APPENDIX B. SEDIT—THE LISP EDITOR

SEdit is the Lisp structure editor. It allows you to edit Lisp code directly in memory. This editor replaces DEdit in Chapter 16, Structure Editor, of the *Interlisp-D Reference Manual*. First introduced in Lyric, the SEdit structure editor has been greatly enhanced in the Medley release. Medley additions are indicated with revision bars in the right margin.

16.1 SEdit - The Structure Editor

As a structure editor, SEdit alters Lisp code directly in memory. The effect this has on the running system depends on what is being edited.

For Common Lisp definitions, SEdit always edits a copy of the object. For example, with functions, it edits the definition of the function. What the system actually runs is the installed function, either compiled or interpreted. The primary difference between the definition and the installed function is that comment forms are removed from the definition to produce the installed function. The changes made while editing a function will not be installed until the edit session is complete.

For Interlisp functions and macros, SEdit edits the actual structure that will be run. An exception to this is an edit of an EXPR definition of a compiled function. In this case, changes are included and the function is unsaved when the edit session is completed.

SEdit edits all other structures, such as variables and property lists, directly. SEdit installs all changes as they are made.

If an error is made during an SEdit session, abort the edit with an Abort command (see Section 16.1.7, Command Keys). This command undoes all changes from the beginning of the edit session and exits from SEdit without changing your environment.

If the definition being edited is redefined while the edit window is open, SEdit redisplays the new definition. Any edits on the old definition will be lost. If SEdit was busy when the redefinition occurred, the SEdit window will be gray. When SEdit is no longer busy, position the cursor in the SEdit window and press the left mouse button; SEdit will get the new definition and display it.

16.1.1 An Edit Session

The List Structure Editor discussion in Chapter 3, Language Integration, explains how to start an editor in Lisp.

Whenever you call SEdit, a new SEdit window is created. This SEdit window has its own process, and thus does not rely on an Exec to run in. You can make edits in the window, shrink it while you do something else, expand it and edit some more, and finally close the window when you are done.

Throughout an edit session, SEdit remembers everything that you do through a change history. All edits can be undone and redone sequentially. When an edit session ends, SEdit forgets this information and installs the changes in the system.

The session ends with an event signalling to the editor that changes are complete. Three events signal completion:

- Closing the window.

Do this to terminate the edit session when you are finished.

- Shrinking the window.

Shrink the window when you have made some edits and may want to continue the editing session at a later time.

- Typing one of the Completion Commands, listed below.

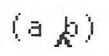
Each of these commands has the effect of installing your changes, completing the edit, and returning the TTY process to the Exec. They vary in what is done in addition to completing. Using these commands the definition that you were editing can be automatically compiled, the edit window can be closed, or both.

A new edit session begins when you come back to an SEdit after completing. The change history is discarded at this point.

If the Exec is waiting for SEdit to return before going on, complete the edit session using any of the methods above to alert the Exec that SEdit is done. The TTY process passes back to the Exec .

16.1.2 SEdit Carets

There are two carets in SEdit, the edit caret and the structure caret. The edit caret appears when characters are edited within a single structure, such as an atom, string, or comment. Anything typed in will appear at the edit caret as part of the structure that the caret is within. The edit caret looks like this:



The structure caret appears when the edit point is between structures, so that anything inserted will go into a new structure. It looks like this:

SEdit changes the caret frequently, depending on where you are in the structure you are editing, and how the caret is positioned. The left mouse button allows an edit caret position to be set.

The middle mouse button allows the structure caret position to be set .

16.1.3 The Mouse

In SEdit, the mouse buttons are used as follows. The left mouse button positions the mouse cursor to point to parts of Lisp structures. The middle mouse button positions the mouse cursor to point to whole Lisp structures. Thus, selecting the Q in LEQ using the left mouse button selects that character, and sets the edit caret after the Q:

```
(LEQ n 1)
```

Any characters typed in at this point would be appended to the atom LEQ.

Selecting the same letter using the middle mouse button selects the whole atom (this convention matches TEdit's character/word selection convention), and sets a structure caret between the LEQ and the n:

```
(LEQ n 1)
```

At this point, any characters typed in would form a new atom between the LEQ and the n.

Larger structures can be selected in two ways. Use the middle mouse button to position the mouse cursor on the parenthesis of the desired list to select that list. Press the mouse button multiple times, without moving the mouse, extends the selection. Using the previous example, if the middle button were pressed twice, the list (LEQ ...) would be selected:

```
(LEQ n 1)
```

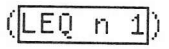
Pressing the button a third time would cause the list containing the (LEQ n 1) to be selected.

The right mouse button positions the mouse cursor for selecting sequences of structures or substructures. Extended selections are indicated by a box enclosing the structures selected. The selection is extended in the same mode as the original selection. That is, if the original selection were a character selection, the right button could be used to select more characters in the same atom. Extended selections also have the property of being marked for pending deletion. That is, the selection takes the place of the caret, and anything typed in is inserted in place of the selection.

For example, selecting the E by pressing the left mouse button and selecting the Q by pressing the right mouse button would produce:

```
(LEQ n 1)
```

Similarly, pressing the middle mouse button and then selecting with the right mouse button extends the selection by whole structures. Thus, in our example, pressing the middle mouse button to select LEQ and pressing the right mouse button to select the 1 would produce:



This is not the same as selecting the entire list, as above. Instead, the elements in the list are collectively selected, but the list itself is not.

16.1.4 Gaps

The SEdit structure editor requires that everything edited must have an underlying Lisp structure, even if the structure is not directly displayed. For example, with quoted forms the actual structure might be (**QUOTE** GREEN), although this would be displayed as 'GREEN. Even when the user is in the midst of typing in a form, the underlying Lisp structure must exist.

Because of this necessity, SEdit provides gaps to serve as dummy Lisp objects during typing. SEdit does not need a gap for every form typed in, but gaps are necessary for quoted objects. When something is typed that requires SEdit to build a Lisp structure and thus create a gap, as the quote character does, the gap will appear marked for pending deletion. This means it is ready to be replaced by the structure to be typed in. In this way it is possible to type special structures, like quotes, directly, while SEdit maintains the structure.

A gap looks like:

A gap displayed after a quote has been typed in would look like this:

with the gap marked for pending deletion, ready for typein of the object to be quoted.

16.1.5 Broken Atoms

When you are typing an atom (a symbol or a number), SEdit saves the characters you type until you finish the atom. SEdit determines that you've finished the atom when you type a character that cannot (without being escaped) belong to an atom, such as a space or open parenthesis. SEdit then tries to create an atom with these characters, just as if it were the Lisp reader. If it succeeds, the atom becomes part of the structure you're editing. However, if it fails, SEdit intercepts the reader error that would otherwise occur and instead creates a special SEdit structure called a Broken-Atom. A Broken-Atom looks and behaves in SEdit just like a normal atom, but is printed in italics to alert you to its needing correction.

SEdit has to create a Broken-Atom when the characters typed don't make a legal atom. For example, the characters

"DECLARE:" cannot make a symbol because the colon is a package specifier, but the form is not correct for a package-qualified symbol. Similarly, the characters "#b123" cannot represent an integer in base two, because 2 and 3 are not legal digits in base two, so SEdit would make a Broken-Atom that looks like *#b123*.

Broken-Atoms can be edited in SEdit just like real atoms. Whenever you finish editing a Broken-Atom, SEdit again tries to create an atom from the characters. If it succeeds, it reprints the atom in SEdit's default font, rather than in italics. You should be sure to correct any Broken-Atoms you create before exiting SEdit, since Broken-Atoms do not behave in any useful way outside SEdit.

16.1.6 Special Characters

A few characters have special meaning in Lisp, and are treated specially by SEdit. SEdit must always have a complete structure to work on at any level of the edit. This means that SEdit needs a special way to type in structures such as lists, strings, and quoted objects. In most instances these structures can be typed in just as they would be to a regular Exec, but in a few cases this is not possible.

Lists- (and)

Lists begin with an open parenthesis character **(**. Typing an open parenthesis gives a balanced list, that is, SEdit inserts both an open and a close parenthesis. The structure caret is between the two parentheses. List elements can be typed in at the structure caret. When a close parenthesis, **)** is typed, the caret will be moved outside the list (and the close parenthesis), effectively finishing the list. Square bracket characters, [and], have no special meaning in SEdit, as they have no special meaning in Common Lisp.

Quoted Structures:

SEdit handles the quote keys so that it is possible to type in all quote forms directly. When typing one of the following quote keys at a structure caret, the quote character typed will appear, followed by a gap to be replaced by the object to be quoted.

Single Quote – ' Use to enter quoted structures.

Backquote –' Use to enter backquoted structures.

Comma – , Use to enter comma forms, as used with a Backquote form.

At Sign – @ Use after a comma to create a comma-at-sign gap. This allows type-in of comma-at forms, e.g. **,@list**, as used within a Backquote form.

Dot – . Use the dot (period) after a comma to create a comma-dot gap. This allows type-in of comma-dot forms, e.g. **,.list**, as used within a Backquote form.

Hash Quote – #' Use this two character sequence to enter the **CL:FUNCTION** abbreviation hash–quote (#').

Dotted Lists:

The dot, or period, character (.) is used to type dotted lists in SEdit. After typing a dot, SEdit inserts a dot and a gap to fill in

for the tail of the list. To dot an existing list, point the cursor between the last and second to the last element in the list, and type a dot. To undot a list, select the tail of the list before the dot while holding down the SHIFT key.

Escape- \ or %

Use to escape from a specific typed in character. Use the escape key to enter characters, like parentheses, which otherwise have special meaning to the SEdit reader. Press the escape key then type in the character to escape. SEdit uses the escape key appropriate to the environment it is editing in; it depends on the readtable that was current when the editor was started. The backslash key (\) is used when editing Common Lisp, and the percent key (%) is used when editing Interlisp.

Multiple Escape- |

Use the multiple escape key, the vertical bar character (|), to escape a sequence of typed in characters. SEdit always balances multiple escape characters. When one multiple escape character is typed, SEdit produces a balanced pair, with the caret between them, ready for typing in the characters to be escaped. If you type a second vertical bar, the caret moves after the second vertical bar, and is still within the same atom, so that you can add more unescaped characters to the atom.

Comments- ;

The comment key, a semicolon (;), starts a comment. When a semicolon is typed, an empty comment is inserted with the caret in position for typing in the comment. Comments can be edited like strings. There are three levels of comments supported by SEdit: single, double, and triple. Single semicolon comments are formatted at the comment column, about three-quarters of the way across the SEdit window, towards the right margin. Double semicolon comments are formatted at the current indentation of the code that they are in. Triple semicolon comments are formatted against the left margin of the SEdit window. The level of a comment can be increased or decreased by pointing after the semicolon, and either typing another semicolon, or backspacing over the preceding semicolon. Comments can be placed anywhere in your Common Lisp code. However, in Interlisp code, they must follow the placement rules for Interlisp comments.

Strings- "

Enter strings in SEdit by typing a double quote ("). SEdit balances the double quotes. When one is typed, SEdit produces a second, with the caret between the two, ready for typing the characters of the string. If a double quote character is typed in the middle of a string, SEdit breaks the string into two smaller strings, leaving the caret between them.

16.1.7 Commands

SEdit commands are most easily entered through the keyboard. When possible, SEdit uses a named key on the keyboard, for example, the DELETE key. The other commands are either Meta, Control, or Meta-Contol key combinations. For the alphabetic command keys, either uppercase or lowercase will work.

There are two menus available, as an alternative means of invoking commands. They are the middle button popup menu,

and the attached command menu. These menus are described in more detail below.

16.1.8 Editing Commands

Redisplay: Control-L [Editor Command]

Redisplays the structure being edited.

Delete Selection: DELETE [Editor Command]

Deletes the current selection.

Delete Word: Control-W [Editor Command]

Deletes the previous atom or whole structure. If the caret is in the middle of an atom, deletes backward to the beginning of the atom only.

Control-Meta-O [Editor Command]

Performs a fast edit by calling ED with its CURRENT option.

16.1.9 Completion Commands

Abort: Meta-A [Editor Command]

Aborts. This command must be confirmed. All changes since the beginning of the edit session are undone, and the edit is closed.

The following commands signal completion of an edit session and install the structure you were editing.

Control-X [Editor Command]

Signals the system that this edit is complete. The window remains open, though, so the user can see the edit and start editing again directly.

Control-C [Editor Command]

Signals the system that this edit is complete and compiles the definition being edited. The variable *compile-fn* determines the function to be called to do the compilation. See the Options section below.

Control-Meta-X [Editor Command]

Signals the system that this edit is complete and closes the window.

Control-Meta-C [Editor Command]

Signals the system that this edit is complete, compiles the definition being editing, and closes the window.

16.1.10 Undo Commands

Undo: Meta-U or UNDO [Editor Command]

Undoes the last edit. All changes since the beginning of the edit session are remembered, and can be undone sequentially.

Redo: Meta-R or AGAIN [Editor Command]

Redoes the edit change that was just undone. Redo only works directly following an Undo. Any number of Undo commands can be sequentially redone.

16.1.11 Find Commands

Find: Meta-F or FIND [Editor Command]

Finds a specified structure, or sequence of structures. If there is a current selection, SEdit looks for the next occurrence of the selected structure. If there is no selection, SEdit prompts for the structure to find, and searches forward from the position of the caret. The found structure will be selected, so the Find command can be used to easily find the same structure again.

If a sequence of structures is selected, SEdit will look for the next occurrence of the same sequence. Similarly, when SEdit prompts for the structure to find, you can type a sequence of structures to look for.

The variable *wrap-search* controls whether or not SEdit wraps around from the end of the structure being edited and continues searching from the beginning.

Reverse Find: Control-Meta-F [Editor Command]

Finds a specified structure, searching in reverse from the position of the caret.

The variable *wrap-search* controls whether or not SEdit wraps around from the beginning of the structure being edited and continues searching from the end.

Find Gap: Meta-N or SKIP-NEXT [Editor Command]

Skips to the next gap in the structure, leaving it selected for pending deletion.

Substitute: Meta-S or SHIFT-FIND [Editor Command]

Substitutes one structure, or sequence of structures, for another structure, or sequence, within the current selection. SEdit prompts you in the SEdit prompt window for the structures to replace, and the structures to replace with.

The selection to substitute within must be a structure selection. To get a structure selection, click with the middle mouse button (not the left), and extend it, if necessary, with the right mouse button. If you begin with the left button, you will get an informational message "Select the structure to substitute within", because the selection was of characters, rather than structures.

Delete Structure: Control-Meta-S [Editor Command]

Removes all occurences of a structure or sequence of structures within the current selection. SEdit prompts the user in the SEdit prompt window for the structures to delete.

16.1.12 General Commands

Arglist: Meta-H or HELP [Editor Command]

Shows the argument list for the function currently selected, or currently being typed in, in the SEdit prompt window. If the argument list will not fit in the SEdit prompt window, it is displayed in the main Prompt Window.

Convert Comments: Meta-; [Editor Command]

Converts old style comments in the selected structure to new style comments. This converter notices any list that begins with an asterisk (*) in the INTERLISP package (IL:*) as an old style comment. Section 16.1.18, Options, describes the converter options.

Comment Out Selection: Control-Meta-; [Editor Command]

This command puts the contents of a structure selection into a comment. This provides an easy way to "comment out" a chunk of code. The Extract command can be used to reverse this process, returning the comment to the structures contained therein.

Edit: Meta-O [Editor Command]

Edits the definition of the current selection. If the selected name has more than one type of definition, SEdit asks for the type to be edited. If the selection has no definition, a menu pops up. This menu lets the user specify either the type of definition to be created, or no definition if none needs to be created.

Eval: Meta-E [Editor Command]

Evaluates the current selection. If the result is a structure, the inspector is called on it, allowing the user to choose how to look at the result. Otherwise, the result is printed in the SEdit prompt window. The evaluation is done in the process from which the edit session was started. Thus, while editing a function from a break window, evaluations are done in the context of the break.

Expand: Meta-X or EXPAND [Editor Command]

Replaces the current selection with its definition. This command can be used to expand macros and translate CLISP.

Extract: Meta-/ [Editor Command]

Extracts one level of structure from the current selection. If there is no selection, but there is a structure caret, the list containing the caret is used. This command can be used to strip the parentheses off a list, or to unquote a quoted structure, or to replace a comment with the structures contained therein.

Inspect: Meta-I [Editor Command]

Inspect the current selection.

Join: Meta-J [Editor Command]

Joins. This command joins any number of sequential Lisp objects of the same type into one object of that type. Join is supported for atoms, strings, lists, and comments. In addition, SEdit permits joining of a sequence of atoms and strings, since either type can easily be coerced into the other. In this case, the result of the Join will be an atom if the first object in the selection is an atom, otherwise the result will be a string.

Mutate: Meta-Z [Editor Command]

Mutates. This command allows the user to do arbitrary operations on a LISP structure. First select the structure to be mutated (it must be a whole structure, not an extended selection). When the user presses Meta-Z SEdit prompts for the function to use for mutating. This function is called with the selected structure as its argument, and the structure is replaced with the result of the mutation.

For example, an atom can be put in upper case by selecting the atom and mutating by the function U-CASE. You can replace a structure with its value by selecting it and mutating by EVAL.

Quote: Meta-'
Meta-`
Meta-,
Meta-.
Meta-@ or Meta-2
Meta-# or Meta-3 [Editor Command]

Quotes the current selection with the specified kind of quote, respectively, Single Quote, Backquote, Comma, Comma-At-Sign, Comma-Dot, or Hash-Quote.

Normalize Selection: Meta-Space or Meta-Return [Editor Command]

Scrolls the current selection to the center of the window. Similarly, the Space or Return key can be used to normalize the caret.

Parenthesize: Meta-) or Meta-0 [Editor Command]

Parenthesizes the current selection, positioning the caret after the new list.

Parenthesize: Meta- (or Meta-9 [Editor Command]

Parenthesizes the current selection, positioning the caret at the beginning of the new list. Only a whole structure selection or an extended selection of a sequence of whole structures can be parenthesized.

16.1.13 Miscellaneous

Change Print Base: Meta-B [Editor Command]

Changes Print Base. Prompts for entry of the desired Print Base, in decimal. SEdit redisplays fixed point numbers in this new base.

Set Package: Meta-P [Editor Command]

Changes the current package for this edit. Prompts the user, in the SEdit prompt window, for a new package name. SEdit will redisplay atoms with respect to that package.

Attached Menu: Meta-M [Editor Command]

Attaches a menu of the commonly used commands (the SEdit Command Menu) to the top of the SEdit window. Each SEdit window can have its own menu, if desired.

16.1.14 Help Menu

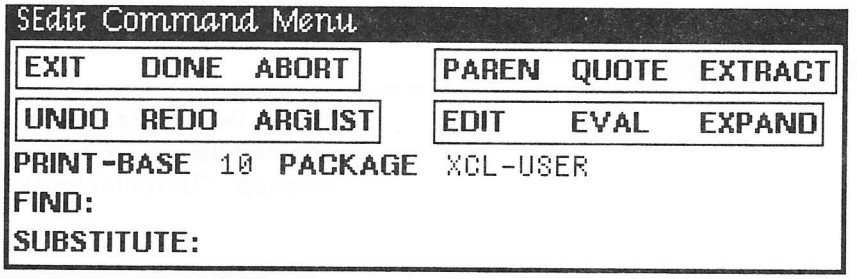
When the mouse cursor is positioned in the SEdit title bar and the middle mouse button is pressed, a Help Menu of commands pops up. The menu looks like this:

Commands	
Abort	M-A
Done	C-X
Done & Compile	C-C
Done & Close	M-C-X
Done, Compile, & Close	M-C-C
Undo	M-U
Redo	M-R
Find	M-F
Reverse Find	M-C-F
Remove	M-C-S
Substitute	M-S
Find Gap	M-N
Arglist	M-H
Convert Comment	M-;
Edit	M-O
Eval	M-E
Expand	M-X
Extract	M-/
Inspect	M-I
Join	M-J
Mutate	M-Z
Parenthesize	M-(
Quote	M-'
Set Print-Base	M-B
Set Package	M-P
Attach Menu	M-M

The Help Menu lists each command and its corresponding Command Key. (In the menu, the letter C stands for CONTROL, while M indicates Meta.) The command selected is executed just as if the command had been entered from the keyboard. The menu remembers which command was selected last, and pops up with the mouse cursor next to that same command the next time the menu is used. This provides a very fast way to repeat the same command when using the mouse.

16.1.15 Command Menu

The SEdit Attached Command Menu contains the commonly used commands. Use the Meta-M keyboard command to bring up this menu. The menu can be closed, independently of the SEdit window, when desired. The menu looks like:



All of the commands in the menu function identically to their corresponding keyboard commands, except for Find and Substitute.

When Find is selected with the mouse cursor, SEdit prompts in the menu window, next to the Find button, for the structures to find. Type in the structures then select Find again. The search begins from the caret position in the SEdit window.

Similarly, Substitute prompts, next to the Find button, for the structures to find, and next to the Substitute button for the structures to substitute with. After both have been typed in, selecting Substitute replaces all occurrences of the Find structures with the Substitute structures, within the current selection.

To do a confirmed substitute, set the edit point before the first desired substitution, and select Find. Then if you want to substitute that occurrence of the structure, select Substitute. Otherwise, select Find again to go on.

Selecting either Find or Substitute with the right mouse button erases the old structure to find or substitute from the menu, and prompts for a new one.

16.1.16 SEdit Programmer's Interface

The following sections describe SEdit's programmer's interface. All symbols are external in the package named "SEdit".

16.1.17 SEdit Window Region Manager

SEdit provides user redefinable functions which control how SEdit chooses the region for a new edit window.

(get-window-region *context reason name type***)** [Function]

This function is called when SEdit wants to know where to place a window it is about to open. This happens whenever the user starts a new SEdit or expands an Sedit icon.The default behavior is to pop a window region off SEdit's stack of regions that have been used in the past. If the stack is empty, SEdit prompts for a new region.

This function can be redefined to provide different behavior. It is called with the edit *context*, a *reason* for needing a region, the *name* of the structure to be edited, and the *type* of the structure to be edited. The edit *context* is SEdit's main data structure and can be useful for associating particular edits with specific regions. The *reason* argument specifies why SEdit wants a region, and will be one of the keywords :CREATE or :EXPAND.

(save-window-region *context reason name type region***)** [Function]

This function is called whenever SEdit is finished with a region and wants to make the region available for other SEdits. This happens whenever an SEdit window is closed or shrunk, or when an SEdit Icon is closed. The default behavior is simply to push the region onto SEdit's stack of regions.

This function can be redefined to provide different behavior. It is also called with the edit *context*, the *reason*, the *name*, the *type*, and additionally the window *region* that is being released. The *reason* argument specifies why SEdit is releasing the region, and will be one of the keywords :CLOSE, :SHRINK, or :CLOSE-ICON.

keep-window-region [Variable]

Default **T**. This flag determines the behavior of the default SEdit region manager, explained above, for shrinking and expanding windows. When set to **T**, shrinking an SEdit window will not give up that window's region; the icon will always expand back into the same region. When set to **NIL**, the window's region is made available for other SEdits when the window is shrunk. Then when an SEdit icon is expanded, the window will be reshaped to the next available region.

This variable is only used by the default implementations of the functions **get-window-region** and **save-window-region**. If these functions are redefined, this flag is no longer used.

16.1.18 Options

The following parameters can be set as desired.

wrap-parens [Variable]

This SEdit pretty printer flag determines whether or not trailing close parenthesis characters,), are forced to be visible in the window without scrolling. By default it is set to NIL, meaning that close parens are allowed to "fall off" the right edge of the window. If set to T, the pretty printer will start a new line before the structure preceding the close parens, so that all the parens will be visible.

wrap-search [Variable]

This flag determines whether or not SEdit find will wrap around to the top of the structure when it reaches the end, or vice versa in the case of reverse find. The default is NIL.

clear-linear-on-completion [Variable]

This flag determines whether or not SEdit completely re-pretty prints the structure being edited when you complete the edit. The default value is NIL, meaning that SEdit reuses the pretty printing.

convert-upgrade [Variable]

Default 100. When using Meta-; to convert old-style single-asterisk comments, if the length of the comment exceeds **convert-upgrade** characters, the comment is converted into a double semicolon comment. Otherwise, the comment is converted into a single semicolon comment.

Old-style double-asterisk comments are always converted into new-style triple-semicolon comments.

16.1.19 Control Functions

(reset) [Function]

This function recomputes the SEdit edit environment. Any changes made in the font profile, or any changes made to SEdit's commands are captured by resetting. Close all SEdit windows before calling this function.

(add-command *key-code form &optional scroll? key-name command-name help-string***)** [Function]

This function allows you to write your own SEdit keyboard commands. You can add commands to new keys, or you can redefine keys that SEdit already uses as command keys. If you mistakenly redefine an SEdit command, the funtion Reset-Commands will remove all user-added commands, leaving SEdit with its default set of commands.

key-code can be a character code, or any form acceptible to il:charcode.

form determines the function to be called when the key command is typed. It can be a symbol naming a function, or a list, whose first element is a symbol naming a function and the rest of the elements are extra arguments to the function. When the command is invoked, SEdit will apply the function to the edit context (SEdit's main data structure), the charcode that was typed, and any extra arguments supplied in *form*. The extra arguments do not get evaluated, but are useful as keywords or flags, depending on how the command was invoked. The command function must return T if it handled the command. If the function returns NIL, SEdit will ignore the command and insert the character typed.

The first optional argument, *scroll?*, determines whether or not SEdit scrolls the window after running the command. This

argument defaults to NIL, meaning don't scroll. If the value of SCROLL is T, then SEdit will scroll the window to ensure that the caret is visible.

The rest of the optional arguments are used to add this command to SEdit's middle button menu. When the item is selected from the menu, the command function will be called as described above, with the charcode argument set to NIL.

key-name is a string to identify the key (combination) to be typed to invoke the command. For example "M-A" to represent the Meta-A key combination, and "C-M-A" for Control-Meta-A.

command-name is a string to identify the command function, and will appear in the menu next to the *key-name*.

help-string is a string to be printed in the prompt window when a mouse button is held down over the menu item.

After adding all the commands that you want, you must call Reset-Commands to install them.

For example:

```
(add-command "↑U" (my-change-case t))

(add-command "↑Y" (my-change-case nil))

(add-command "1,r" my-remove-nil
  "M-R" "Remove NIL"
  "Remove NIL from the selected structure"))

(reset-commands)
```

will add three commands. Suppose `my-change-case` takes the arguments *context*, *charcode*, and *upper-case*?. *upper-case*? will be set to T when `my-change-case` is called from Control-U, and NIL when called from Control-Y. `my-remove-nil` will be called with only *context* and *charcode* arguments when Meta-R is typed.

Below are some SEdit functions which are useful in writing new commands.

(reset-commands) [Function]

This function installs all commands added by **add-command**. SEdits which are open at the time of the **reset-commands** will not see the new commands; only new SEdits will have the new commands available.

(default-commands) [Function]

This function removes all commands added by **add-command**, leaving SEdit with its default set of commands. As in **reset-commands**, open SEdits will not be changed; only new SEdits will have the user commands removed.

(get-prompt-window *context***)** [Function]

This function returns the attached prompt window for a particular SEdit.

(get-selection *context***)** [Function]

This function returns two values: the selected structure, and the type of selection, one of NIL, T, or :SUB-LIST. The selection type NIL means there is not a valid selection (in this case the structure is meaningless). T means the selection is one complete structure. :SUB-LIST means a series of elements in a list is selected, in which case the structure returned is a list of the elements selected.

(replace-selection *context structure selection-type* **)** [Function]

This function replaces the current selection with a new structure, or multiple structures, by deleting the selection and then inserting the new structure(s). The *selection-type* argument must be one of T or :SUB-LIST. If T the *structure* is inserted as one complete structure. If :SUB-LIST, the *structure* is treated as a list of elements, each of which is insertd.

getdef-fn [Variable]

This function is called with the arguments *name*, *type*, and *olddef*, when SEdit needs to refetch the definition for the named object being edited. When SEdit is first started it gets passed the structure, so this function doesn't get called. But after completion, SEdit refetches because it doesn't know if the Edit Interface (File Manager) changed the definition upon installation. The function returns the new definition.

fetch-definition-error-break-flag [Variable]

This flag, along with the error options listed below, determines what happens when the getdef-fn errors. The default value is NIL, causing errors to be suppressed. When set to T, the break will be allowed.

getdef-error-fn [Variable]

This function is funcalled with the arguments *name*, *type*, *olddef*, and *prompt-window*, when the getdef-fn errors, independent of whether or not the break is suppressed. This function should return the structure to be used in place of the unavailable new definition.

edit-fn [Variable]

This function is funcalled with the selected structure and the edit options as its arguments from the Edit (M-O) command. It should start the editor as appropriate, or else generate an error if the selection is not editable.

compile-fn [Variable]

This function is funcalled with the arguments *name*, *type*, and *body*, from the compile completion commands. It should compile the definition, *body*, and install the code as appropriate.

(sedit *structure props options***)** [Function]

This function provides a means of starting SEdit directly. *structure* is the structure to be edited.

props is a property list, which may specify the following properties:

:name - the name of the object being edited

:type - the file manager type of the object being edited. If NIL, SEdit will not call the file manager when it tries to refetch the definition it is editing. Instead, it will just continue to use the structure that it has.

:completion-fn - the function to be called when the edit session is completed. This function is called with the *context*, *structure*, and *changed?* arguments. *context* is SEdits main data structure. *structure* is the structure being edited. *changed?* specifies if any changes have been made, and is one of NIL, T, or :ABORT, where :ABORT means the user is aborting the edit and throwing away any changes made. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the main arguments above.

:root-changed-fn - the function to be called when the entire structure being edited is replaced with a new structure. This function is called with the new structure as its argument. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the structure argument.

options is one or a list of any number of the followng keywords:

:fetch-definition-suppress-errors - If this option is provided, any error under the getdef-fn will be suppressed, regardless of the :fetch-definition-allow-errors option or the value of ***fetch-definition-error-break-flag***.

:fetch-definition-allow-errors - If this option is provided, any error under the getdef-fn will be allowed to break.

:dontwait - This option specifies that the call to **SEdit** should return as soon as the editor is started, rather than waiting for a completion command.

:close-on-completion - This option specifies that SEdit cannot remain active for multiple completions. That is, the SEdit window cannot be shrunk, and the completion commands that normally leave the window open will in this case close the window and terminate the edit.

:compile-on-completion - This option specifies that SEdit should call the *compile-fn* to compile the definition being edited upon completion, regardless of the completion command used.

Warning with Declarations

CAUTION: There is a feature of the BYTECOMPILER that is not supported by SEdit or the XCL compiler. It is possible to insert a comment at the beginning of your function that looks like

```
(* DECLARATIONS: --)
```

The tail, or -- section, of this comment is taken as a set of local record declarations which are then used by the compiler in that function just as if they had been declared globally. See the "Compiler" section in Chapter 3 of these Notes for additional behavior in XCL.

SEdit does not recognize such declarations. Thus, if the "Expand" command is used, the expansion will not be done with these record declarations in effect. The code that you see in SEdit will not be the same code compiled by the BYTECOMPILER.