Editing Interlisp Code with SEdit

The Interlisp editing definitions configure SEdit as an editor for programs written in Interlisp-D, and are ultimately intended as a replacement for DEdit, the system display editor. Although the current system is still missing many convenience features, it currently provides a workable alternative to DEdit. This document provides detailed information on using SEdit as a code editor. It is assumed that the reader has read the introduction to SEdit, and is familiar with the Interlisp-D programming environment. This part of SEdit is under active development; this document will be changed as improvements are made.

Running SEdit

After loading SEdit, the function `SEdit` allows it to be installed and de-installed as the default system display editor. Executing `(EDITMODE 'SEDIT)` will cause future edit requests (from functions such as `DF`, from Masterscope, and from inspectors and browsers) to use SEdit instead of DEdit; executing `(EDITMODE 'DEDIT)` will revert to using DEdit. The function will return the previous editor state (`SEDIT` or `DEDIT`).

Unlike DEdit, SEdit does not run in the process which invokes it. This has some important effects:

a) SEdit processes can be started and stopped in any order. The windows may be shrunken and kept around indefinitely if desired.

b) Calls to editing functions such as `DF` return as soon as the process is started, rather than waiting for the editing to be completed (however, when SEdit is invoked from Masterscope, it forces Masterscope to wait until the user indicates that they are done editing (by closing or shrinking the window); this is simply a convenience to avoid `Edit where any` commands from immediately covering the screen in hundreds of edit windows).

c) As a consequence of (b), some functions normally performed by `DF` (such as informing the file package that the function has been changed and needs to be saved, unsaving the definition of a compiled function, or updating the "last edited" date) are instead performed by SEdit, and often at different times (since SEdit can't wait until the user is "done editing").

Commands

At present, SEdit has no attached menu of commands. Many of the commands in DEdit's menu (such as `Before`, `After` and `Replace`) are completely unnecessary in SEdit (because of its more uniform interface). Some of them (such as `Delete`, and soon `()in` and `()out`) are provided by keyboard commands. A menu may be added in the future with the introduction of more obscure commands.

Pointing and Selecting

Like TEdit, SEdit maintains a current insertion point at which typed, copied, or moved material will be inserted. The point is set by moving the mouse to the desired position and clicking a mouse button, and is indicated by a flashing caret. Unlike TEdit, SEdit has two types of points: structure points and character points. Structure points are allowed within non-atomic structures which have a variable number of components (i.e. lists); they indicate that another Lisp structure can be inserted at the indicated position. In normal (NIL-terminated) lists, structure points can be placed before the first element, after the last element, or between any two adjacent elements; in a non-NIL terminated (dotted) list, points may not be placed anywhere after the dot. Character points are positions at which individual characters may be inserted (rather than whole Lisp structures), and are allowed in atoms and strings. The caret changes to reflect the type of point: structure points look like '▲', and character points look like '∧'.

Similarly, both structures and individual characters can be selected. A selection may be

- one of the characters in the pname of an atom or string

- a sequence of consecutive characters in the pname of an atom or string
- a lisp structure presented as an entity (e.g. an entire list, string, atom, quoted structure, etc.)
- a sequence of such structures appearing consecutively in a list

Note that not all lisp structures are presented as distinct entities, and so not all will be selectable. For instance, the individual cons cells comprising a list are usually not separately selectable. Also, extra characters added to the presentation as punctuation are not individually selectable; you can't select the left parenthesis of a list, or the closing quotation mark of a string.

The type of selection and point made depends on the mouse button used. The left button selects characters and places character points; the middle button selects structures and places structure points (this is supposed to be reminiscent of TEdit). The right button extends the current selection to the smallest selection which covers the current mouse position. As an added convenience, clicking with the left or middle button more than once in the same spot will enlarge the current selection one step, through enclosing layers of structure.

Inserting and Replacing

To insert characters in an atom or string, use a mouse button to place a character point at the desired location and just type the characters. As each character is typed, it will be inserted and the caret point will be moved after it. The `Backspace` key deletes the character to the left of the caret.

To insert new structures in lists, place a structure point at the desired location and type one of

- a left parenthesis to insert a new list
- a double quotation mark (`"`) to insert a new string
- a normal character (i.e. one with syntax class OTHER) to insert an atom beginning with that character

In the first case, an empty list will be inserted and the caret will be moved inside it. In the second, an empty string will be inserted and the caret will become a character point inside the string. In the third case, a new atom will be inserted, and the caret will become a character point to allow appending more characters to the atom's name.

There are a few other characters which are recognized specially:

- a right parenthesis places the caret point immediately after the list immediately enclosing it
- a double quotation mark, while inserting characters in a string, places the caret point immediately after the string (if it was after the last character in the string) or splits the string into two strings (if it was between two characters)
- a blank or carriage return, while inserting characters in an atom, places the caret point immediately after the atom (if it was after the last character) or splits the atom into two atoms (if it was between two characters)

These characters all leave the caret point ready to read another structure. The rules may sound a little bizarre, but they work out to give just the right behavior — typing in the printed representation of a lisp structure will give you that structure. (At present, this only works for (undotted) lists, string, litatoms, and numbers; soon dotted lists and quoted structures will also be implemented).

Special characters, such as parentheses, spaces, and double quotation marks, can be inserted in atoms by preceding them with the escape character (a percent sign).

To replace structure, it is selected "pending delete". Pending deletion selections are made whenever the current selection is extended using the right mouse button (as in TEdit). To distinguish them from normal selections they are displayed by outlining the selected material, rather than underlining it. When structures or characters have been selected pending delete, typing anything will cause them to be replaced with the new material.

Copying, Moving, and Deleting

Structures or characters may be copied or moved from one part of an SEdited structure to another, between two SEdited structures, or between an SEdited structure and any other Interlisp process which will accept or produce character string representations of lisp structures. To copy material, place a point of the appropriate type at the desired destination, and then select the desired material while depressing the `Copy` key (`Shift` on the Dorado keyboard). Selections made with the `Copy` key depressed will be displayed by a gray underline. As soon as the `Copy` key is released, a copy of the current selection is inserted at the point. If the TTY process is a non-SEdit process, a printed representation of the selected material will be BKSYSBUFed for it to read.

Moving material is done in a similar fashion,except that the `Move` key is depressed while making the selection. Move selections are displayed by a gray outline. As soon as the `Move` key is released, the selected material will be inserted at the current caret point and deleted from its original position. On the Dorado keyboard, Move selections are indicated by depressing both the `Shift` and `Control` keys.

Material may also be deleted in this fashion. Depressing the `Control` key while making a selection will cause the selected material to be deleted as soon as the `Control` key is released. Delete selections are displayed by inverting the selected material (displaying it white-on-black instead of black-on-white).

Whenever selections are being made, the selection is considered complete only when the mouse buttons *and* any modifier keys (`Copy`, `Move`, etc.) have been released. Thus, selections requiring more than one mouse click (e.g. sequence selections) can be made by keeping the modifier key depressed throughout the process. Alternatively, if the wrong modifier key is initially depressed, it can be released and another depressed as long as a mouse button is held down during this time. To completely abort the selection, click a mouse button outside the window before releasing the modifier key.

There are two other ways of deleting material. The `Backspace` key, as was previously mentioned, deletes the character to the left of the caret (this strictly true only when the caret is a character point in an atom or string; at other times it does other, reasonable things — you'll have to try it out to find out exatly what). The `Delete` key deletes the current selection. (Note that this is different from the `Control` key, which is a selection modifier; with `Delete` a normal selection is made and then the `Delete` key is depressed, while the `Control` key is depressed while the selection is actually being made — the choice of which is to use is a matter of personal taste.)

Formatting

SEdit attempts to display the structure being edited in as readable a fashion as possible, while keeping within the width of the display window. It uses fairly conventional rules for pretty-printing lisp, augmented with some special formatting rules for Interlisp special forms (e.g. LAMBDA expressions and CLisp). The components of these special forms are given indentation based on their function within the form, and special keywords are displayed in bold face to improve readability. Some effort is made to propagate the width constraint information so that relatively uniform indentation is used, rather than having complex nested expressions end up mashed against the right edge of the window. In extreme cases SEdit will extend the presentation past the right edge of the window rather than produce too ugly a presentation. If this happens, a horizontal scroll bar will be added to the window to allow editing the whole presentation.

SEdit Windows

The windows SEdit creates behave like all good Interlisp windows. They may be moved, reshaped, and scrolled. If they are reshaped to a different width, the formatting is recomputed to make the best use of the available space. The may be shrunk, producing a relatively unexciting icon adorned with the name of the variable or function being edited. When they are shrunk, the process reading commands from the keyboard is deleted (to save stack space, and allow keeping a large number of SEdit windows around), but it is automatically recreated as soon as the window is expanded again, so this should be effectively invisible. Closing an SEdit window or icon terminates the editing and releases the data structures used.

Unlike DEdit, it is not clear when to consider an SEdit editing session complete. The user may start an edit process, do some editing, shrink the window, expand it again later, etc. For some purposes it is important to have such a notion. For instance, the file package must be informed when a function is edited, so that the new definition can be saved to the appropriate file. If a function has been compiled, and the source is then edited, the compiled code should be discarded. If the editor is invoked on a sequence of structures by Masterscope or EDITFNS, it must know when it is time to go on to the next structure. When a function has been edited, a new comment should be added recording the time and date and the initials of the programmer. All of these cases require some way of indicating that this set of editing operations is completed, even if the editing process is not to be terminated. At present, SEdit handles this by assuming that editing is complete when the window is closed or shrunk.

Confusing SEdit

SEdit currently assumes that no changes will be made to the structure being edited during an edit session, other than those made by SEdit itself. Of course, since SEdit exists in a lisp environment replete with shared structures and destructive functions, there is no way to enforce this. For instance, while editing a variable whose value is a list you could (from an executive window) run a function to destructively change that list. There is no way for SEdit to notice the change as it happens, and at present it will not recover gracefully if it encounters the inconsistency later on.

To avoid causing these problems to itself, SEdit automatically avoids starting two edit processes on the same variable or function. This is only a partial solution, however, and the user is advised to watch out for this problem. If you suspect that a structure being edited has been changed by some other process, simply close the window and start another edit process. This will force SEdit to reexamine the structure.

In the immediate future SEdit will be fixed to detect the most common case of such changes, namely corrections made by DWIM. It is also planned to make SEdit much more robust in the face of other changes it may detect (in the not quite as immediate future).