

File created: 2-Dec-92 17:28:09 {PELE:MV:ENVOS}<LISPCORE>SOURCES>SEdit-BASE.;7

changes to: (IL:FNS SETUP-WINDOW-AND-PROCESS)

previous date: 10-Jul-91 15:05:17 {PELE:MV:ENVOS}<LISPCORE>SOURCES>SEdit-BASE.;6

Read Table: XCL

Package: SEDIT

Format: XCCS

; Copyright (c) 1987, 1988, 1990, 1991, 1992 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ **IL:SEdit-BASECOMS**

```
((IL:PROP IL:FILETYPE IL:SEdit-BASE)
 (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:SEdit-BASE)
 (IL:LOCALVARS . T)
 (IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FILES IL:SEdit-DECLS))
 (IL:BITMAPS GAP-BITMAP ARGS-BITMAP BODY-BITMAP)
 (IL:VARIABLES *CLEAR-LINEAR-ON-COMPLETION* *IGNORE-CHANGES-ON-COMPLETION* *COMPILE-FN*)
 (IL:FUNCTIONS COMPLETE THROW-AWAY-CHANGES SET-INITIAL-SELECTION PREV-NODE MAKE-FUNCTION-PROTOTYPE)
 (IL:P (IL:MOVD 'MAKE-FUNCTION-PROTOTYPE 'XCL::%MAKE-FUNCTION-PROTOTYPE))
 (IL:FNS ADJUST-WIDTH ASSIGN-FORMAT-NIL ATOM-CHANGE-RELINEARIZE BUILD-INTERNAL-STRUCTURE
 BUILD-LINEAR-FORM BUILD-NODE BUILD-PRELINEARIZED-NODE CLOSE-NODE COLLECT-UNDO-BLOCK
 COMPILE-STRUCTURE COMPUTE-ALL-FORMATS COMPUTE-FORMATS-AND-FORMAT-VALUES COMPUTE-POINT-POSITION
 COMPUTE-SELECTION-POSITION COMPUTE-SELECTION-POSITION-DEFAULT CONTAINS? COPY-NODE
 COPY-SELECTION COPY-SELECTION-DEFAULT CREATE-CONSTANT-STRINGS CREATE-ENVIRONMENTS
 CREATE-GAP-NODE CREATE-NODE CREATE-PRELINEARIZED-NODE CREATE-PRETTY-PRINT-ENV
 CREATE-SIMPLE-NODE CREATE-STRING-ITEM DEFAULT-COMPILE-FN DEFAULT-GETDEF-FN DEFAULT-PACKAGE
 DELETE-NODES DETACH-NODE FORMAT-VALUES-CHANGED GET-SELECTED-STRUCTURE HANDLE-COMPLETION
 INITIALIZE INSERT INSERT-CHANGED KILL-NODE LINEARIZE-ROOT NEXT-NODE NOTE-CHANGE
 NOTE-CHANGE-FORMAT NOTE-CHANGE-IN-SIMPLE PARSE PARSE--GAP PARSE--UNKNOWN PARSE-NEW
 PROPAGATE-WIDTH-CHANGE RECOMPUTE-WIDTH RELINEARIZE-WHERE-NECESSARY REPLACE-NODE REPLACE-ROOT
 REVIVE-NODE SEDIT1 SELECT-NEXT-GAP SET-DEPTH SET-FORMAT SETUP-CONTEXT
 SETUP-CONTEXT-WINDOW-DEPENDENCIES SETUP-NEW-CONTEXT SETUP-PROFILE SETUP-WINDOW-AND-PROCESS
 SETUP-WINDOW-CONTEXT-DEPENDENCIES SHIFT-LINEAR-FORM STRINGIFY STRINGIFY-GAP SUBNODE-CHANGED
 SUBNODE-CHANGED-ROOT TYPE-OF-INPUT UNDO-EVENT UNDO-REPLACE-ROOT UPDATE VERIFY-STRUCTURE
 WALK-UP-TREE)))
```

(IL:PUTPROPS **IL:SEdit-BASE IL:FILETYPE** :COMPILE-FILE)

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

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

(IL:LOCALVARS . T)
)

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

(IL:FILESLOAD IL:SEdit-DECLS)
)

(IL:RPAQQ **GAP-BITMAP** ~~—x—~~)

(IL:RPAQQ **ARGS-BITMAP** ~~—args—~~)

(IL:RPAQQ **BODY-BITMAP** ~~—body—~~)

(DEFPARAMETER ***CLEAR-LINEAR-ON-COMPLETION*** NIL
"Completion assumes old linear forms may be garbage.")

(DEFPARAMETER ***IGNORE-CHANGES-ON-COMPLETION*** T
"If T, markaschangedfn will ignore calls caused by completion of that edit.")

(DEFPARAMETER ***COMPILE-FN*** 'DEFAULT-COMPILE-FN)

(DEFUN **COMPLETE** (CONTEXT CHARCODE REASON COMPILE?)

;;; entry point into completing an sedit. this function is invoked by the completion commands and the closefn and shrinkfn if the sedit is not busy.
;;; REASON specifies how the user wants to complete, one of :CLOSE, :SHRINK, :ABORT, or :DONE. :ABORT means throw away changes, and
;;; :DONE means complete and leave the window open. In order for SEdit to unwind itself properly, all completion must begin in the SEdit process with
;;; the window still open, so the closefn and shrinkfn return DON'T if they're running under the mouse, and the window will be closed appropriately here.
;;; COMPILE? is T if keyboard command says to compile.

```
(LET ((COMPILE-SUCCEEDED? T)
      (OPTIONS (IL:FETCH EDIT-OPTIONS IL:OF CONTEXT)))
  (CLOSE-OPEN-NODE CONTEXT)
  (WHEN (EQ REASON :ABORT)
    (UNLESS (IL:MOUSECONFIRM "Click LEFT to ABORT ALL changes." T (GET-PROMPT-WINDOW CONTEXT))
```

```

(RETURN-FROM COMPLETE T))
;; IDEALLY: if we're editing an "expression" (not a definition), assume editing structure in place (destructively), and so to abort we must
;; undo all the edits.
;; HOWEVER: since the file manager (editdef, getdef) edits il:fns, il:vars, etc in place, not a definition, to abort on these types we must
;; undo.
;; FOR NOW (1/13/91) just undo always. If the edit interface is changed to always edit a copy/definition for any type, then just undo if
;; type=:expression.
(DO NIL
  ((NULL (IL:FETCH UNDO-LIST IL:OF CONTEXT)))
  (UNDO CONTEXT)))
(HANDLE-COMPLETION CONTEXT REASON)
(WHEN (AND (NOT (EQ REASON :ABORT))
           (OR COMPILER? (MEMBER :COMPILE-ON-COMPLETION OPTIONS))))
  (SETQ COMPILER-SUCCEEDED? (COMPILE-STRUCTURE CONTEXT)))
(COND
  ((NOT COMPILER-SUCCEEDED?)
   ;; if the compile failed, don't continue
  )
  ((OR (EQ REASON :CLOSE)
        (EQ REASON :ABORT)
        (MEMBER :CLOSE-ON-COMPLETION OPTIONS))
   (IL:CLOSEW (IL:fetch) DISPLAY-WINDOW IL:of) CONTEXT))
  (DISINTEGRATE-CONTEXT CONTEXT)
  (IL:DEL.PROCESS (IL:THIS.PROCESS)))
  ((EQ REASON :SHRINK)
   (IL:SHRINKW (IL:fetch) DISPLAY-WINDOW IL:of) CONTEXT))
  (IL:DEL.PROCESS (IL:THIS.PROCESS)))
  ((EQ REASON :DONE)
   (IL:TTY.PROCESS T)))
T))

```

```

(DEFUN THROW-AWAY-CHANGES (CONTEXT)
  (IL:REPLACE (EDIT-CONTEXT ATOM-STARTED) IL:OF CONTEXT IL:WITH NIL)
  (IL:REPLACE (EDIT-CONTEXT ATOM-STARTED-UNDO-POINTER) IL:OF CONTEXT IL:WITH NIL)
  (IL:REPLACE (EDIT-CONTEXT CHANGED-STRUCTURE?) IL:OF CONTEXT IL:WITH NIL)
  (IL:REPLACE (EDIT-CONTEXT UNDO-LIST) IL:OF CONTEXT IL:WITH NIL)
  (IL:REPLACE (EDIT-CONTEXT UNDO-UNDO-LIST) IL:OF CONTEXT IL:WITH NIL))

```

```

(DEFUN SET-INITIAL-SELECTION (CONTEXT)

```

;; the initial selection was stored by set-props in the find-candidate field, in the form (structure . instance). Find and select the nth instance of structure,
 ;; then replace the find candidate with just the structure. If there is no candidate, select the next gap.

```

(LET ((CANDIDATE (IL:FETCH (EDIT-CONTEXT FIND-CANDIDATE) IL:OF CONTEXT)))
  (WHEN (CONSP CANDIDATE)
    (SELECTION-DOWN CONTEXT)
    (COND
      (CANDIDATE (FIND-NTH-STRUCTURE CONTEXT NIL (CAR CANDIDATE)
                                     (CDR CANDIDATE))
                 (IL:REPLACE (EDIT-CONTEXT FIND-CANDIDATE) IL:OF CONTEXT IL:WITH (CAR CANDIDATE)))
      (T (SELECT-NEXT-GAP CONTEXT (IL:fetch) ROOT IL:of) CONTEXT)))
    (COMPUTE-SELECTION-POSITION (IL:fetch) SELECTION IL:of) CONTEXT)
  (SHOW-CARET CONTEXT)
  (SELECTION-UP CONTEXT)))

```

```

(DEFUN PREV-NODE (NODE &OPTIONAL INDEX)

```

;; step to the previous node before this one (in postorder). if index is a fixp, start with the first subnode before the one with that index. if it's T, start with
 ;; the first node before this node. if it's NIL, start with this node's last subnode

```

(DO* ((SUBNODES (IL:fetch) SUB-NODES IL:of) NODE)
      (IL:fetch) SUB-NODES IL:of) NODE))
  (LASTINDEX (1+ (FIRST SUBNODES))
             (1+ (FIRST SUBNODES)))
  (INDEX (OR INDEX LASTINDEX)))
  ((AND (INTEGERP INDEX)
        (> INDEX 1)
        (<= INDEX LASTINDEX))
   (NTH (1- INDEX)
        SUBNODES))
  (SETF INDEX (IL:fetch) SUB-NODE-INDEX IL:of) NODE))
  (UNLESS (SETF NODE (IL:fetch) SUPER-NODE IL:of) NODE))
  (RETURN NIL)))

```

```

(DEFUN MAKE-FUNCTION-PROTOTYPE ()
  (DECLARE (GLOBAL ARGS-GAP BODY-GAP))
  (IF (EQ (IL:EDITMODE)
          'IL:SEdit)
      (LIST ARGS-GAP BODY-GAP)

```

```
(LIST (LIST "Arg List")
      "Body"))
```

```
(IL:MOVD 'MAKE-FUNCTION-PROTOTYPE 'XCL:;%MAKE-FUNCTION-PROTOTYPE)
```

```
(IL:DEFINEQ
```

(ADJUST-WIDTH

```
(IL:LAMBDA (NODE CONTEXT NEW-WIDTH) ; Edited 6-Jul-87 20:48 by DCB
```

;;; we've made some change to an open node. adjust the widths and notice the changes

```
(LET ((STRING-ITEM (CAR (IL:FETCH LINEAR-FORM IL:OF NODE))))
      (IL:REPLACE WIDTH IL:OF STRING-ITEM IL:WITH NEW-WIDTH)
      (IL:REPLACE INLINE-WIDTH IL:OF NODE IL:WITH NEW-WIDTH)
      (IL:REPLACE PREFERRED-WIDTH IL:OF NODE IL:WITH NEW-WIDTH)
      (IL:REPLACE ACTUAL-WIDTH IL:OF NODE IL:WITH NEW-WIDTH)
      (IL:REPLACE ACTUAL-LLENGTH IL:OF NODE IL:WITH NEW-WIDTH)
      (WHEN CONTEXT
        (COND
          ((EQ (IL:FETCH NODE-TYPE IL:OF NODE)
              TYPE-LITATOM)
            (NOTE-CHANGE NODE CONTEXT))
          (T (IL:REPLACE CHANGED? IL:OF NODE IL:WITH T)
             (NOTE-CHANGE (IL:FETCH SUPER-NODE IL:OF NODE)
                           CONTEXT))))))
```

(ASSIGN-FORMAT-NIL

```
(IL:LAMBDA (NODE CONTEXT FORMAT) ; Edited 6-Jul-87 20:48 by DCB
```

;;; assigns NIL as the format for each of the subnodes of node.

;;; IMPORTANT NOTE: all nonleaf node types (except node types which have only prelinearized subnodes) must have a method which actually resets the format of each of their subnodes. if they don't care what format type they assign, they should use this method. (if the subnode format is not actually reset from the unassigned value, the format assigner and width estimator will not be run, with yukky results.)

```
(IL:FOR SUBNODE IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE)) IL:DO (SET-FORMAT SUBNODE CONTEXT NIL))
```

(ATOM-CHANGE-RELINEARIZE

```
(IL:LAMBDA (CONTEXT) ; Edited 24-Aug-87 16:22 by drc:
```

;;; a simple method for relinearizing everything when we think display of atoms may have changed, like if we view the structure from a different package.

;;; need to waste cached atom info in point and selection, as well as make sure the structure is intact.

```
(CLOSE-OPEN-NODE CONTEXT)
(SET-POINT-NOWHERE (IL:|fetch| CARET-POINT IL:|of| CONTEXT))
(SET-SELECTION-NOWHERE (IL:|fetch| SELECTION IL:|of| CONTEXT))
```

;;; recompute widths for the whole tree

```
(WALK-UP-TREE (IL:FETCH ROOT IL:OF CONTEXT)
             CONTEXT
             #'(LAMBDA (NODE CONTEXT)
                 (IF (EQ (IL:FETCH NODE-TYPE IL:OF NODE)
                       TYPE-LITATOM)
                     (LET* ((STRUCTURE (IL:FFETCH STRUCTURE IL:OF NODE))
                            (BROKEN? (IL:TYPE? BROKEN-ATOM STRUCTURE)))
                         (UNLESS BROKEN? ; smash new width into real atom nodes
                           (LET* ((STRING-ITEM (CAR (IL:FFETCH LINEAR-FORM IL:OF NODE)))
                                  (WIDTH (STRINGWIDTH STRUCTURE (IL:FFETCH FONT IL:OF STRING-ITEM)
                                                                    (NOT BROKEN?))))
                               (IL:FREPLACE WIDTH IL:OF STRING-ITEM IL:WITH WIDTH)
                               (IL:FREPLACE INLINE-WIDTH IL:OF NODE IL:WITH WIDTH)
                               (IL:FREPLACE PREFERRED-WIDTH IL:OF NODE IL:WITH WIDTH)
                               (IL:FREPLACE ACTUAL-WIDTH IL:OF NODE IL:WITH WIDTH)
                               (IL:FREPLACE ACTUAL-LLENGTH IL:OF NODE IL:WITH WIDTH))))
                         ;; just call CFV method for other node types
                         (FUNCALL (IL:FETCH COMPUTE-FORMAT-VALUES IL:OF (IL:FFETCH NODE-TYPE IL:OF NODE))
                                  NODE
                                  (IL:FETCH ENVIRONMENT IL:OF CONTEXT)
                                  CONTEXT
                                  (IL:FFETCH FORMAT IL:OF NODE))) ; mark all nodes as changed
                           (IL:FREPLACE CHANGED? IL:OF NODE IL:WITH T))))
                 (RELINERIZE (IL:FETCH ROOT IL:OF CONTEXT)
                             CONTEXT)))
```

(BUILD-INTERNAL-STRUCTURE

```
(IL:LAMBDA (CONTEXT) ; Edited 6-Apr-88 16:25 by woz
```

;;; called when setting up a new context. the structure to parse was stored in the Root field of the context by get.context. here we grab it and then setup the context for parsing.

```
(LET ((STRUCTURE (IL:|fetch| ROOT IL:|of| CONTEXT))
      (ROOT (IL:|create| EDIT-NODE
              NODE-TYPE IL:_ TYPE-ROOT
              DEPTH IL:_ 1
              SUB-NODES IL:_ (LIST 0)
              LINEAR-FORM IL:_ (CONS)
              START-X IL:_ 1
              ACTUAL-WIDTH IL:_ 0))
      (STRING (IL:|ALLOCSTRING| 512 NIL NIL T)))
  (IL:|replace| ROOT IL:|of| CONTEXT IL:|with| ROOT)
  (IL:|replace| CARET-POINT IL:|of| CONTEXT IL:|with| (IL:|create| EDIT-POINT))
  (IL:|replace| SELECTION IL:|of| CONTEXT IL:|with| (IL:|create| EDIT-SELECTION))
  (IL:|replace| CURRENT-NODE IL:|of| CONTEXT IL:|with| ROOT)
  (IL:|replace| \\X IL:|of| CONTEXT IL:|with| NIL)
  (IL:|replace| OPEN-NODE IL:|of| CONTEXT IL:|with| NIL)
  (IL:|replace| OPEN-NODE-INFO IL:|of| CONTEXT IL:|with| (IL:|create| OPEN-STRING
                                                         BUFFER-STRING IL:_ STRING
                                                         SUBSTRING IL:_ (IL:|SUBSTRING| STRING 1 1)))

;; now we're ready to build the actual structures. build SEdit tree; propagate format types and compute space estimates; and compute
;; actual presentation
(PARSE STRUCTURE CONTEXT)
(COMPUTE-ALL-FORMATS CONTEXT)
(BUILD-LINEAR-FORM CONTEXT)))
```

(BUILD-LINEAR-FORM

(IL:LAMBDA (CONTEXT) ; Edited 6-Apr-88 16:38 by woz

;; help initialize this context by filling in the linear form. we fill in initial values for a bunch of fields and then call linearize

```
(LET ((ROOT (IL:|fetch| ROOT IL:|of| CONTEXT)))
  (IL:|replace| CURRENT-X IL:|of| CONTEXT IL:|with| (IL:|fetch| START-X IL:|of| ROOT))
  (IL:|replace| CURRENT-NODE IL:|of| CONTEXT IL:|with| ROOT)
  (IL:|replace| LAST-LINEARIZED-SUB-NODE-INDEX IL:|of| CONTEXT IL:|with| 0)
  (IL:|replace| LINEAR-FORM IL:|of| ROOT
              IL:|with| (CONS (IL:|create| LINE-START
                                PREV-LINE IL:_ NIL
                                NODE IL:_ ROOT
                                LINE-SKIP IL:_ 2
                                LINE-ASCENT IL:_ 0
                                LINE-DESCENT IL:_ 0
                                INDENT IL:_ (IL:|fetch| START-X IL:|of| ROOT)
                                YCOORD IL:_ 0)
                              (CREATE-WEAK-LINK ROOT))) ; create the initial, unfilled-in linear form for the root.
  (IL:|replace| LINEAR-PREV IL:|of| CONTEXT IL:|with| (IL:|fetch| LINEAR-FORM IL:|of| ROOT))
  (IL:|replace| CURRENT-LINE IL:|of| CONTEXT IL:|with| (IL:|fetch| LINEAR-FORM IL:|of| ROOT))
  (IL:|replace| FIRST-LINE IL:|of| ROOT IL:|with| (CAR (IL:|fetch| LINEAR-FORM IL:|of| ROOT)))
  (IL:|replace| LINEAR-POINTER IL:|of| CONTEXT IL:|with| (CDR (IL:|fetch| LINEAR-FORM IL:|of| ROOT)))
  ; this must be special: normally linear-pointer at a weak-link
  ; means we're done with this node and go up to the super.
  (IL:|replace| FIRST-BLOCK IL:|of| CONTEXT IL:|with| (IL:|create| LINE-BLOCK
                                                                BLOCK-NEW-X IL:_ (IL:|fetch| START-X IL:|of| ROOT)
                                                                BLOCK-START IL:_ (IL:|fetch| LINEAR-FORM IL:|of| ROOT)))
  (IL:|replace| CURRENT-BLOCK IL:|of| CONTEXT IL:|with| (IL:|fetch| FIRST-BLOCK IL:|of| CONTEXT))
  (IL:|replace| RELINEARIZATION-TIME-STAMP IL:|of| CONTEXT IL:|with| 0)
  (IL:|replace| BELOW? IL:|of| CONTEXT IL:|with| 'NEW)
  (LINEARIZE (SUBNODE 1 ROOT)
             CONTEXT
             (IL:|IDIFFERENCE| (IL:|WINDOWPROP| (IL:|fetch| DISPLAY-WINDOW IL:|of| CONTEXT)
                                             'IL:|WIDTH|
                                             5)) ; fix up some of the information recorded in the root
  (IL:|replace| LINE-LENGTH IL:|of| (CAR (IL:|fetch| CURRENT-LINE IL:|of| CONTEXT)) IL:|with| (IL:|fetch| CURRENT-X
                                                                                          IL:|of| CONTEXT))
  (IL:|replace| ACTUAL-LENGTH IL:|of| ROOT IL:|with| (IL:|IDIFFERENCE| (IL:|fetch| CURRENT-X IL:|of| CONTEXT)
                                                                    (IL:|fetch| START-X IL:|of| ROOT)))
  (IL:|replace| ACTUAL-WIDTH IL:|of| ROOT IL:|with| (IL:|IDIFFERENCE| (IL:|fetch| ACTUAL-WIDTH IL:|of| ROOT)
                                                                    (IL:|fetch| START-X IL:|of| ROOT)))
  ; used to replace LastLineLinear of root with (fetch CurrentLine of
  ; context)
  (IL:|replace| LAST-LINE IL:|of| ROOT IL:|with| (CAR (IL:|fetch| CURRENT-LINE IL:|of| CONTEXT)))
  ; if we haven't finished updating the window, make sure the last
  ; line is dumped properly
  (WHEN (EQ (IL:|fetch| BELOW? IL:|of| CONTEXT)
            'NEW)
        (REPAINT-NEW-LINE (IL:|fetch| CURRENT-LINE IL:|of| CONTEXT))))))
```

(BUILD-NODE

(IL:LAMBDA (STRUCTURE CONTEXT NODE-TYPE TRUST-SUBNODES) ; Edited 3-Dec-87 15:47 by DCB

```
(IL:|replace| CURRENT-NODE IL:|of| CONTEXT
  IL:|with| (IL:|bind| (TAIL IL:_ (IL:|fetch| \\X IL:|of| CONTEXT))
              SUBNODES IL:|while| (IL:|SETQ| SUBNODES (CDR TAIL))
              IL:|do| (WHEN (EQ STRUCTURE (IL:|fetch| STRUCTURE IL:|of| (CAR SUBNODES)))
                      (COND
                       ((EQ NODE-TYPE (IL:|fetch| NODE-TYPE IL:|of| (CAR SUBNODES)))
                        ;; we can re-use the node
```

```

(RPLACD TAIL (CDR SUBNODES))
(IL:|replace| SUB-NODE-INDEX IL:|of| (IL:SETQ SUBNODES (CAR SUBNODES))
  IL:|with| (IL:ADD1 (CAR (IL:SETQ TAIL (IL:|fetch| SUB-NODES
    IL:|of| (IL:|fetch| CURRENT-NODE IL:|of|
      )))
    )))
(WHEN (NOT TRUST-SUBNODES)
  ;; we were just called from parse-comment, damn it!
  (IL:|replace| \X IL:|of| CONTEXT IL:|with| (IL:|fetch| SUB-NODES IL:|of| SUBNODES))
  (IL:|replace| SUB-NODES IL:|of| SUBNODES IL:|with| (LIST 0))
  (IL:NCONC1 TAIL SUBNODES)
  (RPLACA TAIL (IL:|fetch| SUB-NODE-INDEX IL:|of| SUBNODES))
  (RETURN SUBNODES))
(T
  ;; it's changed type -- make this undoable
  (LET ((NEW-NODE (CREATE-NODE STRUCTURE (IL:|fetch| CURRENT-NODE IL:|of| CONTEXT)
    NODE-TYPE)))
    (UNDO-BY REPLACE-NODE NEW-NODE (CAR SUBNODES))
    (IL:|replace| \X IL:|of| CONTEXT IL:|with| NIL)
    (RETURN NEW-NODE))))
(IL:SETQ TAIL SUBNODES)
IL:|finally| (IL:|replace| \X IL:|of| CONTEXT IL:|with| NIL)
  (RETURN (CREATE-NODE STRUCTURE (IL:|fetch| CURRENT-NODE IL:|of| CONTEXT)
    NODE-TYPE))))

```

(BUILD-PRELINEARIZED-NODE

```

(IL:LAMBDA (STRUCTURE CONTEXT NODE-TYPE STRING PRIN-2? FONT) ; Edited 19-Aug-87 17:44 by drc:
  (IL:REPLACE CURRENT-NODE IL:OF CONTEXT
    IL:WITH (IL:BIND (TAIL IL:_ (IL:FETCH \X IL:OF CONTEXT))
      SUBNODES IL:WHILE (IL:SETQ SUBNODES (CDR TAIL))
    IL:DO (WHEN (EQ STRUCTURE (IL:|fetch| STRUCTURE IL:|of| (CAR SUBNODES)))
      (COND
        ((EQ NODE-TYPE (IL:FETCH NODE-TYPE IL:OF (CAR SUBNODES)))
          ;; we can re-use the node
          (RPLACD TAIL (CDR SUBNODES))
          (IL:REPLACE SUB-NODE-INDEX IL:OF (IL:SETQ SUBNODES (CAR SUBNODES))
            IL:WITH (IL:ADD1 (CAR (IL:SETQ TAIL (IL:FETCH SUB-NODES
              IL:OF (IL:FETCH CURRENT-NODE IL:OF CONTEXT
                )))))
            (IL:REPLACE \X IL:OF CONTEXT IL:WITH (IL:FETCH SUB-NODES IL:OF SUBNODES))
            (IL:REPLACE SUB-NODES IL:OF SUBNODES IL:WITH (LIST 0))
            (IL:NCONC1 TAIL SUBNODES)
            (RPLACA TAIL (IL:FETCH SUB-NODE-INDEX IL:OF SUBNODES))
            (RETURN SUBNODES))
          (T
            ;; it's changed type -- make this undoable
            (LET ((NEW-NODE (CREATE-PRELINEARIZED-NODE STRUCTURE (IL:FETCH CURRENT-NODE
              IL:OF CONTEXT)
                (IL:FETCH ENVIRONMENT IL:OF CONTEXT)
                NODE-TYPE STRING PRIN-2? FONT)))
              (UNDO-BY REPLACE-NODE NEW-NODE (CAR SUBNODES))
              (RETURN NEW-NODE))))
            (IL:SETQ TAIL SUBNODES)
            IL:FINALLY (IL:REPLACE \X IL:OF CONTEXT IL:WITH NIL)
              (RETURN (CREATE-PRELINEARIZED-NODE STRUCTURE (IL:FETCH CURRENT-NODE IL:OF CONTEXT)
                (IL:FETCH ENVIRONMENT IL:OF CONTEXT)
                NODE-TYPE STRING PRIN-2? FONT))))))

```

(CLOSE-NODE

```

(IL:LAMBDA (CONTEXT) ; Edited 6-Jul-87 20:48 by DCB
  (WHEN (IL:FETCH OPEN-NODE IL:OF CONTEXT)
    (IF (DEAD-NODE? (IL:FETCH OPEN-NODE IL:OF CONTEXT))
      (IL:REPLACE OPEN-NODE IL:OF CONTEXT IL:WITH NIL)
      (FUNCALL (IL:FETCH CLOSE-NODE IL:OF (IL:FETCH NODE-TYPE IL:OF (IL:FETCH OPEN-NODE IL:OF CONTEXT)))
        CONTEXT
        (IL:FETCH OPEN-NODE IL:OF CONTEXT))))
  (IL:REPLACE OPEN-NODE-CHANGED? IL:OF CONTEXT IL:WITH NIL)))

```

(COLLECT-UNDO-BLOCK

```

(IL:LAMBDA (CONTEXT) ; Edited 6-Jul-87 20:48 by DCB
  (LET ((BLOCK-START (IL:FETCH UNDO-LIST IL:OF CONTEXT)))
    (COND
      ((NULL BLOCK-START) ; empty undo list -- do nothing
        NIL)
      ((NULL (CAR BLOCK-START)) ; empty block -- throw it out
        (IL:REPLACE UNDO-LIST IL:OF CONTEXT IL:WITH (CDR BLOCK-START)))
      (T (IL:FOR (BLOCK-END IL:_ BLOCK-START) IL:BY (CDR BLOCK-END) IL:WHILE (CADR BLOCK-END)
        IL:EACHTIME (WHEN (NULL (CDR BLOCK-END)) ; no matching blip -- do nothing
          (RETURN))
        IL:FINALLY (COND
          ((EQ BLOCK-START BLOCK-END) ; one element block -- just remove the blip
            (RPLACD BLOCK-END (CDDR BLOCK-END)))

```

```
(T (IL:REPLACE UNDO-LIST IL:OF CONTEXT IL:WITH (CDR BLOCK-END))
  (RPLACD BLOCK-END NIL)
  (RPLACA (IL:FETCH UNDO-LIST IL:OF CONTEXT)
    BLOCK-START))))))
```

(COMPILE-STRUCTURE

(IL:LAMBDA (CONTEXT)

; Edited 5-Dec-90 18:05 by woz

;;; Compile the function being edited (if any). Return T if compilation returns OK, NIL otherwise.

```
(LET ((NAME (IL:fetch ICON-TITLE IL:of) CONTEXT))
  (TYPE (IL:fetch EDIT-TYPE IL:of) CONTEXT))
  (BODY (IL:fetch STRUCTURE IL:of) (CADR (IL:fetch SUB-NODES IL:of) (IL:fetch ROOT IL:of) CONTEXT))))
  (PW (OR (IL:OPENWP (GET-PROMPT-WINDOW CONTEXT))
    IL:PROMPTWINDOW)))
  (WHEN NAME (FORMAT PW "~%Compiling ~A defn of ~A..." TYPE NAME))
  (COND
    ((IL:ERSETQ (FUNCALL *COMPILE-FN* NAME TYPE BODY))
      (FORMAT PW "~%~A compiled." NAME)
      T)
    (T (FORMAT PW "~%Compilation of ~A failed." NAME)
      NIL))))))
```

(COMPUTE-ALL-FORMATS

(IL:LAMBDA (CONTEXT)

; Edited 6-Jul-87 20:48 by DCB

;;; assigns format types to each node in the tree top-down and width estimates to each node bottom-up. Avoids touching any node more than once and does not record changes.

```
(IL:REPLACE DONT-COLLECT-CHANGES? IL:OF CONTEXT IL:WITH T)
(COMPUTE-FORMATS-AND-FORMAT-VALUES (IL:FETCH ROOT IL:OF CONTEXT)
  CONTEXT)
(IL:REPLACE DONT-COLLECT-CHANGES? IL:OF CONTEXT IL:WITH NIL))
```

(COMPUTE-FORMATS-AND-FORMAT-VALUES

(IL:LAMBDA (NODE CONTEXT)

; Edited 6-Jul-87 20:48 by DCB

;;; computes format types and horizontal space estimates for the SEdit subtree rooted at node. assigns format type from root down, so that a node's format type can be based on the format type of its parent. computes format values depth first, so that a node's space estimates can be based on those of its children.

;;; efficiency note: if an assign format method changes the format, it will before returning cause the assign format methods of its subnodes to be run. this means that the assign format method will be run twice for some nodes.

```
(FUNCALL (IL:FETCH ASSIGN-FORMAT IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
  NODE CONTEXT (IL:FETCH FORMAT IL:OF NODE))
(IL:FOR SUBNODE IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE)) IL:DO (COMPUTE-FORMATS-AND-FORMAT-VALUES SUBNODE
  CONTEXT))
(FUNCALL (IL:FETCH COMPUTE-FORMAT-VALUES IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
  NODE
  (IL:FETCH ENVIRONMENT IL:OF CONTEXT)
  CONTEXT
  (IL:FETCH FORMAT IL:OF NODE))))
```

(COMPUTE-POINT-POSITION

(IL:LAMBDA (POINT CONTEXT)

; Edited 14-Jan-88 10:40 by DCB

;;; if there's a caret point, compute its coordinates. each node type has a method for this

;; if we get an error we throw the point away.

```
(WHEN (IL:TYPE? EDIT-NODE (IL:FETCH POINT-NODE IL:OF POINT))
  (IF (DEAD-NODE? (IL:FETCH POINT-NODE IL:OF POINT))
    (SET-POINT-NOWHERE POINT)
    (LET ((ERRVAL (IL:NLSETQ (FUNCALL (IL:FETCH COMPUTE-POINT-POSITION
      IL:OF (IL:FETCH NODE-TYPE IL:OF (IL:FETCH POINT-NODE IL:OF POINT))
      )
      POINT CONTEXT))))
      (UNLESS ERRVAL (SET-POINT-NOWHERE POINT))))))
```

(COMPUTE-SELECTION-POSITION

(IL:LAMBDA (SELECTION CONTEXT)

; Edited 14-Jan-88 10:42 by DCB

;;; if there's a current selection, compute its coordinates. each node has a method for this

;; if this errs out we throw away the selection

```
(WHEN (IL:FETCH SELECT-NODE IL:OF SELECTION)
  (COND
    ((DEAD-NODE? (IL:FETCH SELECT-NODE IL:OF SELECTION))
      (SET-SELECTION-NOWHERE SELECTION))
    ((IL:FETCH SELECT-START IL:OF SELECTION)
      (LET ((ERRVAL (IL:NLSETQ (FUNCALL (IL:FETCH COMPUTE-SELECTION-POSITION
      IL:OF (IL:FETCH NODE-TYPE IL:OF (IL:FETCH SELECT-NODE
```

```

                                                    IL:OF SELECTION)))
      (UNLESS ERRVAL (SET-SELECTION-NOWHERE SELECTION))))
(T (LET ((NODE (IL:FETCH SELECT-NODE IL:OF SELECTION)))
  (IL:REPLACE SELECT-START-X IL:OF SELECTION IL:WITH (IL:FETCH START-X IL:OF NODE))
  (IL:REPLACE SELECT-START-LINE IL:OF SELECTION IL:WITH (IL:FETCH FIRST-LINE IL:OF NODE))
  (IL:REPLACE SELECT-END-X IL:OF SELECTION IL:WITH (IL:IPLUS (IL:FETCH START-X IL:OF NODE)
    (IL:FETCH ACTUAL-LLENGTH IL:OF NODE)))
  (IL:REPLACE SELECT-END-LINE IL:OF SELECTION IL:WITH (IL:FETCH LAST-LINE IL:OF NODE))))))

```

(COMPUTE-SELECTION-POSITION-DEFAULT

(IL:LAMBDA (SELECTION CONTEXT) ; Edited 6-Jul-87 20:48 by DCB

;; a default ComputeSelectionPosition method for aggregate nodes. start and end values are assumed to be subnode indices, and the selection will extend from the beginning of the first selected subnode to the end of the last

```

(LET ((START (SUBNODE (IL:FETCH SELECT-START IL:OF SELECTION)
  (IL:FETCH SELECT-NODE IL:OF SELECTION)))
  END)
  (IL:SETQ END (IF (IL:FETCH SELECT-END IL:OF SELECTION)
    (SUBNODE (IL:FETCH SELECT-END IL:OF SELECTION)
      (IL:FETCH SELECT-NODE IL:OF SELECTION))
    START))
  (IL:REPLACE SELECT-START-X IL:OF SELECTION IL:WITH (IL:FETCH START-X IL:OF START))
  (IL:REPLACE SELECT-START-LINE IL:OF SELECTION IL:WITH (IL:FETCH FIRST-LINE IL:OF START))
  (IL:REPLACE SELECT-END-X IL:OF SELECTION IL:WITH (IL:IPLUS (IL:FETCH START-X IL:OF END)
    (IL:FETCH ACTUAL-LLENGTH IL:OF END)))
  (IL:REPLACE SELECT-END-LINE IL:OF SELECTION IL:WITH (IL:FETCH LAST-LINE IL:OF END))))

```

(CONTAINS?

(IL:LAMBDA (SELECTION-1 SELECTION-2) ; Edited 6-Jul-87 20:48 by DCB

;; check to see if the selection overlaps some or all of these nodes. if there's no overlap, return NIL. if it properly contains them, return T. otherwise return (QUOTE Overlap). (node1, start1, end1) and (node2, start2, end2) describe the two sequences of nodes

```

(LET ((NODE-1 (IL:FETCH SELECT-NODE IL:OF SELECTION-1))
  (START-1 (IL:FETCH SELECT-START IL:OF SELECTION-1))
  (END-1 (IL:FETCH SELECT-END IL:OF SELECTION-1))
  (NODE-2 (IL:FETCH SELECT-NODE IL:OF SELECTION-2))
  (START-2 (IL:FETCH SELECT-START IL:OF SELECTION-2))
  (END-2 (IL:FETCH SELECT-END IL:OF SELECTION-2)))
  (COND
    ((NULL START-1)
      (IL:SETQ START-1 (IL:SETQ END-1 (IL:FETCH SUB-NODE-INDEX IL:OF NODE-1)))
      (IL:SETQ NODE-1 (IL:FETCH SUPER-NODE IL:OF NODE-1)))
    ((NULL END-1)
      (IL:SETQ END-1 START-1)))
  (COND
    ((NULL START-2)
      (IL:SETQ START-2 (IL:SETQ END-2 (IL:FETCH SUB-NODE-INDEX IL:OF NODE-2)))
      (IL:SETQ NODE-2 (IL:FETCH SUPER-NODE IL:OF NODE-2)))
    ((NULL END-2)
      (IL:SETQ END-2 START-2)))

```

;; now we must get the selections at equal tree depth so we can compare bounds. First try to bring node2 up to depth of node1, then do it the other way. It doesn't matter which loop runs, the depths will end up equal.

```

(IL:WHILE (IL:ILESSP (IL:FETCH DEPTH IL:OF NODE-1)
  (IL:FETCH DEPTH IL:OF NODE-2))
  IL:DO (IL:SETQ START-2 (IL:SETQ END-2 (IL:FETCH SUB-NODE-INDEX IL:OF NODE-2)))
  (IL:SETQ NODE-2 (IL:FETCH SUPER-NODE IL:OF NODE-2)))

```

;; bring node 1 up to depth of node 2, in case the first loop was wrong

```

(IL:WHILE (IL:ILESSP (IL:FETCH DEPTH IL:OF NODE-2)
  (IL:FETCH DEPTH IL:OF NODE-1))
  IL:DO (IL:SETQ START-1 (IL:SETQ END-1 (IL:FETCH SUB-NODE-INDEX IL:OF NODE-1)))
  (IL:SETQ NODE-1 (IL:FETCH SUPER-NODE IL:OF NODE-1)))

```

;; and see if the selection contains the node2 sequence.

```

(COND
  ((OR (IL:NEQ NODE-1 NODE-2)
    (IL:ILESSP END-1 START-2)
    (IL:ILESSP END-2 START-1))
    ;; non-overlapping sisters
    NIL)
  (T ;; they do overlap. check if it's proper, otherwise return Unsafe
    (OR (AND (IL:ILEQ START-1 START-2)
      (IL:IGEQ END-1 END-2))
      'OVERLAP))))

```

(COPY-NODE

(IL:LAMBDA (NODE CONTEXT) ; Edited 6-Apr-88 16:42 by woz

;;; copy the subtree rooted at node

```
(LET ((NEW-NODE (IL:create| EDIT-NODE
  NODE-TYPE IL:_ (IL:fetch| NODE-TYPE IL:of| NODE)
  STRUCTURE IL:_ (IL:fetch| STRUCTURE IL:of| NODE)
  SUB-NODE-INDEX IL:_ (IL:fetch| SUB-NODE-INDEX IL:of| NODE)
  CHANGED? IL:_ T
  INLINE-WIDTH IL:_ (IL:fetch| INLINE-WIDTH IL:of| NODE)
  PREFERRED-WIDTH IL:_ (IL:fetch| PREFERRED-WIDTH IL:of| NODE)
  UNASSIGNED IL:_ (IL:fetch| UNASSIGNED IL:of| NODE)))
  (IL:replace| SUB-NODES IL:of| NEW-NODE IL:with| (CONS (CAR (IL:fetch| SUB-NODES IL:of| NODE))
    (IL:for| SUBNODE
      IL:in| (CDR (IL:fetch| SUB-NODES IL:of| NODE))
      IL:collect| (IL:SETQ SUBNODE (COPY-NODE SUBNODE
        CONTEXT))
      (IL:replace| SUPER-NODE IL:of| SUBNODE
        IL:with| NEW-NODE)
      SUBNODE)))
    ;; if this node type has no relinearization method, copy the linear form
    (COND
      ((IL:fetch| LINEARIZE IL:of| (IL:fetch| NODE-TYPE IL:of| NODE))
        (IL:replace| LINEAR-FORM IL:of| NEW-NODE IL:with| (CREATE-WEAK-LINK NEW-NODE)))
      (T (IL:replace| LINEAR-FORM IL:of| NEW-NODE IL:with| (IL:APPEND (IL:fetch| LINEAR-FORM IL:of| NODE)
        (CREATE-WEAK-LINK NEW-NODE)))
        (IL:replace| ACTUAL-WIDTH IL:of| NEW-NODE IL:with| (IL:fetch| ACTUAL-WIDTH IL:of| NODE))
        (IL:replace| ACTUAL-LENGTH IL:of| NEW-NODE IL:with| (IL:fetch| ACTUAL-LENGTH IL:of| NODE))))
    ;; the CopyStructure method will fill in the Structure field appropriately
    (FUNCCALL (IL:fetch| COPY-STRUCTURE IL:of| (IL:fetch| NODE-TYPE IL:of| NODE))
      NEW-NODE CONTEXT)
    NEW-NODE)))
```

(COPY-SELECTION

```
(IL:LAMBDA (SELECTION CONTEXT DESTINATION-CONTEXT POINT DELETE?)
  ; Edited 19-Nov-87 15:45 by DCB
```

;;; apply CopySelection method for the selected node, to copy or move the current selection

```
(IF (OR (NULL DESTINATION-CONTEXT)
  (IL:FETCH POINT-NODE IL:OF POINT))
  (COND
    ((AND DESTINATION-CONTEXT DELETE? (IL:TYPE? EDIT-SELECTION (IL:FETCH POINT-NODE IL:OF POINT))
      (CONTAINS? (IL:FETCH POINT-NODE IL:OF POINT)
        SELECTION))
      ;; this is a move selection into an overlapping pending delete selection. can't handle this case because deleting the selection to move
      ;; deletes some nodes out from under the pending delete selection, and then the selection is wrong. if we could fix up the selection in
      ;; this case (hard) we would be okay.
      (IL:|printout| (GET-PROMPT-WINDOW DESTINATION-CONTEXT)
        T "Can't move a structure which overlaps the selection.))
      (T (WHEN (EQ CONTEXT DESTINATION-CONTEXT)
        (START-UNDO-BLOCK))
        (FUNCCALL (IL:FETCH COPY-SELECTION IL:OF (IL:FETCH NODE-TYPE IL:OF (IL:FETCH SELECT-NODE IL:OF
          SELECTION
          )))
          SELECTION CONTEXT DESTINATION-CONTEXT POINT DELETE?)
        (WHEN (EQ CONTEXT DESTINATION-CONTEXT)
          ;; if we're moving within the same context, then we want the insert and possible delete to be grouped, so we need to close the
          ;; node inserted into (so changes will get recorded). Since we're in the same context, the correct profile closing is the current
          ;; profile.
          (CLOSE-OPEN-NODE CONTEXT)
          (END-UNDO-BLOCK)))
        (IL:|printout| (GET-PROMPT-WINDOW CONTEXT)
          T "Select a place to " (IF DELETE?
            "move"
            "copy")
            " to.))))))
```

(COPY-SELECTION-DEFAULT

```
(IL:LAMBDA (SELECTION CONTEXT DESTINATION POINT DELETE?) ; Edited 6-Jul-87 20:49 by DCB
```

;;; a simple copy selection method for aggregate nodes

```
(LET ((NODE (IL:FETCH SELECT-NODE IL:OF SELECTION))
  (START (IL:FETCH SELECT-START IL:OF SELECTION))
  (END (IL:FETCH SELECT-END IL:OF SELECTION))
  NODES)
  (COND
    (DESTINATION ; copying into an SEdit
      (WITH-PROFILE (IL:FETCH PROFILE IL:OF DESTINATION)
        (IL:SETQ NODES (IF (NULL START)
          (LIST NODE)
          (IL:FOR I IL:FROM START IL:TO (OR END START) IL:AS SUBNODE
```



```

'PARSE--BROKEN-ATOM
'IL:NEW-ATOM
'PARSE--LITATOM)
PARSE-INFO-UNKNOWN IL:_ 'PARSE--UNKNOWN
DEFAULT-FONT IL:_ (IL:FONTCREATE IL:DEFAULTFONT)
ITALIC-FONT IL:_ (IL:FONTCREATE IL:ITALICFONT)
KEYWORD-FONT IL:_ (IL:FONTCREATE IL:CLISPFONT)
COMMENT-FONT IL:_ (IL:FONTCREATE IL:COMMENTFONT)
BROKEN-ATOM-FONT IL:_ (IL:FONTCREATE IL:ITALICFONT)
SPACE-WIDTH IL:_ (IL:CHARWIDTH (IL:CHARCODE IL:SPACE)
(IL:FONTCREATE IL:DEFAULTFONT))
DEFAULT-LINE-SKIP IL:_ 2
COMMAND-TABLE IL:_ (CAR COMMANDS)
HELP-MENU IL:_ (CADR COMMANDS)
DEFAULT-CHAR-HANDLER IL:_ (IL:FUNCTION INPUT-NORMAL-CHAR)
EM-WIDTH IL:_ (IL:CHARWIDTH (IL:CHARCODE "M")
(IL:FONTCREATE IL:DEFAULTFONT))
INDENT-BASE IL:_ (IL:CHARWIDTH (IL:CHARCODE "M")
(IL:FONTCREATE IL:DEFAULTFONT))
INDENT-STEP IL:_ (IL:ITIMES 2 (IL:CHARWIDTH (IL:CHARCODE "M")
(IL:FONTCREATE
IL:DEFAULTFONT)))
MAX-WIDTH IL:_ 500
COMMENT-WIDTH-PERCENT IL:_ 40
INIT-COMMENT-SEPARATION IL:_ 15))

```

(CREATE-CONSTANT-STRINGS LISP-EDIT-ENVIRONMENT)))

(CREATE-GAP-NODE

(IL:LAMBDA (GAP) ; Edited 6-Apr-88 16:43 by woz

```

(LET* ((WIDTH (LINEAR-ITEM-WIDTH (IL:|fetch| LINEAR-ITEM IL:|of| GAP)))
(GAP-NODE (IL:|create| EDIT-NODE
NODE-TYPE IL:_ TYPE-GAP
STRUCTURE IL:_ GAP
SUB-NODES IL:_ (LIST 0)
INLINE-WIDTH IL:_ WIDTH
PREFERRED-WIDTH IL:_ WIDTH
ACTUAL-LENGTH IL:_ WIDTH
ACTUAL-WIDTH IL:_ WIDTH))
(IL:|replace| LINEAR-FORM IL:|of| GAP-NODE IL:|with| (CONS (IL:|fetch| LINEAR-ITEM IL:|of| GAP)
(CREATE-WEAK-LINK GAP-NODE)))
GAP-NODE)))

```

(CREATE-NODE

(IL:LAMBDA (STRUCTURE SUPER-NODE NODETYPE) ; Edited 6-Apr-88 16:44 by woz

;;; construct a new node and fit it into the tree

```

(LET ((NEW-NODE (IL:|create| EDIT-NODE
NODE-TYPE IL:_ NODETYPE
SUPER-NODE IL:_ SUPER-NODE
STRUCTURE IL:_ STRUCTURE
SUB-NODES IL:_ (LIST 0))))
(COND
(SUPER-NODE (IL:|replace| DEPTH IL:|of| NEW-NODE IL:|with| (IL:ADD1 (IL:|fetch| DEPTH IL:|of| SUPER-NODE)))
(IL:|replace| SUB-NODE-INDEX IL:|of| NEW-NODE IL:|with| (IL:ADD1 (CAR (IL:|fetch| SUB-NODES IL:|of|
SUPER-NODE
))))
(IL:NCONC1 (IL:|fetch| SUB-NODES IL:|of| SUPER-NODE)
NEW-NODE)
(RPLACA (IL:|fetch| SUB-NODES IL:|of| SUPER-NODE)
(IL:|fetch| SUB-NODE-INDEX IL:|of| NEW-NODE)))
(T (IL:|replace| DEPTH IL:|of| NEW-NODE IL:|with| 0)))
(IL:|replace| LINEAR-FORM IL:|of| NEW-NODE IL:|with| (CREATE-WEAK-LINK NEW-NODE))
NEW-NODE)))

```

(CREATE-PRELINEARIZED-NODE

(IL:LAMBDA (STRUCTURE SUPER-NODE ENVIRONMENT NODETYPE STRING PRIN-2? FONT) ; Edited 17-Nov-87 11:17 by DCB

;;; construct a new node and fit it into the tree. this node has a fixed linear form, given by string, prin2? and font, so use create.simple.node to construct
;;; it

```

(LET ((NEW-NODE (CREATE-SIMPLE-NODE STRUCTURE ENVIRONMENT NODETYPE STRING PRIN-2? FONT)))
(COND
((IL:REPLACE SUPER-NODE IL:OF NEW-NODE IL:WITH SUPER-NODE)
(IL:REPLACE DEPTH IL:OF NEW-NODE IL:WITH (IL:ADD1 (IL:FETCH DEPTH IL:OF SUPER-NODE)))
(IL:REPLACE SUB-NODE-INDEX IL:OF NEW-NODE IL:WITH (IL:ADD1 (CAR (IL:FETCH SUB-NODES IL:OF SUPER-NODE)
))))
(RPLACA (IL:FETCH SUB-NODES IL:OF SUPER-NODE)
(IL:FETCH SUB-NODE-INDEX IL:OF NEW-NODE))
(IL:NCONC1 (IL:FETCH SUB-NODES IL:OF SUPER-NODE)
NEW-NODE))
(T (IL:REPLACE DEPTH IL:OF NEW-NODE IL:WITH 0)))
NEW-NODE)))

```

(CREATE-PRETTY-PRINT-ENV

```

(IL:LAMBDA NIL
  (IL:SETQ PRETTY-PRINT-ENV (IL:CREATE EDIT-ENV IL:USING LISP-EDIT-ENVIRONMENT DEFAULT-FONT IL:_
    (IL:FONTCREATE IL:DEFAULTFONT NIL NIL NIL
      'IL:INTERPRESS)
    ITALIC-FONT IL:_ (IL:FONTCREATE IL:ITALICFONT NIL NIL
      NIL 'IL:INTERPRESS)
    KEYWORD-FONT IL:_ (IL:FONTCREATE IL:CLISPFONT NIL NIL
      NIL 'IL:INTERPRESS)
    COMMENT-FONT IL:_ (IL:FONTCREATE IL:COMMENTFONT NIL NIL
      NIL 'IL:INTERPRESS)
    BROKEN-ATOM-FONT IL:_ (IL:FONTCREATE IL:ITALICFONT NIL
      NIL NIL 'IL:INTERPRESS)
    SPACE-WIDTH IL:_ (IL:CHARWIDTH (IL:CHARCODE IL:SPACE)
      (IL:FONTCREATE IL:DEFAULTFONT
        NIL NIL NIL
        'IL:INTERPRESS))
    DEFAULT-LINE-SKIP IL:_ 0 INDENT-BASE IL:_
    (IL:FIXR (IL:TIMES IL:MICASPERPT (IL:FETCH INDENT-BASE
      IL:OF
      LISP-EDIT-ENVIRONMENT
      )))
    INDENT-STEP IL:_ (IL:FIXR (IL:TIMES IL:MICASPERPT
      (IL:FETCH INDENT-STEP
      IL:OF
      LISP-EDIT-ENVIRONMENT
      )))
    EM-WIDTH IL:_ (IL:FIXR (IL:TIMES IL:MICASPERPT
      (IL:FETCH EM-WIDTH
      IL:OF
      LISP-EDIT-ENVIRONMENT
      )))
    MAX-WIDTH IL:_ (IL:FIXR (IL:TIMES IL:MICASPERPT
      (IL:FETCH MAX-WIDTH
      IL:OF
      LISP-EDIT-ENVIRONMENT
      ))))
    (CREATE-CONSTANT-STRINGS PRETTY-PRINT-ENV)))

```

(CREATE-SIMPLE-NODE

```

(IL:LAMBDA (STRUCTURE ENVIRONMENT NODETYPE STRING PRIN-2? FONT)
  ; Edited 6-Apr-88 16:44 by woz

```

;;; construct a node with fixed linear form, given by string, prin2? and font.

```

(LET ((WIDTH (STRINGWIDTH STRING FONT PRIN-2?))
      NEW-NODE)
  (IL:SETQ NEW-NODE (IL:create| EDIT-NODE
    NODE-TYPE IL:_ NODETYPE
    STRUCTURE IL:_ STRUCTURE
    SUB-NODES IL:_ (LIST 0)
    INLINE-WIDTH IL:_ WIDTH
    PREFERRED-WIDTH IL:_ WIDTH
    ACTUAL-WIDTH IL:_ WIDTH
    ACTUAL-LENGTH IL:_ WIDTH))
  (IL:replace| LINEAR-FORM IL:of| NEW-NODE
    IL:with| (CONS (IL:create| STRING-ITEM
      STRING IL:_ STRING
      WIDTH IL:_ WIDTH
      FONT IL:_ FONT
      PRIN-2? IL:_ PRIN-2?)
      (CREATE-WEAK-LINK NEW-NODE)))
  NEW-NODE)))

```

(CREATE-STRING-ITEM

```

(IL:LAMBDA (STRING FONT)
  (IL:CREATE STRING-ITEM
    STRING IL:_ STRING
    WIDTH IL:_ (STRINGWIDTH STRING FONT)
    FONT IL:_ FONT
    PRIN-2? IL:_ NIL)))
; Edited 6-Jul-87 20:49 by DCB

```

(DEFAULT-COMPILE-FN

```

(IL:LAMBDA (NAME TYPE BODY)
  (CASE TYPE
    ((IL:FNS) (COMPILE NAME BODY))
    (T (COMPILE-FORM BODY))))
; Edited 31-Aug-87 11:59 by drc:

```

(DEFAULT-GETDEF-FN

```

(IL:LAMBDA (NAME TYPE OLD-DEF)
  (LET ((NEW-DEF (IL:GETDEF NAME TYPE NIL ' (IL:NOERROR))))
  ; Edited 26-Aug-87 10:09 by drc:

```

(OR NEW-DEF (PROGN (CERROR "Use the definition currently being edited." "No ~S definition for ~S" TYPE NAME) OLD-DEF))))))

(DEFAULT-PACKAGE

(IL:LAMBDA (NAME TYPE STRUCTURE)

; Edited 25-Aug-87 17:29 by drc:

;;; called by SETUP-PROFILE to determine what package to use for the edit

;;; We only look at name for now.

(IF (AND NAME (SYMBOLP NAME) (NOT (KEYWORDP NAME))) (SYMBOL-PACKAGE NAME) *PACKAGE*))

(DELETE-NODES

(IL:LAMBDA (NODE CONTEXT START END SET-POINT? STRING)

; Edited 17-Nov-87 11:18 by DCB

;;; delete a node or sequence of nodes. if SET-POINT?, change the caret point to be in the gap. the deletion is handled by the super of the nodes ;;; to be deleted.

(IF START (FUNCCALL (IL:FETCH DELETE IL:OF (IL:FETCH NODE-TYPE IL:OF NODE)) NODE CONTEXT START END SET-POINT? STRING) (FUNCCALL (IL:FETCH DELETE IL:OF (IL:FETCH NODE-TYPE IL:OF (IL:FETCH SUPER-NODE IL:OF NODE)) (IL:FETCH SUPER-NODE IL:OF NODE) CONTEXT NODE NIL SET-POINT?))))

(DETACH-NODE

(IL:LAMBDA (NODE)

; Edited 17-Nov-87 11:18 by DCB

;;; sever any connection between node and its old supernode, before it's inserted somewhere else

(IL:REPLACE LINEAR-THREAD IL:OF NODE IL:WITH NIL))

(FORMAT-VALUES-CHANGED

(IL:LAMBDA (NODE CONTEXT)

; Edited 6-Jul-87 20:49 by DCB

;;; recompute this nodes's width estimates, and check if any have changed

;;; if it's a litatom, we've been updating its width as we go along, so we can safely assume that it's changed. we won't call cvf.litatom if the node's still ;;; open and has been changed.

(COND ((EQ (IL:FETCH NODE-TYPE IL:OF NODE) TYPE-LITATOM) (WHEN (NOT (AND (EQ NODE (IL:FETCH OPEN-NODE IL:OF CONTEXT)) (IL:FETCH OPEN-NODE-CHANGED? IL:OF CONTEXT))) (FUNCCALL (IL:FETCH COMPUTE-FORMAT-VALUES IL:OF (IL:FETCH NODE-TYPE IL:OF NODE)) NODE (IL:FETCH ENVIRONMENT IL:OF CONTEXT) CONTEXT (IL:FETCH FORMAT IL:OF NODE)))) (T) (T (LET ((OLD-INLINE-WIDTH (IL:FETCH INLINE-WIDTH IL:OF NODE)) (OLD-PREFERRED-WIDTH (IL:FETCH PREFERRED-WIDTH IL:OF NODE))) (FUNCCALL (IL:FETCH COMPUTE-FORMAT-VALUES IL:OF (IL:FETCH NODE-TYPE IL:OF NODE)) NODE (IL:FETCH ENVIRONMENT IL:OF CONTEXT) CONTEXT (IL:FETCH FORMAT IL:OF NODE)) (OR (IL:NEQ OLD-INLINE-WIDTH (IL:FETCH INLINE-WIDTH IL:OF NODE)) (IL:NEQ OLD-PREFERRED-WIDTH (IL:FETCH PREFERRED-WIDTH IL:OF NODE))))))))

(GET-SELECTED-STRUCTURE

(IL:LAMBDA (CONTEXT)

; Edited 6-Jul-87 20:49 by DCB

;;; this is the guy who figures out what is selected for operations like eval and open. for now we only want to deal with single selection, not extended ;;; ones. Return NIL if it is an extended selection, or if there is no node selected.

(CLOSE-OPEN-NODE CONTEXT) (LET* ((SELECTION (IL:FETCH SELECTION IL:OF CONTEXT)) (NODE (IL:FETCH SELECT-NODE IL:OF SELECTION))) (AND NODE (NOT (IL:FETCH SELECT-START IL:OF SELECTION)) (IL:FETCH STRUCTURE IL:OF NODE))))

(HANDLE-COMPLETION

(IL:LAMBDA (CONTEXT REASON)

; Edited 25-Jan-91 13:52 by woz

;;; call the completion function. it is either a function or a list of the form (<fn> <extra args>*). The REASON arg will be :ABORT if the edit completes ;;; with an abort command, otherwise it is meaningless. The function is applied to CONTEXT, STRUCTURE, REASON, <extra args>, where ;;; STRUCTURE is the edited structure and REASON is NIL if no changes were made, T if changes were made, and :ABORT is user wants to abort

;; changes.

;; IDEALLY: The completion fn is called in the abort case with the structure including the changes, so that the edit interface could potentially implement "undo abort". But FOR NOW (1/13/91) this doesn't happen (see COMPLETE) because the changes have been undone by the time we get here.

```
(UNLESS (EQ REASON :ABORT)
  (SETQ REASON (IL:|fetch| CHANGED-STRUCTURE? IL:|of| CONTEXT)))
(THROW-AWAY-CHANGES CONTEXT) ; do this before completion-fn runs, so if markaschanged gets
                                ; called, it won't think they're still changes on this edit.
(LET ((FN (IL:|fetch| COMPLETION-FN IL:|of| CONTEXT))
      EXTRA-ARGS)
  (WHEN FN
    (WHEN (AND (LISTP FN)
               (NOT (MEMBER (FIRST FN)
                             '(LAMBDA IL:LAMBDA))))
      ;; catch the #' case by checking for lambda as the car. This is terrible, but cl:functionp returns T for any list, which is wrong, so
      ;; can't use it now.
      (SETQ EXTRA-ARGS (REST FN))
      (SETQ FN (FIRST FN)))
    (APPLY FN (LIST* CONTEXT (IL:|fetch| STRUCTURE IL:|of| (SUBNODE 1 (IL:|fetch| ROOT IL:|of| CONTEXT)))
             REASON EXTRA-ARGS))))))
```

(INITIALIZE

```
(IL:LAMBDA NIL ; Edited 16-Jul-87 15:26 by DCB
  (IL:PUSHNEW IL:MARKASCHANGEDFNS (IL:FUNCTION MARKASCHANGEDFN))
  (IL:CHANGENAME 'IL:EDITFERROR 'IL:COPY 'NEW-FUNCTION-BODY) ; set up SEdit's global variables: the standard environments,
                                                                ; node types, and terminal table
  (CREATE-ENVIRONMENTS)
```

```
(IL:SETQ TYPES (LIST (IL:SETQ TYPE-ROOT (IL:CREATE EDIT-NODE-TYPE
  NAME IL:_ 'ROOT
  ASSIGN-FORMAT IL:_ 'ASSIGN-FORMAT-NIL
  COMPUTE-FORMAT-VALUES IL:_ 'IL:NILL
  LINEARIZE IL:_ 'LINEARIZE-ROOT
  SUB-NODE-CHANGED IL:_ 'SUBNODE-CHANGED-ROOT
  COMPUTE-POINT-POSITION IL:_ 'IL:SHOULDNT
  COMPUTE-SELECTION-POSITION IL:_
  'COMPUTE-SELECTION-POSITION-DEFAULT
  SET-POINT IL:_ 'SET-POINT-NOWHERE
  SET-SELECTION IL:_ 'SET-SELECTION-NOWHERE
  GROW-SELECTION IL:_ 'IL:SHOULDNT
  SELECT-SEGMENT IL:_ 'SELECT-SEGMENT-DEFAULT
  INSERT IL:_ 'REPLACE-ROOT
  DELETE IL:_ 'IL:NILL
  COPY-STRUCTURE IL:_ 'IL:SHOULDNT
  COPY-SELECTION IL:_ 'COPY-SELECTION-DEFAULT
  BACK-SPACE IL:_ 'IL:SHOULDNT))
  (IL:SETQ TYPE-UNKNOWN (IL:CREATE EDIT-NODE-TYPE
  NAME IL:_ 'UNKNOWN
  ASSIGN-FORMAT IL:_ 'ASSIGN-FORMAT-NIL
  COMPUTE-FORMAT-VALUES IL:_ 'IL:NILL
  LINEARIZE IL:_ NIL
  SUB-NODE-CHANGED IL:_ 'IL:SHOULDNT
  COMPUTE-POINT-POSITION IL:_ 'IL:SHOULDNT
  COMPUTE-SELECTION-POSITION IL:_ 'IL:SHOULDNT
  SET-POINT IL:_ 'SET-POINT-UNKNOWN
  SET-SELECTION IL:_ 'SET-SELECTION-ME
  GROW-SELECTION IL:_ 'GROW-SELECTION-DEFAULT
  SELECT-SEGMENT IL:_ 'IL:SHOULDNT
  INSERT IL:_ 'IL:SHOULDNT
  DELETE IL:_ 'IL:SHOULDNT
  COPY-STRUCTURE IL:_ 'IL:NILL
  COPY-SELECTION IL:_ 'COPY-SELECTION-DEFAULT
  STRINGIFY IL:_ 'STRINGIFY-ATOM
  BACK-SPACE IL:_ 'BACKSPACE-UNKNOWN))
  (IL:SETQ TYPE-GAP (IL:CREATE EDIT-NODE-TYPE IL:USING TYPE-UNKNOWN NAME IL:_ 'GAP
  STRINGIFY IL:_ 'STRINGIFY-GAP
  BACK-SPACE IL:_ 'BACKSPACE-GAP))))
; these must be called after types has been created
```

```
(INITIALIZE-ATOMIC)
(INITIALIZE-LISTS)
(INITIALIZE-COMMENTS)
(IL:SETQ TERMINAL-TABLE (IL:COPYTERMTABLE))
(IL:FOR CLASS IL:IN '(IL:CHARDELETE IL:LINEDELETE IL:WORDDELETE IL:RETYPE IL:CTRLV IL:EOL)
  IL:DO (IL:FOR C IL:IN (IL:GETSYNTAX CLASS TERMINAL-TABLE) IL:DO (IL:SETSYNTAX C 'IL:NONE TERMINAL-TABLE)))
(IL:ECHOMODE NIL TERMINAL-TABLE)
(IL:CONTROL T TERMINAL-TABLE)
(IL:SETQ BASIC-GAP (IL:CREATE GAP
  LINEAR-ITEM IL:_ (CONS 0 GAP-BITMAP)))
(IL:SETQ ARGS-GAP (IL:CREATE GAP
  LINEAR-ITEM IL:_ (CONS 3 ARGS-BITMAP)))
(IL:SETQ BODY-GAP (IL:CREATE GAP
  LINEAR-ITEM IL:_ (CONS 3 BODY-BITMAP)))
;; initialize the selection state variables that used to be in the WINDOW file
(IL:SETQ PENDING-SELECTION (IL:CREATE EDIT-SELECTION))
```

```
(IL:SETQ INITIAL-SELECTION (IL:CREATE EDIT-SELECTION))
(IL:SETQ SCRATCH-SELECTION (IL:CREATE EDIT-SELECTION))
(IL:SETQ PENDING-CARET (IL:CREATE EDIT-POINT))
(IL:SETQ SELECTION-PENDING? NIL)
;; initialize the cache point and selection for replace.node
(IL:SETQ TEMP-POINT (IL:CREATE EDIT-POINT))
(IL:SETQ TEMP-SELECTION (IL:CREATE EDIT-SELECTION))
T))
```

(INSERT

(IL:LAMBDA (POINT CONTEXT SUBNODES) ; Edited 13-Jan-88 14:46 by DCB

;; insert handles a lot of different cases, translating where necessary to those handled by the method. point is a normal point or points to a
 ;; pending-delete selection. subnodes is a character or string of characters to be inserted, or a list of subnodes, or NIL (split). we massage the material
 ;; to be inserted according to the type of point. methods called to insert a list of subnodes return the list starting with the last subnode inserted, and we
 ;; automatically fix up the point and handle the uninserted nodes (if any).

```
(LET ((NODE (IL:FETCH POINT-NODE IL:OF POINT))
      (SELECTION (IL:FETCH SELECTION IL:OF CONTEXT))
      (CHARS WHERE PENDING-DELETE?)
      (WHEN (EQ (IL:FETCH POINT-TYPE IL:OF POINT)
                'STRUCTURE)
            (CLOSE-OPEN-NODE CONTEXT))
      (COND
       ((IL:TYPE? EDIT-NODE NODE)
        (IL:SETQ WHERE POINT))
       (T ;; the pending-delete case. the PointNode actually points to a selection framing the material to be replaced
          (IL:SETQ PENDING-DELETE? T)
          (IL:SETQ NODE (IL:FETCH SELECT-NODE IL:OF SELECTION))
          (COND
           ((IL:FETCH SELECT-START IL:OF SELECTION)
            (IL:SETQ WHERE SELECTION))
           (T (IL:SETQ WHERE NODE)
              (IL:SETQ NODE (IL:FETCH SUPER-NODE IL:OF NODE))))))
      (WHEN NODE
       (WHEN (IL:TYPE? EDIT-NODE SUBNODES)
        ;; coerce a single node to a list containing that node
        (IL:SETQ SUBNODES (LIST SUBNODES)))
        ;; make sure these nodes have been properly disconnected from whence they came
        (WHEN (IL:LISTP SUBNODES)
         (IL:FOR SUBNODE IL:IN SUBNODES IL:DO (DETACH-NODE SUBNODE)))
        (COND
         ((EQ (IL:FETCH POINT-TYPE IL:OF POINT)
              'STRUCTURE)
          ;; inserting/replacing at a structure point. inserting NIL does nothing, replacing with it deletes. if subnodes is a string, the
          ;; appropriate atom is constructed. use the value returned by the method to set the point and try again with any leftovers
          (COND
           ((NULL SUBNODES)
            (WHEN PENDING-DELETE?
             (DELETE-NODES NODE CONTEXT (IL:FETCH SELECT-START IL:OF SELECTION)
                            (IL:FETCH SELECT-END IL:OF SELECTION)
                            POINT)))
           ((IL:NLISTP SUBNODES)
            ;; insert the atom NIL, and then replace its characters with the character typed
            (LET ((NEW-NODE (CREATE-SIMPLE-NODE NIL (IL:FETCH ENVIRONMENT IL:OF CONTEXT)
                                                TYPE-LITATOM NIL T (IL:FETCH DEFAULT-FONT
                                                IL:OF (IL:FETCH ENVIRONMENT IL:OF CONTEXT))))))
              (FUNCALL (IL:FETCH INSERT IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
                       NODE CONTEXT WHERE (LIST NEW-NODE)
                       POINT)
              ;; this is a little nasty -- we don't want to bother figuring out the printname of the node, so we tell replace.char it was
              ;; "" but before we do that, we've got to make the inlinewidth agree so that when replace.chars adjusts it it comes
              ;; out right
              (IL:REPLACE INLINE-WIDTH IL:OF NEW-NODE IL:WITH 0)
              (REPLACE-STRING NEW-NODE CONTEXT 1 0 (OR (IL:STRINGP SUBNODES)
                                                         (TRANSLATE-CHARS (IL:MKSTRING SUBNODES)
                                                         'ATOM
                                                         (EQ *PRINT-CASE* 'UPCASE)))
              POINT "" 'ATOM)))
           (T ;; keep trying to insert these nodes and placing the point after them until we run out of nodes or run out of places to put
              ;; them
              (IL:DO (IL:SETQ SUBNODES (FUNCALL (IL:FETCH INSERT IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
                                                NODE CONTEXT WHERE SUBNODES POINT))
                     (IL:REPEATWHILE (AND (SETQ NODE (IL:FETCH POINT-NODE IL:OF (SETQ WHERE POINT)))
                                           SUBNODES))))))
         (T ;; inserting/replacing at an atom or string point. if it's characters, insert them; otherwise split, and if there were any subnodes
           ;; insert them
```

```
(FUNCALL (IL:FETCH INSERT IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
         NODE CONTEXT WHERE (AND (IL:NLISTP SUBNODES)
                                 SUBNODES)
         POINT)
(WHEN (AND (IL:LISTP SUBNODES)
           (IL:FETCH POINT-NODE IL:OF POINT))
      (INSERT POINT CONTEXT SUBNODES))))
;; the copy selection methods rely on there being no selection after an insert. the copy selection methods can't take care of this
;; because sometimes the selection gets set (eg from moving out of a quote.
(SET-SELECTION-NOWHERE SELECTION))))
```

(INSERT-CHANGED

(IL:LAMBDA (NODE LIST) ; Edited 6-Jul-87 20:50 by DCB

;; inserts node into list (but not before the first element) such that list is kept in decreasing order of depth

```
(IL:BIND NEXT (DEPTH IL:_ (IL:FETCH DEPTH IL:OF NODE)) IL:WHILE (AND (IL:SETQ NEXT (CDR LIST))
                                                                    (IL:IGREATERP (IL:FETCH DEPTH
                                                                    IL:OF (CAR NEXT))
                                                                    DEPTH))
              IL:DO (IL:SETQ LIST NEXT) IL:FINALLY (RPLACD LIST (CONS NODE NEXT))))
```

(KILL-NODE

(IL:LAMBDA (NODE) ; Edited 17-Nov-87 11:18 by DCB

;; the subtree rooted at this node is being deleted. mark all the nodes as dead, and cut the first line/last line pointers to avoid confusion.

```
(IL:REPLACE DEPTH IL:OF NODE IL:WITH 0)
(IL:REPLACE FIRST-LINE IL:OF NODE IL:WITH NIL)
;; used to replace LastLineLinear of node with NIL
(IL:REPLACE LAST-LINE IL:OF NODE IL:WITH NIL)
(IL:REPLACE START-X IL:OF NODE IL:WITH 0)
(IL:REPLACE LINEAR-THREAD IL:OF NODE IL:WITH NIL)
(IL:FOR X IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE)) IL:DO (KILL-NODE X)))
```

(LINEARIZE-ROOT

(IL:LAMBDA (NODE CONTEXT INDEX) ; Edited 6-Jul-87 20:50 by DCB

```
(IF INDEX
  (IL:SHOULDNT "can't be within the root node")
  (LINEARIZE (SUBNODE 1 NODE)
             CONTEXT)))
```

(NEXT-NODE

(IL:LAMBDA (NODE INDEX POSTORDER?) ; Edited 11-Dec-87 11:38 by DCB

;; step to the next node after this one (in preorder unless postorder? given). if index is a fixp, start with the next subnode after the one with that index. if it's T, start with the first node after this node. if it's NIL, start with this node's first subnode

```
(OR (AND (NULL INDEX)
         (SUBNODE 1 NODE))
     (IL:FIRST (OR INDEX (IL:SETQ INDEX 0)) IL:DO (WHEN (AND (IL:FIXP INDEX)
                                                            (IL:ILESSP INDEX (CAR (IL:FETCH SUB-NODES
                                                            IL:OF NODE))))
              (RETURN (SUBNODE (IL:ADD1 INDEX)
                              NODE)))
          (WHEN (AND POSTORDER?)
              (IL:SETQ INDEX (IL:FETCH SUB-NODE-INDEX IL:OF NODE))
              (IL:SETQ NODE (IL:FETCH SUPER-NODE IL:OF NODE))
              IL:REPEATWHILE NODE))))
```

(NOTE-CHANGE

(IL:LAMBDA (NODE CONTEXT) ; Edited 6-Jul-87 20:50 by DCB

;; this routine should be called whenever we make a structural change to node. clobber any clisp translation, and insert it into the ChangedNodes list.
;; the ChangedNodes list is kept sorted by increasing depth. this is because the ChangedNodes list will next be used to recompute format types, which
;; must be propagated top-down.

```
(WHEN (NOT (IL:FETCH CHANGED? IL:OF NODE))
      (IL:FOR (SUPER IL:_ NODE) IL:BY (IL:FETCH SUPER-NODE IL:OF SUPER) IL:WHILE SUPER
            IL:WHEN (IL:LISTP (IL:FETCH STRUCTURE IL:OF SUPER)) IL:DO (ZAP-CLISP-TRANSLATION (IL:FETCH STRUCTURE
            IL:OF SUPER)))
      (IL:REPLACE CHANGED? IL:OF NODE IL:WITH T)
      (WHEN (NOT (IL:FETCH DONT-COLLECT-CHANGES? IL:OF CONTEXT))
            (INSERT-CHANGED NODE (IL:FETCH CHANGED-NODES IL:OF CONTEXT)))
      (IL:REPLACE CHANGED-STRUCTURE? IL:OF CONTEXT IL:WITH T)))
```

(NOTE-CHANGE-FORMAT

(IL:LAMBDA (NODE CONTEXT) ; Edited 6-Jul-87 20:50 by DCB

;;; this routine should be called whenever we make a node's format changes. inserts the node should be inserted in the ChangedNodes list. this list will
;;; include nodes whose structure has changed and nodes whose format type has changed. it is kept sorted by decreasing depth. this is because the
;;; ChangedFormatNodes list will next be used to recompute width estimates, which must be propagated bottom-up.

;;; note that the ChangedNodes list should initially contain the reverse of the old ChangedNodes list (as left behind after noting all structure changes).

```
(WHEN (NOT (IL:FETCH CHANGED? IL:OF NODE))
 (IL:REPLACE CHANGED? IL:OF NODE IL:WITH T)
 (WHEN (NOT (IL:FETCH DONT-COLLECT-CHANGES? IL:OF CONTEXT))
 (INSERT-CHANGED NODE (IL:FETCH CHANGED-NODES IL:OF CONTEXT))))))
```

(NOTE-CHANGE-IN-SIMPLE

```
(IL:LAMBDA (NODE CONTEXT OFFSET WIDTH START END) ; Edited 6-Jul-87 20:50 by DCB
```

;;; we've changed a prelinearized node. fix up the linear form and width estimates, and notify its super

```
(LET ((TEMP (CAR (IL:FETCH LINEAR-FORM IL:OF NODE)))
 NEW-WIDTH)
 (IL:REPLACE STRING IL:OF TEMP IL:WITH (IL:FETCH STRUCTURE IL:OF NODE)) ; read table specific
 (IL:SETQ NEW-WIDTH (STRINGWIDTH (IL:FETCH STRING IL:OF TEMP)
 (IL:FETCH FONT IL:OF TEMP)
 (IL:FETCH PRIN-2? IL:OF TEMP)))
 (IL:REPLACE WIDTH IL:OF TEMP IL:WITH NEW-WIDTH)
 (IL:REPLACE INLINE-WIDTH IL:OF NODE IL:WITH NEW-WIDTH)
 (IL:REPLACE PREFERRED-WIDTH IL:OF NODE IL:WITH NEW-WIDTH)
 (IL:REPLACE ACTUAL-WIDTH IL:OF NODE IL:WITH NEW-WIDTH)
 (IL:REPLACE ACTUAL-LENGTH IL:OF NODE IL:WITH NEW-WIDTH)
 (IL:REPLACE CHANGED? IL:OF NODE IL:WITH T)
 (NOTE-CHANGE (IL:FETCH SUPER-NODE IL:OF NODE)
 CONTEXT)))
```

(PARSE

```
(IL:LAMBDA (STRUCTURE CONTEXT PARSER DATA) ; Edited 22-Jun-90 10:37 by narusawa
```

;;; construct the SEdit tree for this structure.

;;; if a node needs to be parsed as a particular kind of structure (for example, the second child of a lambda-list must be parsed as a list, even if it is NIL),
;;; you can specify a particular parsing function.

;;; parse used to compute the width estimates, but it can no longer do that because the new assign format pass must intervene between parsing and
;;; computing width estimates, and the assign format pass, since it is top down, and you can't run it until the lower nodes have been created, cannot
;;; conveniently be intermingled with the parsing pass.

```
(LET ((SUPER-NODE (IL:|fetch| CURRENT-NODE IL:|of| CONTEXT))
 (ENVIRONMENT (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT))
 (REUSE-NODES (IL:|fetch| \\X IL:|of| CONTEXT))
 THIS-NODE)
 (FUNCCALL (OR PARSER (IL:LISTGET (IL:|fetch| PARSE-INFO IL:|of| ENVIRONMENT)
 (IL:TYPE-NAME STRUCTURE))
 (IL:FETCH PARSE-INFO-UNKNOWN IL:OF ENVIRONMENT))
 STRUCTURE CONTEXT DATA)
 (IL:SETQ THIS-NODE (IL:|fetch| CURRENT-NODE IL:|of| CONTEXT))
 (WHEN (CDR (IL:|fetch| \\X IL:|of| CONTEXT))
 ;; some of the old nodes weren't reused (ie deleted). so mark this node changed, and kill the old ones.
 (IL:|replace| CHANGED? IL:|of| THIS-NODE IL:|with| T)
 (KILL-NODE (CADR (IL:|fetch| \\X IL:|of| CONTEXT))))
 (WHEN (OR (IL:|fetch| CHANGED? IL:|of| THIS-NODE)
 (AND (EQ (IL:|fetch| START-X IL:|of| THIS-NODE)
 0)
 REUSE-NODES))
 ;; or first case: this node was reused and it changed
 ;; or second case: this node is new but our super node is being reused
 ;; in either case the super-node has changed
 (IL:|replace| CHANGED? IL:|of| SUPER-NODE IL:|with| T))
 (IL:|replace| \\X IL:|of| CONTEXT IL:|with| REUSE-NODES)
 (IF SUPER-NODE
 (IL:|replace| CURRENT-NODE IL:|of| CONTEXT IL:|with| SUPER-NODE)
 (IL:FETCH CURRENT-NODE IL:OF CONTEXT))))))
```

(PARSE--GAP

```
(IL:LAMBDA (STRUCTURE CONTEXT) ; Edited 6-Apr-88 16:45 by woz
```

;;; parse a gap structure (presumably left there by a previous editing session)

```
(BUILD-NODE STRUCTURE CONTEXT TYPE-GAP)
(LET ((NEW-NODE (IL:|fetch| CURRENT-NODE IL:|of| CONTEXT))
 (WIDTH (LINEAR-ITEM-WIDTH (IL:|fetch| LINEAR-ITEM IL:|of| STRUCTURE))))
 (IL:|replace| LINEAR-FORM IL:|of| NEW-NODE IL:|with| (CONS (IL:|fetch| LINEAR-ITEM IL:|of| STRUCTURE)
 (CREATE-WEAK-LINK NEW-NODE)))
 (IL:|replace| INLINE-WIDTH IL:|of| NEW-NODE IL:|with| WIDTH))
```



```
(IL:replace PREFERRED-WIDTH IL:of NEW-NODE IL:with WIDTH)
(IL:replace ACTUAL-WIDTH IL:of NEW-NODE IL:with WIDTH)
(IL:replace ACTUAL-LLENGTH IL:of NEW-NODE IL:with WIDTH)))
```

(PARSE--UNKNOWN

```
(IL:LAMBDA (STRUCTURE CONTEXT) ; Edited 23-Jun-88 12:42 by Snow
;; this is the default parser for structures of an unknown type. they display in italics, and can't be edited
(BUILD-PRELINEARIZED-NODE STRUCTURE CONTEXT TYPE-UNKNOWN STRUCTURE T (IL:fetch COMMENT-FONT
IL:of (IL:fetch ENVIRONMENT
IL:of CONTEXT))))))
```

(PARSE-NEW

```
(IL:LAMBDA (EXPRESSION CONTEXT) ; Edited 6-Jul-87 20:50 by DCB
(IL:REPLACE CURRENT-NODE IL:OF CONTEXT IL:WITH NIL)
(IL:REPLACE \\X IL:OF CONTEXT IL:WITH NIL)
(PARSE EXPRESSION CONTEXT)))
```

(PROPAGATE-WIDTH-CHANGE

```
(IL:LAMBDA (CONTEXT NODE OLD-WIDTH) ; Edited 17-Nov-87 11:19 by DCB
```

;;; the width of node has been recomputed. this may change the width of some of its super nodes, and possibly even the extent recorded with the
;;; window

```
(LET ((WIDTH (IL:FETCH ACTUAL-WIDTH IL:OF NODE))
(SUPER (IL:FETCH SUPER-NODE IL:OF NODE))
SUPER-WIDTH NEW-WIDTH)
(COND
((EQ WIDTH OLD-WIDTH)
NIL)
(NULL SUPER)
(IL:REPLACE IL:WIDTH IL:OF (IL:WINDOWPROP (IL:FETCH DISPLAY-WINDOW IL:OF CONTEXT)
'IL:EXTENT)
IL:WITH WIDTH))
((IL:FETCH SUPER-NODE IL:OF SUPER)
(IL:SETQ SUPER-WIDTH (IL:FETCH ACTUAL-WIDTH IL:OF SUPER))
(IL:SETQ NEW-WIDTH (IL:IPLUS WIDTH (IL:IDIFFERENCE (IL:FETCH START-X IL:OF NODE)
(IL:FETCH START-X IL:OF SUPER))))))
(COND
((IL:IGEQ NEW-WIDTH SUPER-WIDTH)
(IL:REPLACE ACTUAL-WIDTH IL:OF SUPER IL:WITH NEW-WIDTH)
(PROPAGATE-WIDTH-CHANGE CONTEXT SUPER SUPER-WIDTH))
((EQ SUPER-WIDTH (IL:IPLUS OLD-WIDTH (IL:IDIFFERENCE (IL:FETCH START-X IL:OF NODE)
(IL:FETCH START-X IL:OF SUPER))))))
(RECOMPUTE-WIDTH SUPER)
(PROPAGATE-WIDTH-CHANGE CONTEXT SUPER SUPER-WIDTH))))
(T (IL:REPLACE ACTUAL-WIDTH IL:OF SUPER IL:WITH WIDTH)
(IL:REPLACE IL:WIDTH IL:OF (IL:WINDOWPROP (IL:FETCH DISPLAY-WINDOW IL:OF CONTEXT)
'IL:EXTENT)
IL:WITH WIDTH))))))
```

(RECOMPUTE-WIDTH

```
(IL:LAMBDA (NODE) ; Edited 17-Nov-87 11:19 by DCB
```

;;; determine the width of this node. we keep it fast by being tricky and skipping over subnodes which span several lines (we assume their widths are
;;; correct) so it isn't pretty

```
(PROG ((LINE (IL:FETCH FIRST-LINE IL:OF NODE))
(LAST-LINE (IL:FETCH LAST-LINE IL:OF NODE))
(STARTX (IL:FETCH START-X IL:OF NODE))
WIDTH)
(WHEN (EQ LINE LAST-LINE)
(IL:SHOULDNT "trying to recompute width of an inline node"))
(IL:SETQ WIDTH (IL:IPLUS STARTX (IL:FETCH ACTUAL-LLENGTH IL:OF NODE)))
NEXT-LINE
(IL:SETQ WIDTH (IL:IMAX WIDTH (IL:FETCH LINE-LENGTH IL:OF LINE)))
(WHEN (EQ (IL:SETQ LINE (CAR (IL:FETCH NEXT-LINE IL:OF LINE)))
LAST-LINE)
(GO DONE))
(WHEN (EQ (IL:FETCH NODE IL:OF LINE)
NODE)
(GO NEXT-LINE))
(IL:SETQ LINE (IL:FETCH NODE IL:OF LINE))
(IL:SETQ WIDTH (IL:IMAX WIDTH (IL:IPLUS (IL:FETCH START-X IL:OF LINE)
(IL:FETCH ACTUAL-WIDTH IL:OF LINE))))))
(WHEN (IL:NEQ (IL:SETQ LINE (IL:FETCH LAST-LINE IL:OF LINE))
LAST-LINE)
(GO NEXT-LINE))
DONE
(IL:REPLACE ACTUAL-WIDTH IL:OF NODE IL:WITH (IL:IDIFFERENCE WIDTH STARTX))))))
```

(RELINERIZE-WHERE-NECESSARY

(IL:LAMBDA (CONTEXT) ; Edited 6-Jul-87 20:50 by DCB

;;; we've made some changes to the structure; now it's time to fix up the linear forms and the window. first we recompute the format types top down for
;;; all appropriate nodes. then we recompute the width estimates for all appropriate nodes bottom up. given this, we find a minimal set of nodes to
;;; relinearize (such that each changed node is a subnode of one of these nodes, and the structure and format type and width estimates for these nodes
;;; have not changed)

;;; mdd 4/24/87: as an experiment, i modified this function to always simply relinearize from the top of the tree (by imbedding (format.values.changed ...)
;;; in (OR (format.values.changed ...) T)). this has the potential for considerably simplifying some of the linearizer. unfortunately, it caused very
;;; noticeable performance degradation while editing large structures. oh well.

```
(LET ((CHANGED-NODES (CDR (IL:FETCH CHANGED-NODES IL:OF CONTEXT)))
      NODES-TO-RELINERIZE)
  (COND
   ((NULL CHANGED-NODES)
    NIL)
   ((AND (EQ (IL:FETCH DEPTH IL:OF (CAR CHANGED-NODES))
             1)
        (NULL (CDR CHANGED-NODES)))
    ;; special case for editing a tree of just one node
    (RELINERIZE (CAR CHANGED-NODES)
                CONTEXT)
    (IL:REPLACE CHANGED? IL:OF (CAR CHANGED-NODES) IL:WITH NIL))
   (T ;; collect nodes whose structure has changed or whose format type has changed
    (IL:FOR NODE IL:IN (IL:REVERSE CHANGED-NODES) IL:BIND SUPER-NODE IL:WHEN (NOT (DEAD-NODE? NODE))
      IL:DO ;; call format method on node and its super (super first). mark the super as changed so that it will be relinearized.
        (IL:SETQ SUPER-NODE (IL:FETCH SUPER-NODE IL:OF NODE))
        (WHEN (AND SUPER-NODE (NOT (DEAD-NODE? SUPER-NODE))
                  (NOT (IL:FETCH CHANGED? IL:OF SUPER-NODE)))
          (IL:FETCH SUPER-NODE IL:OF SUPER-NODE))
          (FUNCCALL (IL:FETCH ASSIGN-FORMAT IL:OF (IL:FETCH NODE-TYPE IL:OF SUPER-NODE))
                   SUPER-NODE CONTEXT (IL:FETCH FORMAT IL:OF SUPER-NODE))
          (NOTE-CHANGE-FORMAT SUPER-NODE CONTEXT))
          (FUNCCALL (IL:FETCH ASSIGN-FORMAT IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
                   NODE CONTEXT (IL:FETCH FORMAT IL:OF NODE)))
        ;; collect nodes whose structure has changed or whose format type has changed or whose width formats have changed: in short, all
        ;; the nodes which might need relinearizing.
        (IL:SETQ CHANGED-NODES (CDR (IL:FETCH CHANGED-NODES IL:OF CONTEXT)))
        (IL:WHILE CHANGED-NODES IL:BIND NODE SUPER-NODE
          IL:DO (IL:SETQ NODE (CAR CHANGED-NODES))
                (WHEN (NOT (DEAD-NODE? NODE))
                  (COND
                   ((AND (IL:SETQ SUPER-NODE (IL:FETCH SUPER-NODE IL:OF NODE))
                        (FORMAT-VALUES-CHANGED NODE CONTEXT)
                        (IL:FETCH SUPER-NODE IL:OF SUPER-NODE))
                    ;; climb up the super node links until the width estimates stop changing
                    (WHEN (NOT (IL:FETCH CHANGED? IL:OF SUPER-NODE))
                      (IL:REPLACE CHANGED? IL:OF SUPER-NODE IL:WITH T)
                      (INSERT-CHANGED SUPER-NODE CHANGED-NODES)))
                   (T (IL:PUSH NODES-TO-RELINERIZE NODE))))
                  (IL:SETQ CHANGED-NODES (CDR CHANGED-NODES)))
          (IL:FOR NODE IL:IN IL:OLD (IL:SETQ NODES-TO-RELINERIZE (IL:DREVERSE NODES-TO-RELINERIZE))
            IL:DO (IF (AND (CDR NODES-TO-RELINERIZE)
                          (IL:FOR (SUPER-NODE IL:_ NODE) IL:WHILE (IL:SETQ SUPER-NODE
                                                                              (IL:FETCH SUPER-NODE IL:OF SUPER-NODE))
                                                                              (IL:THEREIS (IL:FETCH CHANGED? IL:OF SUPER-NODE)))
                        (IL:FOR (SUPER-NODE IL:_ NODE) IL:UNTIL (IL:FETCH CHANGED?
                                                                    IL:OF (IL:SETQ SUPER-NODE
                                                                              (IL:FETCH SUPER-NODE IL:OF SUPER-NODE))
                                                                    ))
                      IL:DO (IL:REPLACE CHANGED? IL:OF SUPER-NODE IL:WITH T))
                    (RELINERIZE NODE CONTEXT))))))
    (RPLACD (IL:FETCH CHANGED-NODES IL:OF CONTEXT)
            NIL)))
```

(REPLACE-NODE

(IL:LAMBDA (CONTEXT NODE NEW-NODE) ; Edited 6-Jul-87 20:50 by DCB

;;; replace node with new.node, without changing current selection or point

```
(LET* ((POINT (IL:FETCH CARET-POINT IL:OF CONTEXT))
       (SELECTION (IL:FETCH SELECTION IL:OF CONTEXT))
       (SELECT-NODE (IL:FETCH SELECT-NODE IL:OF SELECTION)))
  (SMASH-USING EDIT-POINT TEMP-POINT POINT)
  (SMASH-USING EDIT-SELECTION TEMP-SELECTION SELECTION)
  (SET-SELECTION-ME SELECTION CONTEXT NODE)
  (PENDING-DELETE POINT SELECTION)
  (INSERT POINT CONTEXT NEW-NODE)
  (SMASH-USING EDIT-POINT POINT TEMP-POINT)
  (COND
   ((AND SELECT-NODE (DEAD-NODE? SELECT-NODE))
```

```
(IL:REPLACE SELECT-NODE IL:OF SELECTION IL:WITH NEW-NODE)
(IL:REPLACE SELECT-START IL:OF SELECTION IL:WITH NIL)
(IL:REPLACE SELECT-END IL:OF SELECTION IL:WITH NIL)
(IL:REPLACE SELECT-TYPE IL:OF SELECTION IL:WITH 'STRUCTURE)
(IL:REPLACE PENDING-DELETE? IL:OF SELECTION IL:WITH NIL))
(T (SMASH-USING EDIT-SELECTION SELECTION TEMP-SELECTION))))))
```

(REPLACE-ROOT

(IL:LAMBDA (ROOT CONTEXT WHERE SUBNODES POINT) ; Edited 11-Apr-88 17:02 by woz

;; Insert method for the root. If we're passed a single node to insert, replace the root with it. If we're passed several nodes, then replace the root with an empty list, set the point in that list, and return the subnodes in toto so that insert will then insert them into the new (empty) root list.

```
(WHEN (AND (NULL POINT)
           (REST SUBNODES))
      (IL:SHOULDNT "Replacing root with list but no point specified!"))
(UNDO-BY UNDO-REPLACE-ROOT ROOT (SUBNODE 1 ROOT))
(LET ((TOP-NODE (IF (REST SUBNODES)
                   (CREATE-NULL-LIST CONTEXT)
                   (CAR SUBNODES))))
      (KILL-NODE (SUBNODE 1 ROOT))
      (RPLACA (CDR (IL:|fetch| SUB-NODES IL:|of| ROOT))
              TOP-NODE)
      (RPLACA (CDR (IL:|fetch| LINEAR-FORM IL:|of| ROOT))
              (CREATE-WEAK-LINK TOP-NODE))
      (WHEN (IL:|fetch| INLINE? IL:|of| TOP-NODE)
            ;; used to be (IL:REPLACE LAST-LINE-LINEAR IL:OF SUBNODE IL:WITH (IL:FETCH LINEAR-FORM IL:OF NODE))
            (IL:|replace| LAST-LINE IL:|of| TOP-NODE IL:|with| (CAR (IL:|fetch| LINEAR-FORM IL:|of| ROOT))))
      (IL:|replace| FIRST-LINE IL:|of| TOP-NODE IL:|with| (CAR (IL:|fetch| LINEAR-FORM IL:|of| ROOT)))
      (IL:|replace| LINEAR-THREAD IL:|of| TOP-NODE IL:|with| (CDR (IL:|fetch| LINEAR-FORM IL:|of| ROOT)))
      (IL:|replace| SUPER-NODE IL:|of| TOP-NODE IL:|with| ROOT)
      (IL:|replace| SUB-NODE-INDEX IL:|of| TOP-NODE IL:|with| 1)
      (SET-DEPTH TOP-NODE (IL:ADD1 (IL:|fetch| DEPTH IL:|of| ROOT)))
      (NOTE-CHANGE ROOT CONTEXT)
      (SUBNODE-CHANGED-ROOT ROOT TOP-NODE CONTEXT)
      (COND
        ((REST SUBNODES)
         ;; set the point ourself since set-point-list will blow out assuming that the list already has a linear form.
         (IL:|replace| POINT-NODE IL:|of| POINT IL:|with| TOP-NODE)
         (IL:|replace| POINT-INDEX IL:|of| POINT IL:|with| 0)
         (IL:|replace| POINT-TYPE IL:|of| POINT IL:|with| 'STRUCTURE)
         SUBNODES)
        (POINT (SET-POINT-NOWHERE POINT)
              NIL))))))
```

(REVIVE-NODE

(IL:LAMBDA (NODE DEPTH) ; Edited 6-Jul-87 20:50 by DCB

```
(IL:REPLACE DEPTH IL:OF NODE IL:WITH (IL:SETQ DEPTH (IL:IPLUS 1 DEPTH)))
(IL:FOR SUBNODE IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE)) IL:DO (REVIVE-NODE SUBNODE DEPTH))))
```

(SEDIT1

(IL:LAMBDA (CONTEXT) ; Edited 5-Dec-90 18:56 by woz

;; this is the function that runs in the sedit process. first finish the initialization that wasn't done in the calling process, then start the main loop. The read-print profile is rebound specially here, so global changes won't affect existing SEDits.

```
(WHEN (IL:NEQ (IL:|fetch| CONTEXT-LOCK IL:|of| CONTEXT)
            'DEAD)
      ;; this SEdit is okay, or new
      (WITH-PROFILE
        (IL:|fetch| PROFILE IL:|of| CONTEXT)
        (SETUP-CONTEXT CONTEXT)
        (SETUP-WINDOW-AND-PROCESS CONTEXT)
        ;; SEDIT (in start-process) is waiting for the initialization to complete before returning
        (IL:NOTIFY.EVENT (IL:|fetch| COMPLETION-EVENT IL:|of| CONTEXT))
        (LET* ((LOCK (IL:|fetch| CONTEXT-LOCK IL:|of| CONTEXT))
              (DEFAULT-CHAR-HANDLER (IL:|fetch| DEFAULT-CHAR-HANDLER IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT)))
              (COMMAND-TABLE (IL:|fetch| COMMAND-TABLE IL:|of| (IL:|fetch| ENVIRONMENT IL:|of| CONTEXT)))
              (WINDOW (IL:|fetch| DISPLAY-WINDOW IL:|of| CONTEXT))
              (PROMPTWINDOW (IL:GETPROMPTWINDOW WINDOW))
              (CHARCODE COMMAND THIS-CHAR-ESCAPED)
              (DECLARE (IL:SPECVARS THIS-CHAR-ESCAPED))
              (LOOP
                (WHEN (NULL (IL:ERSETQ
                           (IL:SETQ CHARCODE (GETKEY CONTEXT))
                           ;; AWAKE-COMMAND-PROCESS will cause getkey to return a command (a list), rather than a
                           ;; charcode
                           (IL:WITH.MONITOR LOCK
                             (WHEN CHARCODE
```

```

(IL:\CARET.DOWN WINDOW)
(SELECTION-DOWN CONTEXT)
;; this COND handles the different command generation cases. A "command" is a list of the form
;; (<fn-name> <normalize?> <extra-args>*), where <fn-name> is the function to apply, <normalize?>
;; is T if SEdit should auto-scroll after this command, and <extra-args>* are zero or more extra args to
;; the command function beyond the normal args of CONTEXT and CHARCODE.
(COND
  ((IL:LISTP CHARCODE)
    ;; a command generated externally. the variable command gets used later, so it
    ;; must be set here
    (IL:SETQ COMMAND CHARCODE)
    (IL:SETQ THIS-CHAR-ESCAPED NIL)
    (FORMAT PROMPTWINDOW "~%")
    (IL:APPLY (CAR COMMAND)
      (LIST* CONTEXT NIL (CDDR COMMAND))))
    (THIS-CHAR-ESCAPED
      ; an escaped char
      (FUNCALL DEFAULT-CHAR-HANDLER CONTEXT CHARCODE)
      (IL:SETQ THIS-CHAR-ESCAPED NIL))
    ((AND (OR (IL:SETQ COMMAND (LOOKUP-COMMAND CHARCODE
      COMMAND-TABLE))
      (IL:SETQ COMMAND (LOOKUP-COMMAND (IL:GETSYNTAX
      CHARCODE)
      COMMAND-TABLE)))
      (IL:APPLY (CAR COMMAND)
        (LIST* CONTEXT CHARCODE (CDDR COMMAND))))
      ;; this is a valid command or syntax char, and it has already been handled
    )
    (T ;; none of the above, or else the command didn't want to run. treat as normal
      ;; input
      (FUNCALL DEFAULT-CHAR-HANDLER CONTEXT CHARCODE)))
    (WHEN (OR (NOT COMMAND)
      (NOT (IL:FMEMB (CAR COMMAND)
        '(UNDO REDO))))
      (IL:|replace| UNDO-UNDO-LIST IL:|of| CONTEXT IL:|with| NIL)))
      ;; unless the user is typing too fast to keep up, fix up the window
      (UNLESS (IL:\SYSBUFP)
        (UPDATE CONTEXT NIL (SECOND COMMAND))
        ;; once the update has triggered on this command, set it to nil so other updates without a new
        ;; command (shift selection...) won't update with an old command.
        (SETQ COMMAND NIL))))))
    ;; on catching of errors, re-update to capture what was undone to run the command, like the current selection
    (UPDATE CONTEXT T))))))

```

(SELECT-NEXT-GAP

```

(IL:LAMBDA (CONTEXT NODE INDEX) ; Edited 23-Nov-87 18:23 by DCB
  (IL:SETQ NODE (NEXT-NODE NODE INDEX))
  (IL:WHILE NODE IL:DO (WHEN (EQ (IL:FETCH NODE-TYPE IL:OF NODE)
    TYPE-GAP)
    (SELECT-SEGMENT (IL:FETCH SELECTION IL:OF CONTEXT)
      CONTEXT
      (IL:FETCH SUPER-NODE IL:OF NODE)
      NODE NODE)
    (PENDING-DELETE (IL:FETCH CARET-POINT IL:OF CONTEXT)
      (IL:FETCH SELECTION IL:OF CONTEXT))
    (RETURN T))
    (IL:SETQ NODE (NEXT-NODE NODE))
  IL:FINALLY (SET-SELECTION-NOWHERE (IL:FETCH SELECTION IL:OF CONTEXT))))

```

(SET-DEPTH

```

(IL:LAMBDA (NODE DEPTH) ; Edited 17-Nov-87 11:19 by DCB

```

;; set the depth of this subtree

```

(IL:REPLACE DEPTH IL:OF NODE IL:WITH DEPTH)
(IL:FOR X IL:IN (CDR (IL:FETCH SUB-NODES IL:OF NODE)) IL:DO (SET-DEPTH X (IL:ADD1 DEPTH))))

```

(SET-FORMAT

```

(IL:LAMBDA (NODE CONTEXT FORMAT) ; Edited 6-Jul-87 20:51 by DCB

```

;; a node's AssignFormat method may assign a format type (such as Keyword or BindingList) to its immediate subnodes. to do so, it should call
 ;; set.format rather than setting the Format field directly. if the format type assigned is different from the old one, this function will note the node as
 ;; changed and run the AssignFormatTypes method for node to give it a chance to assign new format types to its children based upon its own changed
 ;; format type. For example, if a node's parent changed its format type from NIL to BindingList, the node will have to get a chance to change each of its
 ;; children from NIL to Binding; and these will have to get a chance to reset themselves.

;;

;;; if we are visiting every node anyway (for example, just after building the tree), we don't want to collect changed nodes and automatically propagate changed formats.

;;;

;;; the rough equivalent of the format type used to be determined in the parse phase and was known as the ParseMode. back then there was reparsing (which happened every time a node changed), and the reparser could reset the ParseMode. reparsing was determined to be at best unnecessary and at worst an evil, since reparsing during copy-selection can be disastrous. (think of copy-selecting (QUOTE A) from a TEdit window: you reparse from list to quote, get 'A, and do the wrong thing with the closing parenthesis.)

;;;

```
(WHEN (NOT (IL:EQUAL (IL:FETCH FORMAT IL:OF NODE)
                    FORMAT))
      (IL:REPLACE FORMAT IL:OF NODE IL:WITH FORMAT)
      (NOTE-CHANGE-FORMAT NODE CONTEXT)
      (WHEN (NOT (IL:FETCH DONT-COLLECT-CHANGES? IL:OF CONTEXT))
            (FUNCALL (IL:FETCH ASSIGN-FORMAT IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
                     NODE CONTEXT FORMAT))))
```

(SETUP-CONTEXT

(IL:LAMBDA (CONTEXT)

; Edited 5-Dec-90 14:10 by woz

;;; confirm that this context is setup. that means either setting up a new context or verifying the structure in an old one, and setting the initial selection

```
(COND
  ((NULL (IL:|fetch| CONTEXT-LOCK IL:|of| CONTEXT))
   ;; this is a new sedit. setup its profile, and then the context itself
   (SETUP-PROFILE (IL:|fetch| PROFILE IL:|of| CONTEXT)
                  CONTEXT)
   (SETUP-NEW-CONTEXT CONTEXT))
  ((AND (IL:|fetch| CHANGED-STRUCTURE? IL:|of| CONTEXT)
        (NOT (EQ (IL:|fetch| CHANGED-STRUCTURE? IL:|of| CONTEXT)
                 T))))
   ;; this is an old context getting restarted with a new structure stashed in the context by SEDIT. this means the new structure is not EQ with
   ;; our structure. verify what we've got against this new structure, and since it might be different, we have to throw away our edit history.
   (VERIFY-STRUCTURE CONTEXT NIL (IL:|fetch| CHANGED-STRUCTURE? IL:|of| CONTEXT))
   (THROW-AWAY-CHANGES CONTEXT))
  (T ;; just verify what we've already got.
   (VERIFY-STRUCTURE CONTEXT)))
(SET-INITIAL-SELECTION CONTEXT))
```

(SETUP-CONTEXT-WINDOW-DEPENDENCIES

(IL:LAMBDA (CONTEXT)

; Edited 6-Jul-87 20:51 by DCB

;;; setup the fields in the context that depend on the window being built

```
(LET ((WINDOW (IL:FETCH DISPLAY-WINDOW IL:OF CONTEXT)))
  ;; set the context's comment column info based on the window.
  (COMPUTE-COMMENT-COLUMN CONTEXT WINDOW)))
```

(SETUP-NEW-CONTEXT

(IL:LAMBDA (CONTEXT)

; Edited 5-Dec-90 11:59 by woz

;;; this is a new context: build all the necessary data structures.

```
(LET ((LOCK (IL:CREATE-MONITORLOCK (IL:CONCAT EDITOR-NAME (IL:|fetch| ICON-TITLE IL:|of| CONTEXT))))
      (IL:|replace| CONTEXT-LOCK IL:|of| CONTEXT IL:|with| LOCK)
      (IL:WITH-MONITOR LOCK
        (BUILD-WINDOW CONTEXT)
        (SETUP-CONTEXT-WINDOW-DEPENDENCIES CONTEXT)
        (BUILD-INTERNAL-STRUCTURE CONTEXT)
        (SETUP-WINDOW-CONTEXT-DEPENDENCIES CONTEXT))))
```

(SETUP-PROFILE

(IL:LAMBDA (PROFILE CONTEXT)

; Edited 16-Feb-88 11:14 by raf

;;; here we set up the specifics about the profile of the world we're editing in, based on what we're editing. this function must be called under WITH-PROFILE, so that the current bindings reflect the profile, because we update the profile by changing the binding as necessary and then re-saving the profile.

;;; Use current readtable, print-base, print-case, print-level, print-length.

;;; Set package based on name of structure editing. Maybe should be changed to reflect package of profile of file function lives in.

;;; The rest get forced to appropriate values for editing.

```
(IL:SETQ *READ-BASE* 10)
```

```
(IL:SETQ *READ-SUPPRESS* NIL)
(IL:SETQ *PACKAGE* (DEFAULT-PACKAGE (IL:FETCH ICON-TITLE IL:OF CONTEXT)
                                     (IL:FETCH EDIT-TYPE IL:OF CONTEXT)
                                     (IL:FETCH ROOT IL:OF CONTEXT)))

(IL:SETQ *PRINT-ESCAPE* T) ; shouldn't matter
(IL:SETQ *PRINT-PRETTY* NIL)
(IL:SETQ *PRINT-CIRCLE* NIL)
(IL:SETQ *PRINT-RADIX* (IL:NEQ *PRINT-BASE* 10)) ; interlisp semantics
(IL:SETQ *PRINT-GENSYM* T)
(IL:SETQ *PRINT-ARRAY* NIL) ; until we can edit
(IL:SETQ *PRINT-STRUCTURE* NIL) ; the structures.
(SAVE-PROFILE PROFILE))
```

(SETUP-WINDOW-AND-PROCESS

```
(IL:LAMBDA (CONTEXT) ; Edited 2-Dec-92 17:27 by jds
  (LET ((PROCESS (IL:THIS.PROCESS))
        (WINDOW (IL:FETCH DISPLAY-WINDOW IL:OF CONTEXT)))
    (WHEN (AND (IL:WINDOWPROP WINDOW 'IL:PROCESS)
              (IL:NEQ PROCESS (IL:WINDOWPROP WINDOW 'IL:PROCESS)))
      ;; it's okay if the same process is there already. this the case of RESET restarting the process.
      (IL:SHOULDNT "There's already a command process for this Sedit!")
      (IL:WINDOWPROP WINDOW 'IL:PROCESS PROCESS)
      (UPDATE-TITLE CONTEXT WINDOW T)
      (IL:SETTERMTABLE TERMINAL-TABLE)
      (IL:PROCESSPROP PROCESS 'IL:WINDOW WINDOW)
      (IL:PROCESSPROP PROCESS 'IL:TTYEXITFN (IL:FUNCTION TTYEXITFN))
      (IL:PROCESSPROP PROCESS 'IL:KEYACTION (LET ((IL:TABLE (IL:KEYACTIONTABLE IL:\\CURRENTKEYACTION))
                                                (IL:SETINTERRUPT (IL:CHARCODE IL:DEL)
                                                                NIL IL:TABLE)
                                                IL:TABLE))
      (IL:PROCESSPROP PROCESS 'IL:RESTARTABLE T)
      (IL:TTY.PROCESS PROCESS)
      (IL:CLEARW (IL:GETPROMPTWINDOW WINDOW))
      (IL:TTYDISPLAYSTREAM IL:PROMPTWINDOW))))
```

(SETUP-WINDOW-CONTEXT-DEPENDENCIES

```
(IL:LAMBDA (CONTEXT) ; Edited 6-Jul-87 20:51 by DCB

;;; setup the window properties that depend on the context being build and sedit structure computed

(LET* ((WINDOW (IL:FETCH DISPLAY-WINDOW IL:OF CONTEXT))
       (ROOT (IL:FETCH ROOT IL:OF CONTEXT))
       (HEIGHT (IL:IDIFFERENCE (IL:FETCH LINE-HEIGHT IL:OF (IL:FETCH LAST-LINE IL:OF ROOT))
                               (IL:FETCH YCOORD IL:OF (IL:FETCH LAST-LINE IL:OF ROOT)))))
  ;; now that we know about the structures, we can set the window's extent
  (IL:WINDOWPROP WINDOW 'IL:EXTENT (IL:CREATE IL:REGION
                                             IL:LEFT IL:_ 0
                                             IL:BOTTOM IL:_ (IL:IDIFFERENCE 1 HEIGHT)
                                             IL:WIDTH IL:_ (IL:FETCH ACTUAL-WIDTH IL:OF ROOT)
                                             IL:HEIGHT IL:_ HEIGHT))

  ;; and cache the title info for update.title
  (IL:WINDOWPROP WINDOW 'TITLE-INFO (LIST :|ChangedStructure?| NIL :|package| *PACKAGE* :|name|
                                          (IL:FETCH ICON-TITLE IL:OF CONTEXT))))
```

(SHIFT-LINEAR-FORM

```
(IL:LAMBDA (NODE RIGHT-SHIFT) ; Edited 11-Apr-88 17:04 by woz

;;; this node's linear form has just been shifted left or right. adjust its StartX value and that of any of its subnodes which are being displayed

(IL:|replace| START-X IL:|of| NODE IL:|with| (IL:IPLUS (IL:|fetch| START-X IL:|of| NODE)
                                                    RIGHT-SHIFT))
(IL:|for| X IL:|in| (IL:|fetch| LINEAR-FORM IL:|of| NODE) IL:|when| (IL:|type?| WEAK-LINK X)
 IL:|do| (SHIFT-LINEAR-FORM (IL:|fetch| DESTINATION IL:|of| X)
                            RIGHT-SHIFT))))
```

(STRINGIFY

```
(IL:LAMBDA (NODE ENVIRONMENT) ; Edited 6-Jul-87 20:51 by DCB
  (FUNCALL (IL:FETCH STRINGIFY IL:OF (IL:FETCH NODE-TYPE IL:OF NODE))
           NODE ENVIRONMENT))
```

(STRINGIFY-GAP

```
(IL:LAMBDA (NODE ENVIRONMENT) ; Edited 6-Jul-87 20:51 by DCB
  "-?-")
```

(SUBNODE-CHANGED

```
(IL:LAMBDA (NODE CONTEXT) ; Edited 17-Nov-87 11:20 by DCB
```

;;; inform a node that one of its subnodes has been replaced

```
(FUNCALL (IL:FETCH SUB-NODE-CHANGED IL:OF (IL:FETCH NODE-TYPE IL:OF (IL:FETCH SUPER-NODE IL:OF NODE)))
(IL:FETCH SUPER-NODE IL:OF NODE)
NODE CONTEXT)))
```

(SUBNODE-CHANGED-ROOT

```
(IL:LAMBDA (NODE SUBNODE CONTEXT) ; Edited 19-Jan-88 14:25 by woz
(LET ((FN (IL:FETCH ROOT-CHANGED-FN IL:OF CONTEXT))
EXTRA-ARGS)
(WHEN FN
(WHEN (AND (LISTP FN)
(NOT (MEMBER (FIRST FN)
'(LAMBDA IL:LAMBDA))))
;; check for the # case (car fn) = lambda. should be able to use functionp, but it returns t for an arbitrary list, which is a bug.
(SETQ EXTRA-ARGS (REST FN))
(SETQ FN (FIRST FN)))
(APPLY FN (CONS (IL:FETCH STRUCTURE IL:OF SUBNODE)
EXTRA-ARGS))))))
```

(TYPE-OF-INPUT

```
(IL:LAMBDA (CONTEXT) ; Edited 6-Jul-87 20:51 by DCB
; access fn for the type of input expected
(AND (IL:FETCH POINT-NODE IL:OF (IL:FETCH CARET-POINT IL:OF CONTEXT))
(IL:FETCH POINT-TYPE IL:OF (IL:FETCH CARET-POINT IL:OF CONTEXT))))))
```

(UNDO-EVENT

```
(IL:LAMBDA (EVENT CONTEXT) ; Edited 17-Nov-87 11:20 by DCB
(COND
((NULL EVENT)
;; someone got confused and left an unmatched blip on the undo list -- do nothing
NIL)
((IL:LISTP (CAR EVENT))
(START-UNDO-BLOCK)
(IL:FOR SUBEVENT IL:IN EVENT IL:DO (UNDO-EVENT SUBEVENT CONTEXT))
(END-UNDO-BLOCK))
(T (IL:APPLY (CAR EVENT)
(CONS CONTEXT (CDR EVENT))))))
```

(UNDO-REPLACE-ROOT

```
(IL:LAMBDA (CONTEXT NODE OLD-VALUE) ; Edited 6-Jul-87 20:51 by DCB
(WHEN (NOT (DEAD-NODE? OLD-VALUE))
(IL:SHOULDNT "undo is confused!"))
(REPLACE-ROOT NODE CONTEXT (SUBNODE 1 NODE)
(LIST OLD-VALUE)
NIL))
```

(UPDATE

```
(IL:LAMBDA (CONTEXT RELINEARIZE SCROLL?) ; Edited 13-Jun-88 18:57 by Snow
;; fix up the window after changes to the structure. relinearize.where.necessary will fix up the formatting, and we also have to figure out where the
;; point and selection should be displayed
(LET ((WINDOW (IL:|fetch| DISPLAY-WINDOW IL:|of| CONTEXT))
(SELECTION (IL:|fetch| SELECTION IL:|of| CONTEXT)))
(IF RELINEARIZE
(PROGN (RELINEARIZE (IL:|fetch| ROOT IL:|of| CONTEXT)
CONTEXT)
(FORMAT (GET-PROMPT-WINDOW CONTEXT)
"~%~%" )
(RELINEARIZE-WHERE-NECESSARY CONTEXT))
(CHECK-SELECTION SELECTION (IL:|fetch| CARET-POINT IL:|of| CONTEXT))
(COMPUTE-SELECTION-POSITION SELECTION CONTEXT)
(SELECTION-UP CONTEXT)
(UPDATE-TITLE CONTEXT WINDOW)
(SHOW-CARET CONTEXT T SCROLL?)))
```

(VERIFY-STRUCTURE

```
(IL:LAMBDA (CONTEXT CHARCODE STRUCTURE REDISPLAY-ALWAYS? CLEAR-LINEAR-FORMS?) ; Edited 5-Dec-90 14:10 by woz
```

;;; reparse and relinearize as necessary to make sure the sedit is current. can be called as a command, so must have context and charcode args. the
;;; STRUCTURE are can be a new structure to verify against. this happens when someone changed the structure under an existing edit and we found
;;; out about it. otherwise we just check what we've already got.

```
(LET* ((ROOT (IL:|fetch| ROOT IL:|of| CONTEXT))
(CHECK-STRUCTURE (OR STRUCTURE (IL:|fetch| STRUCTURE IL:|of| (CADR (IL:|fetch| SUB-NODES IL:|of| ROOT))))))
(IL:WITH-MONITOR (IL:|fetch| CONTEXT-LOCK IL:|of| CONTEXT)
(CLOSE-OPEN-NODE CONTEXT)
(IL:|replace| CURRENT-NODE IL:|of| CONTEXT IL:|with| ROOT)
(IL:|replace| \X IL:|of| CONTEXT IL:|with| (IL:|fetch| SUB-NODES IL:|of| ROOT)))
```

```

(IL:|replace| SUB-NODES IL:|of| ROOT IL:|with| (LIST 0))
(PARSE CHECK-STRUCTURE CONTEXT)
(COMPUTE-ALL-FORMATS CONTEXT)
(WHEN CLEAR-LINEAR-FORMS? (CLEAR-ALL-LINEAR-FORMS CONTEXT))
(COND
  ((OR CLEAR-LINEAR-FORMS? (IL:|fetch| CHANGED? IL:|of| ROOT))
    (SELECTION-DOWN CONTEXT)
    (RELINERIZE ROOT CONTEXT)
    (SET-SELECTION-NOWHERE (IL:|fetch| SELECTION IL:|of| CONTEXT))
    (SET-POINT-NOWHERE (IL:|fetch| CARET-POINT IL:|of| CONTEXT))
    (UPDATE CONTEXT))
  (REDISPLAY-ALWAYS?
    ;; this used to be here as a way to see the edit-date change. now nobody calls us with this flag set, so it could be
    ;; removed. wasn't cause of change control.
    (IL:REDISPLAYW (IL:|fetch| DISPLAY-WINDOW IL:|of| CONTEXT))))
(RPLACD (IL:|fetch| CHANGED-NODES IL:|of| CONTEXT)
  NIL)))))

```

(WALK-UP-TREE

```

(IL:LAMBDA (NODE CONTEXT FN) ; Edited 25-Aug-87 09:50 by drc:
  (DOLIST (SUBNODE (CDR (IL:FETCH SUB-NODES IL:OF NODE)))
    (WALK-UP-TREE SUBNODE CONTEXT FN))
  (FUNCALL FN NODE CONTEXT))

```

)

```

(IL:PUTPROPS IL:SEEDIT-BASE IL:COPYRIGHT ("Venue & Xerox Corporation" 1987 1988 1990 1991 1992))

```

FUNCTION INDEX

| | | | |
|--|----|---|----|
| ADJUST-WIDTH | 3 | LINEARIZE-ROOT | 15 |
| ASSIGN-FORMAT-NIL | 3 | MAKE-FUNCTION-PROTOTYPE | 2 |
| ATOM-CHANGE-RELINEARIZE | 3 | NEXT-NODE | 15 |
| BUILD-INTERNAL-STRUCTURE | 3 | NOTE-CHANGE | 15 |
| BUILD-LINEAR-FORM | 4 | NOTE-CHANGE-FORMAT | 15 |
| BUILD-NODE | 4 | NOTE-CHANGE-IN-SIMPLE | 16 |
| BUILD-PRELINEARIZED-NODE | 5 | PARSE | 16 |
| CLOSE-NODE | 5 | PARSE--GAP | 16 |
| COLLECT-UNDO-BLOCK | 5 | PARSE--UNKNOWN | 17 |
| COMPILE-STRUCTURE | 6 | PARSE-NEW | 17 |
| COMPLETE | 1 | PREV-NODE | 2 |
| COMPUTE-ALL-FORMATS | 6 | PROPAGATE-WIDTH-CHANGE | 17 |
| COMPUTE-FORMATS-AND-FORMAT-VALUES | 6 | RECOMPUTE-WIDTH | 17 |
| COMPUTE-POINT-POSITION | 6 | RELINEARIZE-WHERE-NECESSARY | 17 |
| COMPUTE-SELECTION-POSITION | 6 | REPLACE-NODE | 18 |
| COMPUTE-SELECTION-POSITION-DEFAULT | 7 | REPLACE-ROOT | 19 |
| CONTAINS? | 7 | REVIVE-NODE | 19 |
| COPY-NODE | 7 | SEdit1 | 19 |
| COPY-SELECTION | 8 | SELECT-NEXT-GAP | 20 |
| COPY-SELECTION-DEFAULT | 8 | SET-DEPTH | 20 |
| CREATE-CONSTANT-STRINGS | 9 | SET-FORMAT | 20 |
| CREATE-ENVIRONMENTS | 9 | SET-INITIAL-SELECTION | 2 |
| CREATE-GAP-NODE | 10 | SETUP-CONTEXT | 21 |
| CREATE-NODE | 10 | SETUP-CONTEXT-WINDOW-DEPENDENCIES | 21 |
| CREATE-PRELINEARIZED-NODE | 10 | SETUP-NEW-CONTEXT | 21 |
| CREATE-PRETTY-PRINT-ENV | 11 | SETUP-PROFILE | 21 |
| CREATE-SIMPLE-NODE | 11 | SETUP-WINDOW-AND-PROCESS | 22 |
| CREATE-STRING-ITEM | 11 | SETUP-WINDOW-CONTEXT-DEPENDENCIES | 22 |
| DEFAULT-COMPILE-FN | 11 | SHIFT-LINEAR-FORM | 22 |
| DEFAULT-GETDEF-FN | 11 | STRINGIFY | 22 |
| DEFAULT-PACKAGE | 12 | STRINGIFY-GAP | 22 |
| DELETE-NODES | 12 | SUBNODE-CHANGED | 22 |
| DETACH-NODE | 12 | SUBNODE-CHANGED-ROOT | 23 |
| FORMAT-VALUES-CHANGED | 12 | THROW-AWAY-CHANGES | 2 |
| GET-SELECTED-STRUCTURE | 12 | TYPE-OF-INPUT | 23 |
| HANDLE-COMPLETION | 12 | UNDO-EVENT | 23 |
| INITIALIZE | 13 | UNDO-REPLACE-ROOT | 23 |
| INSERT | 14 | UPDATE | 23 |
| INSERT-CHANGED | 15 | VERIFY-STRUCTURE | 23 |
| KILL-NODE | 15 | WALK-UP-TREE | 24 |

VARIABLE INDEX

| | | | | | |
|------------------------------------|---|--------------------------------------|---|-------------------|---|
| *CLEAR-LINEAR-ON-COMPLETION* | 1 | *IGNORE-CHANGES-ON-COMPLETION* | 1 | BODY-BITMAP | 1 |
| *COMPILE-FN* | 1 | ARGS-BITMAP | 1 | GAP-BITMAP | 1 |

PROPERTY INDEX

| | |
|---------------------|---|
| IL:SEdit-BASE | 1 |
|---------------------|---|
