## List Formats

SEdit allows one to specify how a class of forms are to be pretty printed. This is done by defining a list format on a symbol. This causes all forms whose car is this symbol to be displayed with the specified format.

List formats for most Common LISP and Interlisp special forms are provided with SEdit. The source code for these definitions can be found on the Lisp Library Floppy #XXX.

`(def-list-format` *name* {*doc-string*} {*format-name* | `&key` *:args :sublists :inline :miser :last :indent*}) [Definer]

Tells SEdit how to prettyprint forms whose CAR is NAME.

**Short form**

If *FORMAT-NAME* is provided then NAME is defined to be formatted just like *FORMAT-NAME*. If, for example one had defined a list format for `dotimes` one could then define `dolist` to be formatted the same way with:

```
(def-list-format dotimes dolist)
```

**Long Form**

The keyword arguments have the following meanings:

`:ARGS` -- value should be a list of names of list formats. These formats are assigned to the elements of the list in order starting with the first element (which will be `NAME`). Note that these formats override any formats that would normally be assigned to the elements of the list (based on their first elements). NIL is allowed in the `:ARGS` list, and means do *not* override the format of this element; that is, allow it to be formatted normally. Also, a symbol S is allowed in the :ARGS list if S has earlier been assigned a format; this means to assign S's format to this element. There are also two special keywords allowed as entries in the :ARGS list: :KEYWORD and :RECURSIVE. :KEYWORD means that if the element assigned this format is a symbol then treat it like a keyword, i.e., put it in bold face. (This list uses the convention that all symbols which allow declarations in their body [such as DO and LET] are formatted as keywords.) :RECURSIVE means to assign this element the same format as is being defined; that is, the entire top level format is assigned recursively to this element. This is very useful for formats like :DATA format (see below). If L has more elements than there are entries in the :ARGS list, the last entry in the :ARGS list is repeated for all the extra elements of L. Hint: most :ARGS entries have NIL as their last element. If no :ARGS list is specified, the elements of L get their natural formatting.

**:SUBLISTS** -- value should be a list of element positions (counting from 1) or T. T means all of the arguments should be parsed as lists even if they are NIL (so NIL will display as () rather than NIL). A list of element position means those element positions will be parsed as lists. For example LET has :SUBLISTS (2) meaning the second element of a form whose first element is LET is a list (i.e., the binding list). DO has :SUBLISTS (2 3), DEFUN has :SUBLISTS (3) and COND has :SUBLISTS T. Default is :SUBLISTS NIL meaning print all NIL args as NIL not ().

**:INLINE** -- value can be T or NIL (default NIL). If T, the form will go all on one line if it fits. If NIL, the form will be broken across lines at arg boundaries even if it would all fit on one line. For example, OR has :INLINE T and LET has :INLINE NIL.

**:MISER** -- value can be :ALWAYS, :NEVER, or :TOFIT (default :TOFIT). Specifies when to use miser indentation. The default means use miser indentation if non-miser indentation would force the arguments into miser indentation. **[need to explain what miser mode is]**

**:LAST** -- value should be a format specification like those in the :ARGS list. This format specification will be applied to the last element of L but *only* if doing so would supercede the last entry in the :ARGS list. In other words, if the last element of L would receive the repeated format from the :ARGS list, it gets the :LAST format instead. This option is really only useful for pathologically formatted forms like Interlisp's SELECTQ.

:indent -- An indentation specification is either a symbol (normally a keyword) or a list. If it's a symbol, it's looked up on the SEDIT::*INDENT-ALIST* (which see) and the SEdit-internal indent specification found

there is used. If it's a list, it consists of some optional keywords (described below) followed by argument group specifications. Each argument group specification is either a number or a list containing a single number. In both formats, the number indicates that that many arguments should be grouped together at a single indentation level. The simple number format means that each of those arguments should go on its own line (they will line up vertically with each other), while the number-in-a-list format means that the arguments in the group can go together on a single line if they fit. The indentation level for each argument group is determined by how many groups follow it in the indentation list. Each group is indented 1 level further in than the group which follows it; thus, the first argument group is indented most, the next one next most, and so on until the last one, which is always indented one step in from lambda-body level.

This is best explained with examples. A simple example is LET, whose indentation specification is (1). This means that LET will be followed by a single distinguished argument group consisting of one element (the binding list) which will be indented one step in from the let body. Another simple example is DO, whose indentation specification is (2). This means that DO will be followed by a single distinguished argument group consisting of two elements (the binding list and the termination clause) which will be indented one step in from the do body. It also means that the bindings and the termination will be required to go on separate lines. Contrast DO with DEFUN, whose indentation specification is ((2)). Like DO's spec, DEFUN's spec says there is one group with two members (the name and the lambda-list), but unlike DO's spec, DEFUN's spec says that the first two args can go on the same line if they fit there. Finally, consider a possible spec for MULTIPLE-VALUE-BIND of (1 1) which says that the first group consists of one arg (the variable list) and the second group consists of one arg (the form to eval). The form to eval will be indented one step in from the body, and the list of variables will be indented one step in from there.

Note that a group specification of 0 (zero) is allowed: this occupies an indentation step but does not put any arguments at that level. But we do not allow (0) as a group specification since this would not be any different than plain 0 and probably means that the specification is confused in some way.

The keywords allowed at the beginning of an indent specification are:

**:BREAK** or **:NOBREAK** or **:FIT** -- These specify placement of the first argument in the first group. Default is :FIT, which means put this arg on the same line as the CAR of the form if it fits there in preferred mode, otherwise put it on the next line. Note that if the first arg goes on the same line as the CAR, its placement specifies the indentation level for the entire first group. That way long CARs will move the first group over to the right. (This makes the binding and termination of both DO and DO* line up, for example.) Specifying :NOBREAK means the first arg is forced to go on the same line as the CAR. Specifying :BREAK means the first arg is forced to go on the next line (and thus at the indentation level derived from the number of groups). UNWIND-PROTECT is a good example of using :BREAK to force the first arg onto its own line. Note that you can only specify one of :NOBREAK, :BREAK, or :FIT.

**:TAGBODY** -- Normally all forms in the body (whether atomic or not) go at the same indent level. Specifying :TAGBODY indicates that atomic body elements (*not* atomic elements of the argument groups) should be exdented to line up with the CAR of the entire list (such as PROG or TAGBODY, which see for examples).

**:STEP** -- This can be specified as many times as desired and each time increases the indentation of the body (and thus all the argument groups) by one step. If you just want to move some of the groups in but not all of them (and not the body) then use 0 groups at the appropriate place instead of using :STEP at the beginning. :STEP is very useful with :TAGBODY.

By the way, the normal body indentation is taken from the INDENT-BASE field of the LISP edit environment, which is initialized to the width of a capital 'M' in the SEdit default font. The normal indentation step is taken from the INDENT-STEP field of the LISP edit environment, which is initialized to twice the width of a capital 'M' (that is, twice INDENT-BASE). These defaults are chosen so that, in a fixed-width font, the body of a form lines up two characters in from the '(' of the form, and each argument group line up two characters in from the next one (or the body). If you want non-standard values for either of these parameters, you can change the values in the LISP edit environment and then reinitialize

your SEdit formats.  Also, if you change font profiles, reinitializing SEdit will fix up the indents appropriately.