

File created: 17-May-90 11:03:43 {DSK}<usr>local>lde>lispcore>sources>SEDIT-INDENT.;2

changes to: (IL:VARS IL:SEDIT-INDENTCOMS)

previous date: 27-Jun-88 17:37:44 {DSK}<usr>local>lde>lispcore>sources>SEDIT-INDENT.;1

Read Table: XCL

Package: SEDIT

Format: XCCS

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

```
(IL:RPAQQ IL:SEDIT-INDENTCOMS
  ((IL:PROP IL:FILETYPE IL:SEDIT-INDENT)
   (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:SEDIT-INDENT)
   (IL:DECLARE\ IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FILES IL:SEDIT-DECLS))
   (IL:VARIABLES LIST-FORMATS-TABLE *FSPEC-TABLE* *FSPEC-TABLE-COPY* *INDENT-ALIST*)
   (IL:FUNCTIONS GET-INDENT GET-FORMAT)
   (IL:SETFS GET-INDENT GET-FORMAT)
   (IL:FUNCTIONS RESET-FORMATS INSTALL-SPECIAL-FORMATS PARSE-FORMAT SETF-OF-GET-FORMAT
    FORMAT-FROM-INDENT PARSE-INDENT PARSE-INDENT-NAME PARSE-INDENT-BODY PARSE-INDENT-GROUP
    PARSE-INDENT-GROUP-BODY PARSE-INDENT-GROUP-ONE PARSE-INDENT-GROUPS SCALE-INDENT)
   (IL:FUNCTIONS COPY-HASH-TABLE)
   (IL:COMS
    ;; a definer for formats
    (IL:DEFINE-TYPES IL:SEDIT-FORMATS)
    (IL:FUNCTIONS DEF-LIST-FORMAT)
    (IL:PROP IL:ARGNAMES DEF-LIST-FORMAT)))

(IL:PUTPROPS IL:SEDIT-INDENT IL:FILETYPE :COMPILE-FILE)

(IL:PUTPROPS IL:SEDIT-INDENT 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)
)

(DEFVAR LIST-FORMATS-TABLE (MAKE-HASH-TABLE :SIZE 1000))

(DEFVAR *FSPEC-TABLE* (MAKE-HASH-TABLE :TEST #'EQUAL :SIZE 2000)
 "Associates function names with their format specifications.")

(DEFVAR *FSPEC-TABLE-COPY* NIL
 "hash table containing original format specs")

(DEFPARAMETER *INDENT-ALIST*
 (LIST
```

;;; Each entry associates a name with a list of two indentation specifications. The first is for preferred mode and the second is for miser mode. Each
;;; number in the specs is taken as an indentation level (0=none, 1=body, 2=step1, etc.) and will be scaled appropriately at installation time.

```
(LIST :VERTICAL
  ;; vertical indentation aligns all args with first, each on their own line. In preferred mode, the first arg goes on the same line with the
  ;; CAR. In miser mode, it goes on the next line at body indentation. If the CAR is non-atomic then the first arg always goes on next
  ;; line with NO indentation.
  (LIST (LIST* 'BREAK 'FROM-INDENT 0)
        (LIST* 'SET-INDENT 'PREV-ATOM? 1 'BREAK 0)
        (LIST* 'BREAK 'FROM-INDENT 0))
  (LIST (LIST* 'BREAK 'FROM-INDENT 0)
        (LIST* 'SET-INDENT 'BREAK 'PREV-ATOM? 1 0)
        (LIST* 'BREAK 'FROM-INDENT 0)))
(LIST :KEYWORD-ARG
  ;; Keyword-arg indentation is like vertical, but args which follow keywords go on the same line as the keyword. Note this won't work
  ;; real well if there are keyword values being specified for regular args.
  (LIST (LIST* 'PREV-KEYWORD? (LIST* 'NEXT-INLINE? 1 'BREAK 'FROM-INDENT 1)
        'BREAK
        'FROM-INDENT 0)
        (LIST* 'SET-INDENT 'PREV-ATOM? 1 'BREAK 0)
        (LIST* 'PREV-KEYWORD? (LIST* 'NEXT-INLINE? 1 'BREAK 'FROM-INDENT 1)
        'BREAK
        'FROM-INDENT 0))
  (LIST (LIST* 'PREV-KEYWORD? (LIST* 'NEXT-INLINE? 1 'BREAK 'FROM-INDENT 1)
        'BREAK
        'FROM-INDENT 0)
        (LIST* 'SET-INDENT 'BREAK 'PREV-ATOM? 1 0)
        (LIST* 'PREV-KEYWORD? (LIST* 'NEXT-INLINE? 1 'BREAK 'FROM-INDENT 1)
        'BREAK
```

```

      'FROM-INDENT 0)))
(LIST :HORIZONTAL
  ;; Horizontal packs as many args on a single line as will fit. Note that only complete forms are packed together on a line, not pieces
  ;; of forms. Also, the notes under :vertical about miser/preferred modes and non-atomic CARs apply here as well.
  (LIST (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)
    (LIST* 'SET-INDENT 'PREV-ATOM? 1 'BREAK 0)
    (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0))
  (LIST (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)
    (LIST* 'SET-INDENT 'BREAK 'PREV-ATOM? 1 0)
    (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)))
(LIST :HORIZONTAL-ATOM
  ;; break before & after keyword/arg pairs & lists, otherwise atoms stay on one line
  (LIST (LIST* 'FROM-INDENT 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)
    (LIST* 'SET-INDENT 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)
    (LIST* 'FROM-INDENT 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0))
  (LIST (LIST* 'FROM-INDENT 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)
    (LIST* 'SET-INDENT 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0)
    (LIST* 'FROM-INDENT 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
    'BREAK 0))
  (LIST :SQUASH
    ;; Squash is like horizontal except it will also pack complete forms on the ends of lines which finish off partial forms. As in
    ;; (FOO/BAR) BAZ/BAM where each / indicates a line break.
    (LIST (LIST* 'FROM-INDENT 'NEXT-INLINE? 0 'BREAK 0)
      (LIST* 'SET-INDENT 'PREV-ATOM? 1 'BREAK 0)
      (LIST* 'FROM-INDENT 'NEXT-INLINE? 0 'BREAK 0))
    (LIST (LIST* 'FROM-INDENT 'NEXT-INLINE? 0 'BREAK 0)
      (LIST* 'SET-INDENT 'BREAK 'PREV-ATOM? 1 0)
      (LIST* 'FROM-INDENT 'NEXT-INLINE? 0 'BREAK 0)))
  (LIST :DATA
    ;; Data packs as many atoms on a line as will go, possibly followed by a single list. There isn't any difference between regular and
    ;; miser modes.
    (LIST (LIST* 'PREV-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
      'BREAK 0))
    (LIST (LIST* 'PREV-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
      'BREAK 0)))
  (LIST :BINDING
    ;; This is an "extended binding", as in a lambda list or the binding list of a DO. The CAR is the variable bound, the CADR is the form
    ;; to bind to, and the (optional) CADDR is a sometimes-evaluated form. We line up the second and third forms, basically like vertical
    ;; mode.
    (LIST (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
      'BREAK 0)
      (LIST* 'SET-INDENT 1)
      (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
      'BREAK 0))
    (LIST (LIST* 'FROM-INDENT 'BREAK 0)
      (LIST* 'SET-INDENT 'BREAK 1)
      (LIST* 'FROM-INDENT 'BREAK 0)))
  (LIST :BINDING-LIST
    ;; This is a list of bindings, as in LETs and DOs. They all line up vertically, and each goes on its own line except strings of atoms are
    ;; grouped on one line.
    (LIST (LIST* 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
      'BREAK 0)
      'BREAK 0))
    (LIST (LIST* 'PREV-ATOM? (LIST* 'NEXT-ATOM? (LIST* 'NEXT-INLINE? 0 'BREAK 0)
      'BREAK 0)
      'BREAK 0))
  (LIST :COND-CLAUSE
    ;; Cond clauses line up each form on its own line unless all can go on one.
    (LIST (LIST* 'BREAK 0))
    (LIST (LIST* 'BREAK 0)))
  (LIST :LAMBDA-LIST
    ;; Lambda lists go all on one line if possible. Otherwise they group strings of atoms on one line and put each initialized binding on a
    ;; line by itself. Lambda-list words like &optional get extended to the list margin and start a new level of indentation for the following
    ;; forms.

```



```

LIST-INLINE? IL:_ 'ASSIGN-FORMAT-CLISP
LIST-PFORMAT IL:_ 'CFV-CLISP
LIST-MFORMAT IL:_ 'LINEARIZE-CLISP))
(SET-LIST-FORMAT :DOTLIST (IL:CREATE LIST-FORMAT
LIST-FORMATS IL:_ NIL
LIST-INLINE? IL:_ 'ASSIGN-FORMAT-DOTLIST
LIST-PFORMAT IL:_ 'CFV-DOTLIST
LIST-MFORMAT IL:_ 'LINEARIZE-DOTLIST)))

```

```
(DEFUN PARSE-FORMAT (FORMAT-SPEC)
```

```
;; A format specification is a plist. We parse it and return an SEdit internal list format object for it.
```

```
;; REUSE is a list-format object to be re-used
```

```

(DESTRUCTURING-BIND (&KEY INDENT ARGS SUBLISTS INLINE MISER LAST)
  FORMAT-SPEC
  (FORMAT-FROM-INDENT (PARSE-INDENT INDENT)
    ARGS SUBLISTS INLINE MISER (GETF FORMAT-SPEC :LAST :REPEAT))))

```

```
(DEFUN SETF-OF-GET-FORMAT (NAME SPEC)
```

```
;; Replace the external format spec associated with NAME, or add a new one if necessary. Side effect is to associate the parsed version of the spec
;; with NAME internally. This way, external and internal versions always stay in sync.
```

```
;; SPEC is either a plist or the name of a defined format.
```

```

(WHEN *FSPEC-TABLE-COPY*
  ;; don't parse when we're bootstrapping
  (LET ((FORMAT (ETYPESPEC SPEC
    (SYMBOL SPEC) ; it's an alias
    (CONS (PARSE-FORMAT SPEC)) ; it's a real format
  )))
    ;; store the internal definition
    (SET-LIST-FORMAT NAME FORMAT)))
  ;; finally store external definition
  (IF SPEC
    (SETF (GETHASH NAME *FSPEC-TABLE*)
      SPEC)
    (REMHASH NAME *FSPEC-TABLE*)))

```

```
(DEFUN FORMAT-FROM-INDENT (INDENTS ARGS SUBLISTS INLINE? &OPTIONAL MISER LAST)
```

```
;; We are passed the SEdit-internal preferred and miser indents (in a list), a list of the SEdit formats for the arguments (if any), and the setting of the
;; ListInline? field, and we return a SEdit format structure that carries this information. The optional args are used to determine (1) which of the indents
;; to put into the SEdit format, and (2) how to process the subforms list before stuffing it into the format.
```

```

(CASE LAST
  (:REPEAT
    ;; The last form should get the repeat formatting, so we take the last element of the subforms and push it on the front. (Note this works
    ;; even if the arg info is NIL.)
    (PUSH (CAR (LAST ARGS))
      ARGS))
  (OTHERWISE
    ;; The user wants to format the last arg specially, so we put this format at the front of the SEdit arg list.
    (PUSH LAST ARGS)))

```

```
;; The miser arg flags if we always or never use the miser format. The default is to use whichever makes things fit best (as SEdit figures it).
```

```

(CASE MISER
  (:ALWAYS (SETF INDENTS (LIST (SECOND INDENTS)
    (SECOND INDENTS))))
  (:NEVER (SETF INDENTS (LIST (FIRST INDENTS)
    (FIRST INDENTS))))
  (IL:CREATE LIST-FORMAT
    LIST-FORMATS IL:_ ARGS
    LIST-SUBLISTS IL:_ SUBLISTS
    LIST-INLINE? IL:_ INLINE?
    LIST-PFORMAT IL:_ (SCALE-INDENT (FIRST INDENTS))
    LIST-MFORMAT IL:_ (SCALE-INDENT (SECOND INDENTS))))

```

```
(DEFUN PARSE-INDENT (ISPEC &AUX INDENTS)
```

```
;; An indent specification is either a keyword indent name or a list of groupings. An indent (which we return) is a list of two SEdit-internal indentation
;; specs: the first for preferred mode and the second for miser mode. See *INDENT-ALIST* for details.
```

```

(COND
  ((KEYWORDP ISPEC)
    (SETF INDENTS (PARSE-INDENT-NAME ISPEC)))
  ((LISTP ISPEC)

```

```

    (SETF INDENTS (PARSE-INDENT-GROUPS ISPEC)))
    (T (ERROR "Use :vertical indentation." "Illegal indent specification: ~S" ISPEC)
      (SETF INDENTS (PARSE-INDENT-NAME :VERTICAL))))
    (UNLESS (AND INDENTS (LISTP INDENTS))
      (ERROR "Unanticipated parse error in parse-indent!"))
    INDENTS)

```

```
(DEFUN PARSE-INDENT-NAME (NAME)
```

;; The only special indent names are those in the *indent-alist*. So we do an error-checked lookup.

```

  (LET ((INDENTS (GET-INDENT NAME)))
    (UNLESS INDENTS
      (CERROR "Use :vertical indentation." "Not a known indentation: ~S." NAME)
      (SETF INDENTS (CDR (ASSOC :VERTICAL *INDENT-ALIST*))))
    INDENTS))

```

```
(DEFUN PARSE-INDENT-BODY (INDENTS OFFSET TAGBODY?)
```

;; Creates the body part of an indentation spec. If this is a tagbody, we exdent atoms. Note that, since the first element sets the indent, the indent will be set exdented if the first element is a tag. This will screw up double-semi comments. To compensate as best we can, we make every non-exdented form in a tagbody set the tab. That way, the only double-semi comments that get screwed up are ones following an initial tag but preceding all the forms.

;; As usual, we precede everything with WholeInline? tests that disable the breaks, that way you can win if you specify :inline as a format option.

```

  (LET* ((FIRST (IF TAGBODY?
    (LIST* 'BREAK 'NEXT-ATOM? 0 'SET-INDENT OFFSET)
    (LIST* 'SET-INDENT 'BREAK OFFSET)))
    (REPEAT (IF TAGBODY?
    (LIST* 'BREAK 'NEXT-ATOM? 0 'SET-INDENT OFFSET)
    (LIST* 'BREAK OFFSET))))

```

;; Start off with the first and repeat forms.

```

  (PUSH FIRST (FIRST INDENTS))
  (PUSH FIRST (SECOND INDENTS))
  (PUSH REPEAT (FIRST INDENTS))
  (PUSH REPEAT (SECOND INDENTS))

```

;; Now, since indents have this screwy last-element first format, we reverse the whole thing and then add the repeat to the beginning.

```

  (LIST (CONS REPEAT (NREVERSE (FIRST INDENTS)))
    (CONS REPEAT (NREVERSE (SECOND INDENTS)))))

```

```
(DEFUN PARSE-INDENT-GROUP (INDENTS GROUP OFFSET &AUX (BREAK (LIST* 'FROM-INDENT 'BREAK 0))
  (NOBREAK (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0
    'BREAK 0)))
  'BREAK 0)))

```

;; Each group after the first (if it has any members at all) starts on a new line. And we should only be called for groups after the first. Elements after the first are handled as a normal group body.

```

  (UNLESS (AND (NUMBERP GROUP)
    (= GROUP 0))
    (PUSH (LIST* 'SET-INDENT 'BREAK OFFSET)
      (FIRST INDENTS))
    (PUSH (LIST* 'SET-INDENT 'BREAK OFFSET)
      (SECOND INDENTS))
    (PARSE-INDENT-GROUP-BODY INDENTS GROUP)))

```

```
(DEFUN PARSE-INDENT-GROUP-BODY (INDENTS GROUP &AUX (BREAK (LIST* 'FROM-INDENT 'BREAK 0))
  (NOBREAK (LIST* 'FROM-INDENT 'PREV-INLINE? (LIST* 'NEXT-INLINE? 0
    'BREAK 0)))
  'BREAK 0)))

```

;; Creates the body part of one of the distinguished groups in an indentation spec. If the spec is a simple number, we force each form in the body onto a separate line starting at the tab stop. If the spec is a number inside a list, we allow the body forms to go together on lines if they fit in line. The idea is really to allow them either to ALL go on one line or else ALL go on separate lines, but the SEdit indentation mechanism doesn't have enough power to allow this.

;; As usual, we precede everything with WholeInline? tests that disable the breaks, that way you can win if you specify :inline as a format option.

```

  (COND
    ((NUMBERP GROUP)
      (DOTIMES (I (1- GROUP))
        (PUSH BREAK (FIRST INDENTS))
        (PUSH BREAK (SECOND INDENTS))))
    ((AND (LISTP GROUP)
      (NUMBERP (FIRST GROUP)))
      (DOTIMES (I (1- (FIRST GROUP)))
        (PUSH NOBREAK (FIRST INDENTS))
        (PUSH NOBREAK (SECOND INDENTS))))
    (T (ERROR "Illegal indent group specification: ~S" GROUP)))

```

```
(DEFUN PARSE-INDENT-GROUP-ONE (INDENTS GROUP OFFSET ARG-1)
```

;;; The first distinguished group in an indent spec has to specially place the first arg if desired. We do that and then call the normal code to place any other args in the first group.

```
(UNLESS (AND (NUMBERP GROUP)
              (= GROUP 0))
  (PUSH (CASE ARG-1
          (:BREAK (LIST* 'SET-INDENT 'BREAK OFFSET))
          (OTHERWISE (LIST* 'SET-INDENT OFFSET)))
        (FIRST INDENTS))
  (PUSH (CASE ARG-1
          (:NOBREAK (LIST* 'SET-INDENT OFFSET))
          (:BREAK (LIST* 'SET-INDENT 'BREAK OFFSET))
          (OTHERWISE (LIST* 'SET-INDENT 'NEXT-PREFERRED? OFFSET 'BREAK OFFSET)))
        (SECOND INDENTS))
  (PARSE-INDENT-GROUP-BODY INDENTS GROUP)))
```

```
(DEFUN PARSE-INDENT-GROUPS (GROUPS &AUX (ARG-1 NIL)
                              (TAGBODY? NIL)
                              (CURIN 1)
                              (INDENTS (LIST NIL NIL)))
```

;;; A grouping is either a number or a list containing a single number. Each number indicates how many forms are to be indented at the current level. Each group is indented 1 step further in from the next group, except the first group is sometimes indented as a first arg. A parenthesized group number indicates that the group members can sit on one line with each other, else each form goes on its own line. Each group sets the tab.

```
(COND
  (GROUPS
    ;; the spec can be preceded by keywords: :step (increase all indentations one step), :tagbody (the body part is a tagbody),
    ;; :break/:nobreak/:fit (describes where to place the first group). These can come in any order.
    (DO ((G GROUPS (REST G)))
        ((OR (NULL G)
              (NOT (KEYWORDP (FIRST G)))))
      (SETF GROUPS G))
    (CASE (FIRST G)
      (:STEP (INCF CURIN))
      (:TAGBODY (SETF TAGBODY? T))
      ((:BREAK :NOBREAK :FIT)
       (WHEN ARG-1
         (CERROR "Ignore it." "Extra placement keyword in indentation: ~A" (FIRST G)))
       (SETF ARG-1 (FIRST G)))
      (OTHERWISE (CERROR "Ignore it." "Unrecognized indentation keyword: ~A." (FIRST G))))
    (INCF CURIN (LENGTH GROUPS))
    (WHEN GROUPS
      (PARSE-INDENT-GROUP-ONE INDENTS (FIRST GROUPS)
                              CURIN ARG-1)
      (DECF CURIN)
      (DOLIST (IL:GROUP (REST GROUPS))
        (PARSE-INDENT-GROUP INDENTS IL:GROUP CURIN)
        (DECF CURIN)))
      (PARSE-INDENT-BODY INDENTS CURIN TAGBODY?))
    (T (CERROR "Use :vertical indentation." "Null indentation specification.")
      (PARSE-INDENT-NAME :VERTICAL))))
```

```
(DEFUN SCALE-INDENT (INDENT &OPTIONAL (INDENT-BASE (IL:FETCH INDENT-BASE IL:OF LISP-EDIT-ENVIRONMENT))
                              (INDENT-STEP (IL:FETCH INDENT-STEP IL:OF LISP-EDIT-ENVIRONMENT)))
```

;;; Substitute point sizes for the indentation tab stop specifications in an indent. This definition is adapted from that for SUBST given in the Common Lisp manual. The result shares as much structure with the original as possible.

```
(COND
  ((NUMBERP INDENT)
   (IF (= INDENT 0)
       INDENT
       (+ INDENT-BASE (* INDENT-STEP (1- INDENT)))))
  ((CONSP INDENT)
   (LET ((IL:LEFT (SCALE-INDENT (CAR INDENT)
                                   INDENT-BASE INDENT-STEP))
         (IL:RIGHT (SCALE-INDENT (CDR INDENT)
                                   INDENT-BASE INDENT-STEP)))
     (IF (AND (EQL IL:LEFT (CAR INDENT))
              (EQL IL:RIGHT (CDR INDENT)))
         INDENT
         (CONS IL:LEFT IL:RIGHT))))
  (T INDENT)))
```

```
(DEFUN COPY-HASH-TABLE (OLD-TABLE NEW-TABLE)
```

;; copies the contents of OLD-TABLE into NEW-TABLE

```
(MAPHASH #'(LAMBDA (KEY VALUE)
              (SETF (GETHASH KEY NEW-TABLE)
```

```
        VALUE))
      OLD-TABLE)
    NEW-TABLE)
```

;; a definer for formats

```
(DEF-DEFINE-TYPE IL:SEdit-FORMATS "SEdit list formats")

(DEFDEFINER (DEF-LIST-FORMAT (:UNDEFINER (LAMBDA (NAME)
                                           (SETF (GET-FORMAT NAME)
                                                 'NIL))))
  IL:SEdit-FORMATS (NAME &REST REST)
  (LET* ((DOCUMENTATION (IF (STRINGP (CAR REST))
                             (CAR REST)))
        (REST (IF DOCUMENTATION
                    (CDR REST)
                    REST)))
    `(SETF (GET-FORMAT ',NAME)
      ',(IF (ENDP (CDR REST))
            (CAR REST)
            REST)
      ,@(IF DOCUMENTATION
            `((DOCUMENTATION ,NAME 'IL:SEdit-FORMATS)
              ,DOCUMENTATION))))))

(IL:PUTPROPS DEF-LIST-FORMAT IL:ARGNAMES (NAME {DOC} &KEY INDENT ARGS INLINE MISER LAST SUBLISTS))

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

FUNCTION INDEX

| | | | | | | | |
|--------------------------|---|---------------------------|---|---------------------------|---|--------------------------|---|
| COPY-HASH-TABLE | 6 | INSTALL-SPECIAL-FORMATS . | 3 | PARSE-INDENT-GROUP | 5 | PARSE-INDENT-NAME | 5 |
| FORMAT-FROM-INDENT | 4 | PARSE-FORMAT | 4 | PARSE-INDENT-GROUP-BODY . | 5 | RESET-FORMATS | 3 |
| GET-FORMAT | 3 | PARSE-INDENT | 4 | PARSE-INDENT-GROUP-ONE . | 6 | SCALE-INDENT | 6 |
| GET-INDENT | 3 | PARSE-INDENT-BODY | 5 | PARSE-INDENT-GROUPS | 6 | SETF-OF-GET-FORMAT | 4 |

VARIABLE INDEX

| | | | | | | | |
|---------------------|---|--------------------------|---|----------------------|---|--------------------------|---|
| *FSPEC-TABLE* | 1 | *FSPEC-TABLE-COPY* | 1 | *INDENT-ALIST* | 1 | LIST-FORMATS-TABLE | 1 |
|---------------------|---|--------------------------|---|----------------------|---|--------------------------|---|

PROPERTY INDEX

| | | | |
|-----------------------|---|-----------------------|---|
| DEF-LIST-FORMAT | 7 | IL:SEdit-INDENT | 1 |
|-----------------------|---|-----------------------|---|

SETF INDEX

| | | | |
|------------------|---|------------------|---|
| GET-FORMAT | 3 | GET-INDENT | 3 |
|------------------|---|------------------|---|

DEFINER INDEX

| | |
|-----------------------|---|
| DEF-LIST-FORMAT | 7 |
|-----------------------|---|

DEFINE-TYPE INDEX

| | |
|------------------------|---|
| IL:SEdit-FORMATS | 7 |
|------------------------|---|