULL-LIST**

```

(IL:LAMBDA (CONTEXT) ; Edited 17-Nov-87 11:30 by DCB

```

;;; implements the left paren command: insert an empty list

```

(WHEN (IL:FMEMB (TYPE-OF-INPUT CONTEXT)
 ' (ATOM STRUCTURE))
 (LET ((POINT (IL:FETCH CARET-POINT IL:OF CONTEXT))

```

```

NEW-LIST)
(INSERT POINT CONTEXT (LIST (IL:SETQ NEW-LIST (CREATE-NULL-LIST CONTEXT))))
(WHEN (NOT (DEAD-NODE? NEW-LIST))
  ((IL:REPLACE POINT-NODE IL:OF POINT IL:WITH NEW-LIST)
  (IL:REPLACE POINT-INDEX IL:OF POINT IL:WITH 0)
  (IL:REPLACE POINT-TYPE IL:OF POINT IL:WITH 'STRUCTURE)
  (SET-SELECTION-NOWHERE (IL:FETCH SELECTION IL:OF CONTEXT))))
;; must return non-NIL if command executed
T)))

```

(INSERT-QUOTED-GAP

```

(IL:LAMBDA (CONTEXT CHARCODE QUOTE-TYPE) ; Edited 7-Jul-87 12:53 by DCB
; implements the ' command: insert a quoted gap
(WHEN (EQ (TYPE-OF-INPUT CONTEXT)
  'STRUCTURE)
  (LET ((SELECTION (IL:FETCH SELECTION IL:OF CONTEXT))
        (POINT (IL:FETCH CARET-POINT IL:OF CONTEXT))
        NEW-QUOTE GAP)
    (IL:SETQ NEW-QUOTE (CREATE-QUOTED-GAP BASIC-GAP CONTEXT QUOTE-TYPE))
    (IL:SETQ GAP (SUBNODE 1 NEW-QUOTE)) ; we get our hands on the gap node now, to handle the case
; where the insert reparses the new.quote
(INSERT (IL:FETCH CARET-POINT IL:OF CONTEXT)
  CONTEXT
  (LIST NEW-QUOTE))
(WHEN (NOT (DEAD-NODE? NEW-QUOTE))
  (SET-SELECTION-ME SELECTION CONTEXT GAP)
  (PENDING-DELETE POINT SELECTION)) ; must return non-NIL if command executed
T)))

```

(LINEARIZE-CLISP

```

(IL:LAMBDA (NODE CONTEXT INDEX) ; Edited 11-Apr-88 15:45 by woz

```

;; the Linearize method for clisp expressions. the variable ok keeps track of our state: (NIL: next item starts a new line) (T: next item stays on this line)  
;; (check: next item goes on this line if it fits) (atom: next item goes on this line if it fits and is an atom)

;; the formatting rules are that (1) keywords not on clisp.indent.words always start new lines (2) always start a new line after anything non-atomic (3)  
;; non-atomic things can only follow keywords on the same line (4) clisp.indent.words can go on the same line as the preceding material if they're the  
;; last thing in the expression or followed by another keyword or by something that will fit inline on the same line (5) if clisp.indent.words start a new line  
;; they are indented by the minimum indentation (6) if anything else starts a new line it is indented by the width of the most recent keyword to start a line,  
;; plus one blank

;; at present, if keywords always start new lines. this could be improved with a little more smarts

```

(IL:|bind| INDENT COMMENT-START-X COMMENT-INDENT COMMENT? PROGRAM-WORD? (KEYWORD? IL:_ T)
(SECOND-SUBNODE IL:_ T)
(OK IL:_ T)
(SPACE-WIDTH IL:_ (IL:|fetch| SPACE-WIDTH IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT)))
(MIN-INDENT IL:_ (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)
  (IL:|fetch| INDENT-BASE IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT))))
(PAREN-WIDTH IL:_ (IL:|fetch| WIDTH IL:|of| (IL:|fetch| LPAREN-STRING IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT)
  )))
(COULD-INLINE? IL:_ (AND (IL:|fetch| INLINE-WIDTH IL:|of| NODE)
  (IL:ILEQ (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)
    (IL:|fetch| INLINE-WIDTH IL:|of| NODE))
  (IL:|fetch| RIGHT-MARGIN IL:|of| NODE))))
(IF? IL:_ (IL:MEMB (CAR (IL:|fetch| STRUCTURE IL:|of| NODE))
  ' (IL:IF IL:|if|)))
IL:|first| (COND
  (INDEX (IL:SETQ INDEX (AND (IL:NEQ INDEX 1)
    (IL:SUB1 INDEX))))
  (T ;; start with an open paren and the first subnode (which should be a keyword) since system won't recognize clisp if first
  ;; subnode is comment, don't have to handle that case here. it will be formatted as a form.
  (OUTPUT-CONSTANT-STRING CONTEXT (IL:|fetch| LPAREN-STRING IL:|of| (IL:|fetch| ENVIRONMENT
    IL:|of| CONTEXT)))
  (LINEARIZE (CADR (IL:|fetch| SUB-NODES IL:|of| NODE))
    CONTEXT))
  ;; set indentation to one blank after the end of the keyword
  (IL:SETQ INDENT (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)
    PAREN-WIDTH
    (IL:|fetch| INLINE-WIDTH IL:|of| (CADR (IL:|fetch| SUB-NODES IL:|of| NODE))
    SPACE-WIDTH))
  (SET-COMMENT-POSITIONS COMMENT-START-X COMMENT-INDENT INDENT PAREN-WIDTH NODE CONTEXT))
IL:|for| SUBNODE IL:|in| (CDDR (IL:|fetch| SUB-NODES IL:|of| NODE))
IL:|do| (COND
  (INDEX ;; we don't actually linearize this subnode, but need to update our state as if we had
  (IL:SETQ INDEX (AND (IL:NEQ INDEX 1)
    (IL:SUB1 INDEX)))
  (COND
    ((IL:SETQ COMMENT? (EQ (IL:|fetch| NODE-TYPE IL:|of| SUBNODE)
      TYPE-COMMENT))
    ;; this is a comment, so the next guy must start a new line. if following the first keyword, change indent to min.indent

```

```

(IL:SETQ OK NIL)
(WHEN SECOND-SUBNODE (IL:SETQ INDENT MIN-INDENT)))
((IL:SETQ KEYWORD? (EQ (IL:|fetch| FORMAT IL:|of| SUBNODE)
                       :KEYWORD)))

;; this is a keyword. is it the first thing on this line?
(COND
  ((LET ((ITEM (CADR (IL:MEMB (IL:|fetch| LAST-LINE IL:|of| SUBNODE)
                             (IL:|fetch| LINEAR-FORM IL:|of| NODE))))))
    (AND (IL:|type?| WEAK-LINK ITEM)
         (EQ SUBNODE (IL:|fetch| DESTINATION IL:|of| ITEM))))
    ;; the test for this branch used to be:
    ;; (eq subnode (cadr (il:|fetch| last-line-linear il:|of| subnode)))
    ;; yep. set the indentation to be one blank after the end of it
    (IL:SETQ INDENT (IL:IPLUS (IL:|fetch| START-X IL:|of| SUBNODE)
                              (IL:|fetch| INLINE-WIDTH IL:|of| SUBNODE)
                              SPACE-WIDTH))
    ;; and the next thing goes on this line
    (IL:SETQ OK T))
  (T ;; the next thing goes on this line if it fits
    (IL:SETQ OK 'CHECK)))
  (T ;; the next thing can go on this line if i'm atomic, and it's atomic too
    (IL:SETQ OK (AND (IL:ATOM (IL:|fetch| STRUCTURE IL:|of| SUBNODE))
                    'ATOM))))
(T ;; we really are linearizing this subnode
  (COND
    ((IL:SETQ COMMENT? (EQ (IL:|fetch| NODE-TYPE IL:|of| SUBNODE)
                          TYPE-COMMENT))
     (IL:SETQ COMMENT? (SELECT-COMMENT-INDENT (IL:|fetch| UNASSIGNED IL:|of| SUBNODE)
                                             COMMENT-INDENT INDENT (IL:|fetch| START-X
                                                                    IL:|of| (IL:|fetch| ROOT IL:|of| CONTEXT))))))
    (IF (OR (NOT OK)
            (IL:IGREATERP (IL:|fetch| CURRENT-X IL:|of| CONTEXT)
                          (IL:SELECTQ (IL:|fetch| UNASSIGNED IL:|of| SUBNODE)
                                       (1 COMMENT-START-X)
                                       (2 (IL:IDIFFERENCE INDENT SPACE-WIDTH)
                                          0))))
        (OUTPUT-CR CONTEXT COMMENT?)
        (OUTPUT-SPACE CONTEXT (IL:IDIFFERENCE COMMENT? (IL:|fetch| CURRENT-X IL:|of| CONTEXT))))
     (IL:SETQ OK NIL)
     (WHEN SECOND-SUBNODE (IL:SETQ INDENT MIN-INDENT)))
    ((IL:SETQ KEYWORD? (EQ (IL:|fetch| FORMAT IL:|of| SUBNODE)
                          :KEYWORD)))
     ;; we've got a keyword
     (IL:SETQ PROGRAM-WORD? (IL:FMEMB (CDR (IL:GETPROP (IL:|fetch| STRUCTURE IL:|of| SUBNODE)
                                                       'IL:CLISPWORD))
                                       CLISP-PROGRAM-WORDS))
      (COND
        ((IL:FMEMB (CDR (IL:GETPROP (IL:|fetch| STRUCTURE IL:|of| SUBNODE)
                                    'IL:CLISPWORD))
                   CLISP-INDENT-WORDS)
         ;; perhaps it can go on this line
         (COND
           ((AND OK (OR COULD-INLINE? (NOT IF?))
                 (IL:ILEQ (IL:IPLUS (IL:|fetch| CURRENT-X IL:|of| CONTEXT)
                                    SPACE-WIDTH
                                    (IL:|fetch| INLINE-WIDTH IL:|of| SUBNODE)
                                    (IF (AND (CDR IL:$LST1)
                                            (IL:NEQ (IL:|fetch| FORMAT IL:|of| (CADR IL:$LST1))
                                                    :KEYWORD))
                                        (IL:IPLUS SPACE-WIDTH (OR (IL:|fetch| INLINE-WIDTH
                                                                    IL:|of| (CADR IL:$LST1))
                                                                    (IL:|fetch| RIGHT-MARGIN
                                                                    IL:|of| NODE))))
                                    0))
                (IL:|fetch| RIGHT-MARGIN IL:|of| NODE))))
            ;; it'll go on this line
            (OUTPUT-SPACE CONTEXT SPACE-WIDTH)
            (IL:SETQ OK 'CHECK))
           (T ;; new line, indented by minimum indentation
              (OUTPUT-CR CONTEXT MIN-INDENT)
              (IL:SETQ INDENT (IL:IPLUS MIN-INDENT (IL:|fetch| INLINE-WIDTH IL:|of| SUBNODE)
                                        SPACE-WIDTH))
              (IL:SETQ OK T))))
        (T ;; new line, no indentation
          (OUTPUT-CR CONTEXT (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)

```

```

(PAREN-WIDTH))
(IL:SETQ INDENT (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)
                          PAREN-WIDTH
                          (IL:|fetch| INLINE-WIDTH IL:|of| SUBNODE)
                          SPACE-WIDTH))
(T (IF (OR (EQ OK T)))
      (AND OK (IL:|fetch| INLINE-WIDTH IL:|of| SUBNODE)
            (IL:ILEQ (IL:IPLUS (IL:|fetch| CURRENT-X IL:|of| CONTEXT)
                              SPACE-WIDTH
                              (IL:|fetch| INLINE-WIDTH IL:|of| SUBNODE))
                    (IL:|fetch| RIGHT-MARGIN IL:|of| NODE))
            (OR (EQ OK 'CHECK)
                (IL:ATOM (IL:|fetch| STRUCTURE IL:|of| SUBNODE))))))
(OUTPUT-SPACE CONTEXT SPACE-WIDTH)
(OUTPUT-CR CONTEXT INDENT))
(IL:SETQ OK 'ATOM))
(LINEARIZE SUBNODE CONTEXT)
(WHEN (AND (EQ OK 'ATOM)
           (NOT (IL:|fetch| INLINE? IL:|of| SUBNODE))))
      (IL:SETQ OK NIL)))
(IL:SETQ SECOND-SUBNODE NIL)
IL:|finally| (WHEN COMMENT?
             (OUTPUT-CR CONTEXT (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)
                                         PAREN-WIDTH)))
(WHEN INDEX (IL:SHOULDN'T "linearize index out of range"))
(OUTPUT-CONSTANT-STRING CONTEXT (IL:|fetch| RPAREN-STRING IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT))))))

```

(LINEARIZE-DOTLIST

(IL:LAMBDA (NODE CONTEXT INDEX)

; Edited 7-Jul-87 12:54 by DCB

;;; the Linearize method for dotted lists. nothing is indented, non-atomic things go on separate lines, and we put as many atoms on a line as we can fit.
;;; the last element of a dotted list is preceded by a dot.

```

(WHEN (NOT INDEX)
      (OUTPUT-CONSTANT-STRING CONTEXT (IL:FETCH LPAREN-STRING IL:OF (IL:FETCH ENVIRONMENT IL:OF CONTEXT))))
(WHEN (CDR (IL:FETCH SUB-NODES IL:OF NODE))
      (IL:BIND (FIRST-TIME? IL:_ T)
              (SPACE-WIDTH IL:_ (IL:FETCH SPACE-WIDTH IL:OF (IL:FETCH ENVIRONMENT IL:OF CONTEXT)))
              (PAREN-WIDTH IL:_ (IL:FETCH WIDTH IL:OF (IL:FETCH LPAREN-STRING IL:OF (IL:FETCH ENVIRONMENT IL:OF CONTEXT))))))
      THIS-LINE? NEEDS-DOT? COMMENT? COMMENT-START-X COMMENT-INDENT
      IL:FIRST (SET-COMMENT-POSITIONS COMMENT-START-X COMMENT-INDENT PAREN-WIDTH PAREN-WIDTH NODE CONTEXT)
      IL:FOR SUBNODE IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE))
      IL:DO (IL:SETQ COMMENT? (EQ (IL:FETCH NODE-TYPE IL:OF SUBNODE)
                                TYPE-COMMENT))
            (COND
              (INDEX (IL:SETQ INDEX (AND (IL:NEQ INDEX 1)
                                         (IL:SUB1 INDEX)))
                    (WHEN COMMENT?
                      (OUTPUT-CR CONTEXT (IL:IPLUS PAREN-WIDTH (IL:FETCH START-X IL:OF NODE))))))
              (T (IL:SETQ NEEDS-DOT? (AND (EQ (IL:FETCH NODE-TYPE IL:OF NODE)
                                             TYPE-DOTLIST)
                                         (NULL (CDR IL:$LST1))
                                         (IL:IPLUS SPACE-WIDTH (IL:FETCH WIDTH
                                                                IL:OF (IL:FETCH DOT-STRING
                                                                IL:OF (IL:FETCH ENVIRONMENT
                                                                IL:OF CONTEXT))))))
                    (COND
                      (COMMENT? (IL:SETQ FIRST-TIME? NIL)
                               (IF (OR (IL:NEQ (IL:FETCH UNASSIGNED IL:OF SUBNODE)
                                             1)
                                      (IL:IGREATERP (IL:FETCH CURRENT-X IL:OF CONTEXT)
                                                    COMMENT-START-X))
                                   (OUTPUT-CR CONTEXT (SELECT-COMMENT-INDENT (IL:FETCH UNASSIGNED IL:OF SUBNODE)
                                                                              COMMENT-INDENT
                                                                              (IL:IPLUS PAREN-WIDTH (IL:FETCH START-X
                                                                              IL:OF NODE))
                                                                              (IL:FETCH START-X IL:OF (IL:FETCH ROOT IL:OF CONTEXT)
                                                                              ))))
                                   (OUTPUT-SPACE CONTEXT (IL:IDIFFERENCE COMMENT-INDENT (IL:FETCH CURRENT-X
                                                                              IL:OF CONTEXT))))))
                    ((AND FIRST-TIME? (NOT COMMENT?)) ; first time through, if not a comment, then i'm already in the right
                    ; place for the first subnode
                    (IL:SETQ FIRST-TIME? NIL))
                    ((AND THIS-LINE? (NULL (CDR (IL:FETCH SUB-NODES IL:OF SUBNODE)))
                      (IL:LEQ (IL:IPLUS (IL:FETCH CURRENT-X IL:OF CONTEXT)
                                       SPACE-WIDTH
                                       (IL:FETCH INLINE-WIDTH SUBNODE)
                                       (OR NEEDS-DOT? 0))
                              (IL:FETCH RIGHT-MARGIN IL:OF NODE)))
                    ; the last node said i could go on this line, i'm atomic so i can go
                    ; on this line, and i will fit
                    (OUTPUT-SPACE CONTEXT SPACE-WIDTH))
                    (T ; somebody forced be to the next line
                     (OUTPUT-CR CONTEXT (IL:IPLUS PAREN-WIDTH (IL:FETCH START-X IL:OF NODE))))))

```

```

(WHEN NEEDS-DOT?
 (OUTPUT-CONSTANT-STRING CONTEXT (IL:FETCH DOT-STRING IL:OF (IL:FETCH ENVIRONMENT
 IL:OF CONTEXT)))
 (OUTPUT-SPACE CONTEXT SPACE-WIDTH))
 (LINEARIZE SUBNODE CONTEXT)))
(IL:SETQ THIS-LINE? (AND (NOT COMMENT?)
 (NULL (CDR (IL:FETCH SUB-NODES IL:OF SUBNODE))))))
IL:FINALLY (WHEN COMMENT?
 (OUTPUT-CR CONTEXT (IL:IPLUS PAREN-WIDTH (IL:FETCH START-X IL:OF NODE))))))
(WHEN INDEX (IL:SHOULDN'T "linearize index out of range"))
(OUTPUT-CONSTANT-STRING CONTEXT (IL:FETCH RPAREN-STRING IL:OF (IL:FETCH ENVIRONMENT IL:OF CONTEXT))))

```

**LINEARIZE-LIST**

(IL:LAMBDA (NODE CONTEXT INDEX)

; Edited 15-Feb-88 13:24 by raf

;; The list linearizer. Present this list, driven by the previously-determined ListFormat.

```

(LET
 ((INFO (IL:FETCH UNASSIGNED IL:OF NODE)))
 (COND
 ((IL:FETCH NON-STANDARD? IL:OF INFO)
 ;; Non-standard ListFormats provide their own Linearize method -- use it.
 (FUNCCALL (IL:FETCH LINEARIZE-LIST IL:OF INFO)
 NODE CONTEXT INDEX))
 (T
 ;; Otherwise, we do the work
 (LET*
 ((ENVIRONMENT (IL:FETCH ENVIRONMENT IL:OF CONTEXT))
 (LPAREN (IL:FETCH LPAREN-STRING IL:OF ENVIRONMENT))
 (PAREN-WIDTH (IL:FETCH WIDTH IL:OF LPAREN))
 (SPACE-WIDTH (IL:FETCH SPACE-WIDTH IL:OF ENVIRONMENT))
 (STARTX (IL:FETCH START-X IL:OF NODE))
 (INDENT (IL:IPLUS STARTX PAREN-WIDTH) ; this will record the current tab setting
 )
 (FIRST T ; true until we've printed the first non-comment subnode
 )
 (PREV-TYPE NIL ; one of Atom, Comment, or NIL (other)
 )
 (NEXT-TYPE
 (PREV-INLINE NIL ; true if the last subnode printed inline
 )
 (SUBNODES (CDR (IL:FETCH SUB-NODES IL:OF NODE))) ; will point to the tail of subnodes beginning with the last
 (LAST-SUBNODE SUBNODES ; non-comment subnode
 )
 (RIGHT-MARGIN (IL:FETCH RIGHT-MARGIN IL:OF NODE))
 (COMMENT-SEPARATION (IL:FETCH COMMENT-SEPARATION IL:OF CONTEXT))
 (COMMENT-START (IL:IPLUS (IL:IDIFFERENCE RIGHT-MARGIN (IL:FETCH COMMENT-WIDTH IL:OF CONTEXT))
 COMMENT-SEPARATION))
 (INLINE? (AND (IL:FETCH INLINE-WIDTH IL:OF NODE)
 (IL:ILEQ (IL:IPLUS (IL:FETCH INLINE-WIDTH IL:OF NODE)
 STARTX)
 RIGHT-MARGIN)) ; true if we could fit this whole node inline
 )
 LAST-INFO ALREADY-INDENTED?) ; 'already.indented' is a real pain. part of the comment-indent
 ; look-ahead
 ;; Use either the preferred or minimal spacing information, depending on how much room we have
 (WHEN (NOT INDEX)
 (OUTPUT-CONSTANT-STRING CONTEXT LPAREN))
 (IL:FOR P IL:ON SUBNODES IL:WHEN (NOT (EQ (IL:FETCH NODE-TYPE IL:OF (CAR P))
 TYPE-COMMENT))
 IL:DO (IL:SETQ LAST-SUBNODE P))
 (COND
 (INLINE?
 ;; NODE will fit inline, so we don't run formatting rules, just print it.
 (DOLIST (SUBNODE (IF INDEX
 (NTHCDR INDEX SUBNODES)
 SUBNODES))
 (LINEARIZE SUBNODE CONTEXT)
 (UNLESS (EQ SUBNODE (CAR LAST-SUBNODE))
 (OUTPUT-SPACE CONTEXT SPACE-WIDTH))))))
 (T
 (IL:SETQ INFO (IF (IL:IGREATERP (IL:IPLUS (IL:FETCH PREFERRED-WIDTH IL:OF NODE)
 STARTX)
 RIGHT-MARGIN)
 (IL:FETCH LIST-MFORMAT IL:OF INFO)
 (IL:FETCH LIST-PFORMAT IL:OF INFO)))
 (IL:SETQ LAST-INFO (CAR INFO))
 ;; Find the last non-comment subnode

```

```

(IL:WHILE SUBNODES
  IL:DO
    (LET
      ((SUBNODE (CAR SUBNODES)))
      (COND
        ((EQ (IL:FETCH NODE-TYPE IL:OF SUBNODE)
              TYPE-COMMENT)
          (COND
            (INDEX (WHEN (EQ (IL:FETCH UNASSIGNED IL:OF SUBNODE)
                              2)
                        (WHEN (IL:NEQ INDENT (IL:FETCH START-X IL:OF SUBNODE))
                            (IL:SETQ ALREADY-INDENTED? T)
                            (IL:SETQ INDENT (IL:FETCH START-X IL:OF SUBNODE))))))
          (T
            ;; The rules for spacing before comments are tricky
            (IL:SELECTQ (IL:FETCH UNASSIGNED IL:OF SUBNODE)
              (1 ;; Level 1 comments will always start at comment.start. If the current line isn't already past the comment
                    ;; margin, start at the end of it -- otherwise on a new line
                    (OUTPUT-CR-OR-SPACE CONTEXT COMMENT-START COMMENT-SEPARATION))
              (2 ;; Level 2 comments start on a new line, unless they're the first thing in the list, and are indented to the
                    ;; tab setting. The trick is that unless we've just printed a comment, or we've already printed the last
                    ;; non-comment node in the list, we want the tab setting for the "next" element of the list (e.g. suppose
                    ;; we just printed a 'then') -- and the next element hasn't been printed yet... so we interpret the next
                    ;; separation info, and give it a chance to reset the tab first
                    (COND
                      (ALREADY-INDENTED? (OUTPUT-CR CONTEXT INDENT))
                      (NULL INFO)
                      (OUTPUT-CR-OR-SPACE CONTEXT INDENT SPACE-WIDTH))
                    ((AND FIRST (NULL PREV-TYPE))
                     ;; Level 2 comments at the beginning of a list (and not following other comments) immediately
                     ;; follow the (
                     )
                    (T
                     ;; Determine the separation info for the next element, and see if it sets the tab
                     (LET ((SEPR-INFO (CAR (OR (CDR INFO)
                                                INFO)))
                           (BREAK? NIL)
                           (SET-INDENT? NIL)
                           (INDENT-BASE (IL:IPLUS STARTX PAREN-WIDTH)))
                       (IL:WHILE (IL:LISTP SEPR-INFO)
                         IL:DO (IL:SETQ SEPR-INFO
                                      (IL:SELECTQ (CAR SEPR-INFO)
                                        (PREV-INLINE? (IF PREV-INLINE
                                                            (CADR SEPR-INFO)
                                                            (CDDR SEPR-INFO)))
                                        ((NEXT-INLINE? NEXT-PREFERRED? NEXT-ATOM?
                                                            NEXT-KEYWORD? NEXT-LAMBDAWORD?)
                                        (CDDR SEPR-INFO))
                                        (PREV-ATOM? (IF (IL:FMEMB PREV-TYPE
                                                                ' (ATOM KEYWORD LAMBDAWORD))
                                                            (CADR SEPR-INFO)
                                                            (CDDR SEPR-INFO)))
                                        (PREV-KEYWORD? (IF (EQ PREV-TYPE 'KEYWORD)
                                                            (CADR SEPR-INFO)
                                                            (CDDR SEPR-INFO)))
                                        (PREV-LAMBDAWORD? (IF (EQ PREV-TYPE 'LAMBDAWORD)
                                                            (CADR SEPR-INFO)
                                                            (CDDR SEPR-INFO)))
                                        (SET-INDENT (IL:SETQ SET-INDENT? T)
                                                    (CDR SEPR-INFO))
                                        (FROM-INDENT (IL:SETQ INDENT-BASE INDENT)
                                                    (CDR SEPR-INFO))
                                        (BREAK (IL:SETQ BREAK? T)
                                                    (CDR SEPR-INFO))
                                        (IL:SHOULDNT "Bad List Format"))))
                         (COND
                           (SET-INDENT? (IF BREAK?
                                              (OUTPUT-CR-OR-SPACE CONTEXT
                                                (IL:IMAX 1 (IL:IPLUS SEPR-INFO
                                                                INDENT-BASE))
                                                SPACE-WIDTH)
                                              (OUTPUT-SPACE CONTEXT SPACE-WIDTH))
                             (IL:SETQ INDENT (IL:FETCH CURRENT-X IL:OF CONTEXT))
                             (IL:SETQ ALREADY-INDENTED? T))
                           (T (OUTPUT-CR-OR-SPACE CONTEXT INDENT SPACE-WIDTH))))))
              ((3 4 5) ;; Level 3, 4 and 5 comments are aligned with the left edge of the root
                (OUTPUT-CR CONTEXT (IL:FETCH START-X IL:OF (IL:FETCH ROOT IL:OF CONTEXT))))
                (IL:SHOULDNT "unexpected comment level"))
              (LINEARIZE SUBNODE CONTEXT)))
            (IL:SETQ PREV-TYPE 'COMMENT)
            (IL:SETQ PREV-INLINE NIL))
          (T
            )
        )
      )
    )
  )
)

```

```

;; A non-comment node
(IL:SETQ NEXT-TYPE (NEXT-NODE-TYPE SUBNODE))
(COND
  (FIRST (IL:SETQ FIRST NIL)
    ;; If it was preceded by a comment, we'll need a new line
    (WHEN (AND PREV-TYPE (NOT INDEX))
      (OUTPUT-CR CONTEXT INDENT)))
  (ALREADY-INDENTED?
    ;; doesn't matter if this was the last subnode, since there won't be any more
    (WHEN (CDR INFO)
      (IL:SETQ INFO (CDR INFO)))
    (WHEN (NOT INDEX)
      (OUTPUT-CR CONTEXT INDENT)))
  (T (LET ((SEPR-INFO (COND
    ((AND (EQ SUBNODES LAST-SUBNODE)
      (NULL (CDDR INFO)))
      (IL:SETQ INFO NIL)
      LAST-INFO)
    (T (CAR (IL:SETQ INFO (OR (CDR INFO)
      INFO))))))
    (BREAK? NIL)
    (SET-INDENT? NIL)
    (INDENT-BASE (IL:IPLUS STARTX PAREN-WIDTH)))
    (IL:WHILE (IL:LISTP SEPR-INFO)
      IL:DO (IL:SETQ SEPR-INFO
        (IL:SELECTQ (CAR SEPR-INFO)
          (PREV-INLINE? (IF PREV-INLINE
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (NEXT-INLINE? (IF (AND (IL:FETCH INLINE-WIDTH IL:OF SUBNODE)
            (IL:ILEQ (IL:IPLUS (IL:FETCH CURRENT-X
              IL:OF CONTEXT)
              SPACE-WIDTH
              (IL:FETCH INLINE-WIDTH
                IL:OF SUBNODE))
              RIGHT-MARGIN))
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (NEXT-PREFERRED? (IF (IL:ILEQ (IL:IPLUS (IL:FETCH CURRENT-X
            IL:OF CONTEXT)
            SPACE-WIDTH
            (IL:FETCH PREFERRED-WIDTH
              IL:OF SUBNODE))
            RIGHT-MARGIN)
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (PREV-ATOM? (IF (IL:FMEMB PREV-TYPE ' (ATOM KEYWORD LAMBDABWORD)
            ))
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (PREV-KEYWORD? (IF (EQ PREV-TYPE 'KEYWORD)
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (PREV-LAMBDABWORD? (IF (EQ PREV-TYPE 'LAMBDABWORD)
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (NEXT-ATOM? (IF (IL:FMEMB NEXT-TYPE ' (ATOM KEYWORD LAMBDABWORD)
            ))
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (NEXT-KEYWORD? (IF (EQ NEXT-TYPE 'KEYWORD)
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (NEXT-LAMBDABWORD? (IF (EQ NEXT-TYPE 'LAMBDABWORD)
            (CADR SEPR-INFO)
            (CDDR SEPR-INFO)))
          (SET-INDENT (IL:SETQ SET-INDENT? T)
            (CDR SEPR-INFO))
          (FROM-INDENT (IL:SETQ INDENT-BASE INDENT)
            (CDR SEPR-INFO))
          (BREAK (IL:SETQ BREAK? T)
            (CDR SEPR-INFO))
          (IL:SHOULDNT "Bad List Format"))))
    (COND
      (INDEX (WHEN SET-INDENT?
        (IL:SETQ INDENT (IL:FETCH START-X IL:OF SUBNODE))))
      (T (COND
        ((EQ PREV-TYPE 'COMMENT)
          (OUTPUT-CR CONTEXT (IL:IMAX 1 (IL:IPLUS SEPR-INFO INDENT-BASE))))
        (BREAK? (OUTPUT-CR-OR-SPACE CONTEXT (IL:IMAX 1 (IL:IPLUS SEPR-INFO
          INDENT-BASE))
          SPACE-WIDTH))
        (T (OUTPUT-SPACE CONTEXT SPACE-WIDTH)))
        (WHEN SET-INDENT?
          (IL:SETQ INDENT (IL:FETCH CURRENT-X IL:OF CONTEXT)))))))))

```

;; Now we've got the appropriate spacing, linearize the subnode and set prev.inline and prev.type appropriately

```
(IL:SETQ PREV-INLINE (IF INDEX
                        (IL:FETCH INLINE? IL:OF SUBNODE)
                        (LINEARIZE SUBNODE CONTEXT)))
(IL:SETQ PREV-TYPE NEXT-TYPE)
(IL:SETQ ALREADY-INDENTED? NIL)))
(WHEN INDEX
 (IL:SETQ INDEX (AND (IL:NEQ INDEX 1)
                    (IL:SUB1 INDEX))))
 (IL:SETQ SUBNODES (CDR SUBNODES)))
(WHEN INDEX (IL:SHOULDNT "linearize index out of range"))
;; The closing paren goes on a new line if it's following a comment or there's no room for it on the previous line
(WHEN (OR (EQ PREV-TYPE 'COMMENT)
          (AND *WRAP-PARENS* (IL:IGREATERP (IL:IPLUS (IL:FETCH CURRENT-X IL:OF CONTEXT)
                                                    PAREN-WIDTH)
                                             RIGHT-MARGIN)
          (IL:ILESSP INDENT RIGHT-MARGIN))))
 (OUTPUT-CR CONTEXT INDENT)))
(OUTPUT-CONSTANT-STRING CONTEXT (IL:FETCH RPAREN-STRING IL:OF ENVIRONMENT))))))
```

**(LINEARIZE-QUOTE**

```
(IL:LAMBDA (X CONTEXT INDEX)
```

; Edited 17-Nov-87 11:33 by DCB

;;; the Linearize method for quoted structures. trivial

```
(COND
 ((NOT INDEX)
 (OUTPUT-CONSTANT-STRING CONTEXT (IL:FETCH UNASSIGNED IL:OF X))
 (LINEARIZE (CADR (IL:FETCH SUB-NODES IL:OF X))
            CONTEXT))
 ((IL:NEQ INDEX 1)
 (IL:SHOULDNT "linearize index out of range"))))
```

**(NEXT-NODE-TYPE**

```
(IL:LAMBDA (NODE)
```

; Edited 7-Jan-88 13:56 by DCB

;;; Return the "indentation type" of a node, one of atom, keyword, lambda word, or nil. Quote nodes return the type of their quoted structure; NIL nodes  
 ;; return atom or NIL depending on the node type.

```
(LET* ((STR (IL:|fetch| STRUCTURE IL:|of| NODE))
       (TYPE (IL:|ffetch| NODE-TYPE IL:|of| NODE)))
 (TYPECASE STR
  (CONS (IF (EQ TYPE TYPE-QUOTE)
            (NEXT-NODE-TYPE (SUBNODE 1 NODE))
            'NIL))
  (KEYWORD 'KEYWORD)
  (SYMBOL (COND
           ((EQ TYPE TYPE-LIST)
            NIL)
           ((IL:FMEMB STR LAMBDA-LIST-KEYWORDS)
            'LAMBDAWORD)
           (T 'ATOM))))
  (T 'ATOM))))
```

**(OUTPUT-CR-OR-SPACE**

```
(IL:LAMBDA (CONTEXT INDENT SPACE-WIDTH)
 (IF (IL:IGREATERP (IL:IPLUS (IL:FETCH CURRENT-X IL:OF CONTEXT)
                             SPACE-WIDTH)
          INDENT)
 (OUTPUT-CR CONTEXT INDENT)
 (OUTPUT-SPACE CONTEXT (IL:IDIFFERENCE INDENT (IL:FETCH CURRENT-X IL:OF CONTEXT))))))
```

; Edited 7-Jul-87 12:55 by DCB

**(PARENTESIZE-CURRENT-SELECTION**

```
(IL:LAMBDA (CONTEXT CHARCODE POINT-AFTER?)
```

; Edited 22-Dec-87 08:51 by DCB

```
(LET* ((SELECTION (IL:FETCH SELECTION IL:OF CONTEXT))
       (NODE (IL:FETCH SELECT-NODE IL:OF SELECTION))
       (START (IL:FETCH SELECT-START IL:OF SELECTION))
       (END (IL:FETCH SELECT-END IL:OF SELECTION))
       (POINT (IL:FETCH CARET-POINT IL:OF CONTEXT))
       (NODES NEW-NODE))
 (COND
  ((AND NODE (EQ (IL:FETCH SELECT-TYPE IL:OF SELECTION)
                 'STRUCTURE))
   (START-UNDO-BLOCK)
   (IF START
    (IL:SETQ NODES (IL:FOR I IL:FROM START IL:TO (OR END START) IL:AS SUBNODES
                          IL:ON (CDR (IL:NTH (IL:FETCH SUB-NODES IL:OF NODE)
                                             START))
                          IL:COLLECT (CAR SUBNODES)))
    (IL:SETQ NODES (LIST NODE)))
   (IL:REPLACE POINT-NODE IL:OF POINT IL:WITH SELECTION))
```



```
(IL:REPLACE POINT-TYPE IL:OF POINT IL:WITH 'STRUCTURE)
(IL:SETQ NEW-NODE (CREATE-NULL-LIST CONTEXT))
(INSERT POINT CONTEXT NEW-NODE)
(IL:SETQ NODES (IL:FOR N IL:IN NODES IL:WHEN (DEAD-NODE? N) IL:COLLECT N))
(IL:REPLACE POINT-NODE IL:OF POINT IL:WITH NEW-NODE)
(IL:REPLACE POINT-TYPE IL:OF POINT IL:WITH 'STRUCTURE)
(IL:REPLACE POINT-INDEX IL:OF POINT IL:WITH 0)
(INSERT POINT CONTEXT NODES)
(SELECT-NODE CONTEXT NEW-NODE)
(COND
  (POINT-AFTER? (SET-POINT POINT CONTEXT NEW-NODE NIL T))
  (T (IL:REPLACE POINT-NODE IL:OF POINT IL:WITH NEW-NODE)
      (IL:REPLACE POINT-TYPE IL:OF POINT IL:WITH 'STRUCTURE)
      (IL:REPLACE POINT-INDEX IL:OF POINT IL:WITH 0)))
(END-UNDO-BLOCK)
(T (FORMAT (GET-PROMPT-WINDOW CONTEXT)
  "%Select structure to parenthesize.")))
;; must return non-NIL if command executed
T))
```

(PARSE--LIST

```
(IL:LAMBDA (STRUCTURE CONTEXT) ; Edited 14-Jun-88 20:47 by drc:
```

;;; parse a list. if we're in default mode and it's undotted, check to see if it starts with a special word and if so parse it appropriately

```
(LET* ((PARSER (AND (IL:LITATOM (CAR STRUCTURE))
  (IL:LISTGET LIST-PARSE-INFO (CAR STRUCTURE))))
  (WHEN (NOT (AND PARSER (FUNCALL PARSER STRUCTURE CONTEXT)))
    (PARSE--LIST-INTERNAL STRUCTURE CONTEXT (AND (LISTP STRUCTURE)
      (ATOM (CAR STRUCTURE))
      (GET-LIST-FORMAT (CAR STRUCTURE)))))))
```

(PARSE--LIST-INTERNAL

```
(IL:LAMBDA (STRUCTURE CONTEXT FORMAT) ; Edited 14-Jun-88 21:26 by drc:
```

```
(LET ((NODE (BUILD-NODE STRUCTURE CONTEXT TYPE-LIST)))
  (LET* (LIST-POSITIONS SUB-FORMATS SUB-FORMATS-LENGTH SUBFORMAT)
    (WHEN FORMAT
      (SETQ LIST-POSITIONS (IL:|fetch| LIST-SUBLISTS IL:|of| FORMAT))
      (SETQ SUB-FORMATS (IL:|ffetch| LIST-FORMATS IL:|of| FORMAT))
      (SETQ SUB-FORMATS-LENGTH (IF SUB-FORMATS
        (LENGTH SUB-FORMATS)
        0)))
    (DO ((SUBLIST? NIL)
        (COMMENT? NIL)
        (NODE-COUNT 0)
        (TAIL STRUCTURE (CDR TAIL)))
      ((OR (ATOM TAIL)
        (AND (CONSP (CDR TAIL))
          (NULL (CDDR TAIL))
          (MEMBER (CAR TAIL)
            INTERNAL-WRAPPERS :TEST 'EQ)))
        (WHEN TAIL
          ;; when it's a real dotted-list or it's a dotted-wrapper, [e.g. (a . #'b)] then smash the type to dotted & parse TAIL as the
          ;; last subnode.
          (IL:|replace| NODE-TYPE IL:|of| NODE IL:|with| TYPE-DOTLIST)
          (PARSE TAIL CONTEXT)))
        (SETQ SUBNODE (CAR TAIL))
        (SETQ COMMENT? (AND (CONSP SUBNODE)
          (EQ (CAR SUBNODE)
            'IL:*)))
        (COND
          ((NOT COMMENT?)
            (INCF NODE-COUNT)
            (SETQ SUBLIST? (AND LIST-POSITIONS (NULL SUBNODE)
              (OR (EQ LIST-POSITIONS T)
                (MEMBER NODE-COUNT LIST-POSITIONS :TEST 'EQ))))
            (SETQ SUBFORMAT (WHEN (AND SUB-FORMATS (CONSP SUBNODE)
              (NOT (MEMBER (CAR SUBNODE)
                INTERNAL-WRAPPERS :TEST 'EQ)))
              (GET-LIST-FORMAT (IF (>= NODE-COUNT SUB-FORMATS-LENGTH)
                (FIRST SUB-FORMATS)
                (NTH NODE-COUNT SUB-FORMATS))))))
            (T (SETQ SUBLIST? NIL)
              (SETQ SUBFORMAT NIL)))
          (PARSE SUBNODE CONTEXT (WHEN (OR SUBLIST? SUBFORMAT)
            (IL:FUNCTION PARSE--LIST-INTERNAL)
            SUBFORMAT))))))
```

(PARSE--QUOTE

```
(IL:LAMBDA (STRUCTURE CONTEXT) ; Edited 7-Jul-87 12:55 by DCB
```

;;; try to parse this list as a quoted structure

```
(WHEN (AND (CDR STRUCTURE)
           (NULL (CDDR STRUCTURE)))
      (BUILD-NODE STRUCTURE CONTEXT TYPE-QUOTE)
      (IL:REPLACE UNASSIGNED IL:OF (IL:FETCH CURRENT-NODE IL:OF CONTEXT)
                 IL:WITH (IL:LISTGET (IL:FETCH QUOTE-STRING IL:OF (IL:FETCH ENVIRONMENT IL:OF CONTEXT))
                              (QUOTE-WRAPPER-NAME (CAR STRUCTURE))))
      (PARSE (CADR STRUCTURE)
            CONTEXT)
      ;; that is, if the object is quoted and not backquoted, then it can be parsed in Data mode, and not as a form
      T)))
```

**(REPLACE-LIST**

```
(IL:LAMBDA (NODE CONTEXT START END SUBNODES POINT REDOT?) ; Edited 22-Dec-87 11:12 by DCB
```

;; replaces the subnodes of NODE indexed by START through END with new subnodes SUBNODES. turns the list into a dotted list if REDOT? is true.  
 ;; may also undot a list.

```
(LET ((DOT-LIST? (EQ (IL:FETCH NODE-TYPE IL:OF NODE)
                    TYPE-DOTLIST))
      (INSERT-AFTER (IL:NTH (IL:FETCH SUB-NODES IL:OF NODE)
                          START))
      (TRAILING-SUBNODES (IL:NTH (IL:FETCH SUB-NODES IL:OF NODE)
                              (IL:IPLUS END 2)))
      (DELTA-LENGTH (IL:IDIFFERENCE (IL:LENGTH SUBNODES)
                                   (IL:ADD1 (IL:IDIFFERENCE END START))))
      TRAILING-STRUCTURE STRUCTURE CONVERTED? NEW-SUBNODE-COUNT UNDO-BOUNDS UNDO-STRUCTURE)
  ;; fix up subnode indices for those to follow the inserted material
  (IL:FOR S IL:IN TRAILING-SUBNODES IL:DO (IL:ADD (IL:FETCH SUB-NODE-INDEX IL:OF S)
                                                  DELTA-LENGTH))

  ;; fix the subnode count
  (IL:SETQ NEW-SUBNODE-COUNT (IL:IPLUS (CAR (IL:FETCH SUB-NODES IL:OF NODE))
                                       DELTA-LENGTH))
  (RPLACA (IL:FETCH SUB-NODES IL:OF NODE)
         NEW-SUBNODE-COUNT)

  ;; mark the deleted subnodes as dead, dead, dead
  (IL:FOR (DEAD-NODES IL: (CDR INSERT-AFTER)) IL:BY (CDR DEAD-NODES) IL:BIND DEAD-NODE
        IL:WHILE (IL:NEQ DEAD-NODES TRAILING-SUBNODES) IL:DO (IL:REPLACE SUPER-NODE IL:OF (IL:SETQ DEAD-NODE
                                                                 (CAR DEAD-NODES))
                                                                    IL:WITH 'DEAD!)
                (KILL-NODE DEAD-NODE)
                (IL:SETQ UNDO-STRUCTURE DEAD-NODES))

  ;; fix up the nodes to be inserted, and make a list out of their structures
  (COND
    (SUBNODES (IL:SETQ UNDO-BOUNDS (CONS START (IL:IPLUS END DELTA-LENGTH)))
              (IL:SETQ STRUCTURE (IL:FOR X IL:IN SUBNODES IL:AS I IL:FROM START
                                        IL:BIND (DEPTH IL: (IL:ADD1 (IL:FETCH DEPTH IL:OF NODE)))
                                        IL:COLLECT (IL:REPLACE SUB-NODE-INDEX IL:OF X IL:WITH I)
                                        (IL:REPLACE SUPER-NODE IL:OF X IL:WITH NODE)
                                        (SET-DEPTH X DEPTH)
                                        (IL:FETCH STRUCTURE IL:OF X))))
              (T (IL:SETQ UNDO-BOUNDS START)))
    (WHEN UNDO-STRUCTURE
      (RPLACD UNDO-STRUCTURE NIL)
      (IL:SETQ UNDO-STRUCTURE (CDR INSERT-AFTER)))

    ;; then insert those subnodes into the super's list
    (RPLACD INSERT-AFTER (NCONC SUBNODES TRAILING-SUBNODES))

    ;; and fix up the structure
    (COND
      ((OR (NULL (IL:FETCH STRUCTURE IL:OF NODE))
           (EQ 0 NEW-SUBNODE-COUNT))
        ;; changed this list to or from NIL. just replace it
        (IL:REPLACE STRUCTURE IL:OF NODE IL:WITH STRUCTURE)
        (SUBNODE-CHANGED NODE CONTEXT))
      (T (WHEN TRAILING-SUBNODES
            (IL:SETQ TRAILING-STRUCTURE (IL:NTH (IL:FETCH STRUCTURE IL:OF NODE)
                                                (IL:ADD1 END))))))

    (COND
      (EQ START 1)
      ;; replacing at the beginning of a list. play games with pointers
      (COND
        (EQ END 0)
        ;; straight insertion (nothing being replaced)
        (IL:SETQ TRAILING-STRUCTURE (CONS (CAR TRAILING-STRUCTURE)
                                          (CDR TRAILING-STRUCTURE))))
      ((AND DOT-LIST? (EQ NEW-SUBNODE-COUNT 1))
```

```

;; deleting everything in a dotted list but the element after the dot undots it
(IL:SETQ CONVERTED? T)
(IL:SETQ TRAILING-STRUCTURE (LIST TRAILING-STRUCTURE)))
(IL:RPLNODE2 (IL:FETCH STRUCTURE IL:OF NODE)
(NCONC STRUCTURE TRAILING-STRUCTURE)))
(T (IF (AND DOT-LIST? (NULL TRAILING-SUBNODES))
(WHEN (AND (EQ 0 DELTA-LENGTH)
(NULL (CDR SUBNODES)))
(IL:SETQ STRUCTURE (CAR STRUCTURE)))
(IL:SETQ STRUCTURE (NCONC STRUCTURE TRAILING-STRUCTURE)))
(RPLACD (IL:NTH (IL:FETCH STRUCTURE IL:OF NODE)
(IL:SUB1 START))
STRUCTURE))))))
;; fix up selection and insertion point
(WHEN POINT
(IL:REPLACE POINT-NODE IL:OF POINT IL:WITH NODE)
(IL:REPLACE POINT-INDEX IL:OF POINT IL:WITH (IL:IPLUS END DELTA-LENGTH))
(IL:REPLACE POINT-TYPE IL:OF POINT IL:WITH 'STRUCTURE))
(LET ((CARET (IL:FETCH CARET-POINT IL:OF CONTEXT)))
(COND
((AND (IL:NEQ CARET POINT)
(IL:TYPE? EDIT-NODE (IL:FETCH POINT-NODE IL:OF CARET)))
(COND
((DEAD-NODE? (IL:FETCH POINT-NODE IL:OF CARET))
;; if the caret was in the deleted material, we'll put it in the space the material was deleted from
(IL:REPLACE POINT-NODE IL:OF CARET IL:WITH NODE)
(IL:REPLACE POINT-INDEX IL:OF CARET IL:WITH (IL:IPLUS END DELTA-LENGTH))
(IL:REPLACE POINT-TYPE IL:OF CARET IL:WITH 'STRUCTURE))
((AND (EQ (IL:FETCH POINT-NODE IL:OF CARET)
NODE)
(IL:IGEQ (IL:FETCH POINT-INDEX IL:OF CARET)
START))
;; if it was between deleted items or after them in the list, it will need to be fixed up
(IL:REPLACE POINT-INDEX IL:OF CARET IL:WITH (IL:IPLUS DELTA-LENGTH
(IL:IMAX (IL:FETCH POINT-INDEX
IL:OF CARET)
END))))))
((AND (IL:NEQ CARET POINT)
(IL:TYPE? EDIT-SELECTION (IL:FETCH POINT-NODE IL:OF CARET)))
(LET* ((SELECTION (IL:FETCH POINT-NODE IL:OF CARET)))
(COND
((DEAD-NODE? (IL:FETCH SELECT-NODE IL:OF SELECTION))
(SET-SELECTION-NOWHERE SELECTION))
((AND (EQ (IL:FETCH SELECT-NODE IL:OF SELECTION)
NODE)
(IL:FETCH SELECT-START IL:OF SELECTION)
(IL:IGREATERP (IL:FETCH SELECT-START IL:OF SELECTION)
END))
;; the selection is after the stuff deleted. fix up the selection. don't need to worry about overlaps, because delete
;; overlaps cancel the selection and move overlaps aren't allowed, so can just do simple index translation.
(IL:REPLACE SELECT-START IL:OF SELECTION IL:WITH (IL:IPLUS DELTA-LENGTH
(IL:FETCH SELECT-START
IL:OF SELECTION)))
(IL:REPLACE SELECT-END IL:OF SELECTION IL:WITH (IL:IPLUS DELTA-LENGTH
(IL:FETCH SELECT-END
IL:OF SELECTION)))))))))
;; make sure this is a dotted list or not, as appropriate
(COND
(REDOT? (WHEN (OR DOT-LIST? (IL:ILESSP NEW-SUBNODE-COUNT 2))
(IL:SHOULDNT "shouldn't be redotting this one"))
(IL:REPLACE NODE-TYPE IL:OF NODE IL:WITH TYPE-DOTLIST)
(IL:SETQ DOT-LIST? T))
((OR CONVERTED? (AND DOT-LIST? (<= START END)
(NULL TRAILING-SUBNODES)
(OR (IL:ILESSP START END)
(IL:NEQ DELTA-LENGTH 0))))
;; dotted lists stop being dotted if you (a) delete everything but the last element, (b) replace a sequence of more than one subnode
;; including the last element, (c) delete the last element, or (d) replace the last element with more than one element
(IL:REPLACE NODE-TYPE IL:OF NODE IL:WITH TYPE-LIST)
(IL:SETQ CONVERTED? T)))
;; note change so that pretty-printer will fix up presentation
(NOTE-CHANGE NODE CONTEXT)
;; record how to undo this change
(UNDO-BY UNDO-LIST-REPLACE NODE UNDO-BOUNDS UNDO-STRUCTURE CONVERTED?)
(NIL))

```

(REPLACE-QUOTE

```
(IL:LAMBDA (NODE CONTEXT WHERE SUBNODES POINT) ; Edited 17-Jul-87 10:04 by DCB
  (LET ((SUBNODE (CAR SUBNODES)))
    (WHEN (NOT (OR (AND (IL:TYPE? EDIT-SELECTION WHERE)
                       (EQ (IL:FETCH SELECT-START IL:OF WHERE)
                            1)
                       (EQ (IL:FETCH SELECT-END IL:OF WHERE)
                            1)
                       (IL:TYPE? EDIT-NODE WHERE)))
      (IL:SHOULDNT "weird bounds for replace.quote"))
    (UNDO-BY UNDO-REPLACE-QUOTE NODE (SUBNODE 1 NODE))
    (KILL-NODE (SUBNODE 1 NODE))
    (RPLACA (CDR (IL:FETCH SUB-NODES IL:OF NODE))
            SUBNODE)
    (IL:REPLACE SUPER-NODE IL:OF SUBNODE IL:WITH NODE)
    (IL:REPLACE SUB-NODE-INDEX IL:OF SUBNODE IL:WITH 1)
    (RPLACA (CDR (IL:FETCH STRUCTURE IL:OF NODE))
            (IL:FETCH STRUCTURE IL:OF SUBNODE))
    (SET-DEPTH SUBNODE (IL:ADD1 (IL:FETCH DEPTH IL:OF NODE)))
    (NOTE-CHANGE NODE CONTEXT)
    (WHEN POINT (PUNT-SET-POINT POINT CONTEXT NODE T))
    (CDR SUBNODES))))
```

**(SET-LIST-FORMAT**

```
(IL:LAMBDA (FN FORMAT) ; Edited 1-Sep-87 14:54 by drc:
  (IF FORMAT
    (SETF (GETHASH FN LIST-FORMATS-TABLE)
          FORMAT)
    (REMHASH FN LIST-FORMATS-TABLE)))
```

**(SET-POINT-LIST**

```
(IL:LAMBDA (POINT CONTEXT NODE INDEX OFFSET ITEM TYPE COMPUTE-LOCATION?) ; Edited 22-Feb-88 14:33 by woz
```

;; the SetPoint method for lists, lambdas, clisps, etc.

```
(PROG ((DOTTED? (EQ (IL:|fetch| NODE-TYPE IL:|of| NODE)
                    TYPE-DOTLIST))
       (NUMBER-SUBNODES (CAR (IL:|fetch| SUB-NODES IL:|of| NODE))))
  (WHEN (NOT INDEX)
    ;; we can't set a point at our left or right boundary, but maybe our super can
    (RETURN (PUNT-SET-POINT POINT CONTEXT NODE OFFSET COMPUTE-LOCATION?)))
  (COND
   ((IL:|type?| STRING-ITEM ITEM)
    ;; pointing to the left paren, right paren, or dot. figure out which side they're pointing to
    (IL:SETQ OFFSET (IL:ILESSP OFFSET (IL:HALF (IL:FETCH WIDTH IL:OF ITEM))))
    (COND
     ((IL:STREQUAL (IL:FETCH STRING IL:OF ITEM) ".") ; it's a dot
      (IL:SETQ INDEX (IF OFFSET
                        (IL:SUB1 NUMBER-SUBNODES)
                        NUMBER-SUBNODES)))
      ((EQ OFFSET (EQ INDEX 1))
       ;; left side of the left paren or right side of the right paren puts us outside the list
       (RETURN (PUNT-SET-POINT POINT CONTEXT NODE (NOT OFFSET)
                               COMPUTE-LOCATION?)))
      (OFFSET
       ;; we must be on the right paren
       (IL:SETQ INDEX NUMBER-SUBNODES)
       ;; the left paren case is already correct, since index=0
       ))
    ((IL:|type?| EDIT-NODE ITEM)
     (IL:SETQ TYPE 'STRUCTURE))
    (T
     ;; space or cr. figure out which end we're closer to
     (WHEN (OR (IL:SMALLP (CADR (IL:FETCH LINEAR-FORM IL:OF NODE)))
              (IL:|type?| LINE-START (CADR (IL:FETCH LINEAR-FORM IL:OF NODE))))
       ;; starts with a comment (single-semi causing space, triple-semi causing line-start), so there's something extra as the second
       ;; thing in the linear form that we have to skip over
       (IL:SETQ INDEX (IL:SUB1 INDEX)))
     (IL:SETQ OFFSET (IL:ILESSP OFFSET (IL:HALF (OR (IL:SMALLP ITEM)
                                                      0))))
     (IF OFFSET
      (IL:SETQ INDEX (IL:HALF INDEX))
      (IL:SETQ INDEX (IL:HALF (IL:IPLUS 2 INDEX))))
     (WHEN DOTTED?
      (COND
       ((EQ INDEX NUMBER-SUBNODES)
        (WHEN (IL:SETQ OFFSET (NOT OFFSET))
          (IL:SETQ INDEX (IL:SUB1 INDEX))))
```

```

      ((EQ INDEX (IL:ADD1 NUMBER-SUBNODES))
       (IL:SETQ INDEX NUMBER-SUBNODES))))
    (WHEN (IL:IGREATERP INDEX NUMBER-SUBNODES)
      (IL:SETQ INDEX NUMBER-SUBNODES)
      (IL:SETQ OFFSET T))))
(COND
  ((AND (EQ TYPE 'ATOM)
        (IL:NEQ INDEX 0)
        (IL:ILEQ INDEX NUMBER-SUBNODES))
    (SET-POINT POINT CONTEXT (SUBNODE INDEX NODE)
      NIL OFFSET NIL 'ATOM COMPUTE-LOCATION?))
  ((AND DOTTED? (EQ INDEX NUMBER-SUBNODES))
    ;; can't insert structure after the dot in a dotted list
    (SET-POINT-NOWHERE POINT))
  (T (IL:|replace| POINT-NODE IL:|of| POINT IL:|with| NODE)
      (IL:|replace| POINT-INDEX IL:|of| POINT IL:|with| (IF OFFSET
                                                            INDEX
                                                            (IL:SETQ INDEX (IL:SUB1 INDEX))))
      (IL:|replace| POINT-TYPE IL:|of| POINT IL:|with| 'STRUCTURE)
      (WHEN COMPUTE-LOCATION? (COMPUTE-POINT-POSITION-LIST POINT))))))

```

**(SET-POINT-QUOTE**

```

(IL:LAMBDA (POINT CONTEXT NODE INDEX OFFSET ITEM TYPE COMPUTE-LOCATION?)
           ; Edited 17-Nov-87 11:34 by DCB

```

;;; the SetPoint method for quoted structures. there's no place to insert, so if we can't punt to the super or sub node there'll be no point

```

(COND
  ((NOT INDEX)
   (IF OFFSET
     (SET-POINT POINT CONTEXT (SUBNODE 1 NODE)
       NIL T NIL TYPE COMPUTE-LOCATION?)
     (PUNT-SET-POINT POINT CONTEXT NODE NIL COMPUTE-LOCATION?)))
  ((IL:TYPE? STRING-ITEM ITEM)
   (SET-POINT POINT CONTEXT (SUBNODE 1 NODE)
     NIL NIL NIL TYPE COMPUTE-LOCATION?))
  (OFFSET (PUNT-SET-POINT POINT CONTEXT NODE OFFSET COMPUTE-LOCATION?))
  (T (SET-POINT-NOWHERE POINT))))

```

**(SET-SELECTION-LIST**

```

(IL:LAMBDA (SELECTION CONTEXT NODE INDEX OFFSET ITEM TYPE)
           ; Edited 17-Nov-87 11:36 by DCB

```

;;; the SetSelection method for lists. pointing to the parens gets the whole list, pointing to whitespace gets nothing

```

(IF (OR (AND (IL:TYPE? STRING-ITEM ITEM)
             (EQ TYPE 'STRUCTURE))
        (IL:TYPE? EDIT-NODE ITEM))
    (SET-SELECTION-ME SELECTION CONTEXT NODE)
    (SET-SELECTION-NOWHERE SELECTION))))

```

**(SET-SELECTION-QUOTE**

```

(IL:LAMBDA (SELECTION CONTEXT NODE INDEX OFFSET ITEM TYPE)
           ; Edited 17-Nov-87 11:36 by DCB

```

;;; the SetSelection method for quoted structures

```

(IF (OR (AND (EQ INDEX 1)
             (EQ TYPE 'STRUCTURE))
        (IL:TYPE? EDIT-NODE ITEM))
    (SET-SELECTION-ME SELECTION CONTEXT NODE)
    (SET-SELECTION-NOWHERE SELECTION))))

```

**(STRINGIFY-LIST**

```

(IL:LAMBDA (NODE ENVIRONMENT)
           ; Edited 7-Jul-87 12:56 by DCB

```

```

(IL:|BIND| (STRINGS IL:_ '("")))
  (DOT IL:_ (EQ (IL:|FETCH| NODE-TYPE IL:|OF| NODE)
                TYPE-DOTLIST))
  IL:|FOR| SUBNODE IL:|IN| (IL:|REVERSE| (CDR (IL:|FETCH| SUB-NODES IL:|OF| NODE)))
  IL:|DO| (IL:|SETQ| STRINGS (CONS (COND
                                (DOT (IL:|SETQ| DOT NIL)
                                      ".")
                                (T " ")))
                                (CONS (STRINGIFY SUBNODE ENVIRONMENT)
                                      STRINGS)))
  IL:|FINALLY| (RETURN (IL:|CONCATLIST| (CONS "(" (CDR STRINGS))))))

```

**(STRINGIFY-QUOTE**

```

(IL:LAMBDA (NODE ENVIRONMENT)
           ; Edited 7-Jul-87 12:56 by DCB

```

```

(IL:|CONCAT| (IL:|FETCH| STRING IL:|OF| (IL:|FETCH| UNASSIGNED IL:|OF| NODE))
  (STRINGIFY (SUBNODE 1 NODE)
    ENVIRONMENT)))

```

**(SUBNODE-CHANGED-LIST**

(IL:LAMBDA (NODE SUBNODE CONTEXT)

; Edited 7-Jul-87 12:56 by DCB

;;; the SubNodeChanged method for lists of all flavours

;; stick in the new subnode

```
(IF (AND (EQ (IL:FETCH NODE-TYPE IL:OF NODE)
             TYPE-DOTLIST)
        (EQ (IL:FETCH SUB-NODE-INDEX IL:OF SUBNODE)
             (CAR (IL:FETCH SUB-NODES IL:OF NODE))))
    (RPLACD (IL:NTH (IL:FETCH STRUCTURE IL:OF NODE)
                (IL:SUB1 (IL:FETCH SUB-NODE-INDEX IL:OF SUBNODE)))
            (IL:FETCH STRUCTURE IL:OF SUBNODE))
    (RPLACA (IL:NTH (IL:FETCH STRUCTURE IL:OF NODE)
                  (IL:FETCH SUB-NODE-INDEX IL:OF SUBNODE))
            (IL:FETCH STRUCTURE IL:OF SUBNODE)))
```

;; note the change so that the pretty-printer can fix things up

```
(NOTE-CHANGE NODE CONTEXT))
```

**(SUBNODE-CHANGED-QUOTE**

(IL:LAMBDA (NODE SUBNODE)

; Edited 17-Nov-87 11:36 by DCB

;;; the SubNodeChanged method for quoted structures. not much interesting to happen here

```
(RPLACA (CDR (IL:FETCH STRUCTURE IL:OF NODE))
        (IL:FETCH STRUCTURE IL:OF SUBNODE)))
```

**(UNDO-LIST-REPLACE**

(IL:LAMBDA (CONTEXT NODE BOUNDS OLD-SUBNODES REDOT?)

; Edited 7-Jul-87 12:56 by DCB

;;; undo method for replaces within lists.

;; make sure you revive only dead nodes

```
(IL:FOR SUBNODE IL:IN OLD-SUBNODES IL:UNLESS (DEAD-NODE? SUBNODE) IL:DO (IL:SHOULDNT "undo is confused!"))
(LET ((LAST-INSERTED-SUBNODE (AND OLD-SUBNODES (CAR (LAST OLD-SUBNODES)))))
```

;; stick the dead nodes back in the list in place of the ones they were replaced by. replace.list will note the change to the list, which will  
;; cause the pretty-printer to fix up the presentation.

```
(REPLACE-LIST NODE CONTEXT (OR (IL:FIXP BOUNDS)
                                (CAR BOUNDS))
              (OR (CDR (IL:LISTP BOUNDS))
                  (IL:SUB1 BOUNDS))
              OLD-SUBNODES
              (IL:FETCH CARET-POINT IL:OF CONTEXT)
              REDOT?)
```

;; patch up selection

```
(WHEN OLD-SUBNODES
    (SELECT-SEGMENT (IL:FETCH SELECTION IL:OF CONTEXT)
                    CONTEXT NODE (CAR OLD-SUBNODES)
                    LAST-INSERTED-SUBNODE)
    (IL:REPLACE PENDING-DELETE? IL:OF (IL:FETCH SELECTION IL:OF CONTEXT) IL:WITH NIL))))
```

**(UNDO-REPLACE-QUOTE**

(IL:LAMBDA (CONTEXT NODE OLD-VALUE)

; Edited 7-Jul-87 12:56 by DCB

```
(WHEN (NOT (DEAD-NODE? OLD-VALUE))
    (IL:SHOULDNT "undo is confused!"))
(REPLACE-QUOTE NODE CONTEXT (SUBNODE 1 NODE)
    (LIST OLD-VALUE)
    NIL)
```

```
(WHEN (EQ (IL:FETCH NODE-TYPE IL:OF OLD-VALUE)
          TYPE-GAP)
    (SELECT-SEGMENT (IL:FETCH SELECTION IL:OF CONTEXT)
                    CONTEXT NODE OLD-VALUE OLD-VALUE)
    (PENDING-DELETE (IL:FETCH CARET-POINT IL:OF CONTEXT)
                    (IL:FETCH SELECTION IL:OF CONTEXT))))
```

)

(IL:PUTPROPS IL:SEdit-LISTS IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1990))

---

**FUNCTION INDEX**

ASSIGN-FORMAT-CLISP .....	1	DELETE-LIST .....	8	PARSE--LIST-INTERNAL .....	17
ASSIGN-FORMAT-DOTLIST .....	2	DELETE-QUOTE .....	8	PARSE--QUOTE .....	17
ASSIGN-FORMAT-LIST .....	2	DOT-THIS-LIST .....	8	REPLACE-LIST .....	18
ASSIGN-FORMAT-QUOTE .....	2	GET-LIST-FORMAT .....	8	REPLACE-QUOTE .....	19
BACKSPACE-LIST .....	3	INITIALIZE-LISTS .....	9	SET-LIST-FORMAT .....	20
BACKSPACE-QUOTE .....	3	INSERT-LIST .....	9	SET-POINT-LIST .....	20
CFV-CLISP .....	3	INSERT-NULL-LIST .....	9	SET-POINT-QUOTE .....	21
CFV-DOTLIST .....	4	INSERT-QUOTED-GAP .....	10	SET-SELECTION-LIST .....	21
CFV-LIST .....	4	LINEARIZE-CLISP .....	10	SET-SELECTION-QUOTE .....	21
CFV-QUOTE .....	6	LINEARIZE-DOTLIST .....	12	STRINGIFY-LIST .....	21
CLOSE-LIST .....	6	LINEARIZE-LIST .....	13	STRINGIFY-QUOTE .....	21
COMPUTE-POINT-POSITION-LIST .....	6	LINEARIZE-QUOTE .....	16	SUBNODE-CHANGED-LIST .....	22
COPY-STRUCTURE-LIST .....	7	NEXT-NODE-TYPE .....	16	SUBNODE-CHANGED-QUOTE .....	22
COPY-STRUCTURE-QUOTE .....	7	OUTPUT-CR-OR-SPACE .....	16	UNDO-LIST-REPLACE .....	22
CREATE-NULL-LIST .....	7	PARENTHE SIZE-CURRENT-SELECTION ..	16	UNDO-REPLACE-QUOTE .....	22
CREATE-QUOTED-GAP .....	7	PARSE--LIST .....	17		

---

**VARIABLE INDEX**

*FORMAT-ALIAS-DEPTH-LIMIT* .....	1	CLISP-INDENT-WORDS .....	1	INTERNAL-WRAPPERS .....	1
*WRAP-PARENS* .....	1	CLISP-PROGRAM-WORDS .....	1	LIST-PARSE-INFO .....	1

---

**PROPERTY INDEX**

IL:SEDIT-LISTS .....	1
----------------------	---

---