

Many important objects such as function definitions, property lists, and variable values are represented as list structures. There are two list structure editors (SEdit in the system, and DEdit in the library, for backward compatibility) to allow users to modify list structures rapidly and conveniently.

Description

The list structure editor is most often used to edit function definitions. Editing function definitions in memory is a facility not offered by many Lisp systems, where typically the user edits external text files containing function definitions, then loads them into the environment. In Lisp, function definitions are edited in the environment, and written to an external file using the file manager (see *IRM*), which provides tools for managing the contents of a file.

History

Early implementations of Interlisp using primitive terminals offered a teletype-oriented editor, which included a large set of cryptic commands for printing different parts of a list structure, searching a list, replacing elements, etc. The library includes an extended, display-oriented version of the teletype list structure editor, called DEdit.

The teletype editor is still available (see *IRM*), as it offers a facility for doing complex modifications of program structure under program control. DEdit also provides facilities for using the teletype editor commands from within DEdit.

DEdit

DEdit is a structure oriented, modeless, display based editor for objects represented as list structures, such as functions, property lists, data values, etc.

DEdit incorporates the interfaces of the teletype-oriented Interlisp editor, so the two can be used interchangeably. In addition, the full power of the teletype editor, and indeed the full Interlisp system is easily accessible from within DEdit.

DEdit is structure-oriented rather than character-oriented, to facilitate selecting and operating on pieces of structure as objects in their own right, rather than as collections of characters. However, for the occasional situation when character-oriented editing is appropriate, DEdit provides access to the text editing facilities. DEdit is modeless, in that all commands operate on previously selected arguments, rather than causing the behavior of the interface to change during argument specification.

Requirements

DEDITPP

Installation

Load DEDIT.LCOM from the library.

Loading DEdit makes it the default Lisp structure editor.

If another Lisp structure editor is already the default and you want to make DEdit the default, call (EDITMODE 'DEDIT) after loading DEdit.

User Interface

DEdit is normally called using one of the DEdit functions. See also "Advanced Features" below.

DEdit Window

When DEdit is called for the first time, it prompts for an edit window which is preserved and reused for later DEdits, and it pretty-prints the expression to be edited therein.

Note: The DEdit pretty printer ignores user PRETTYPRINTMACROS because they do not provide enough structural information during printing to enable selection.

The expression being edited can be scrolled by using the standard scroll bar on the left edge of the window. DEdit adds a command menu, which remains active throughout the edit, on the right edge of the edit window. If you type anything, an edit buffer window is positioned below the edit window. This is illustrated in the figure below, which shows the definition of a function called FACT. While DEdit is running, it yields control so that background activities, such as mouse commands in other windows, continue to be performed.

DEdit of function FACT		EditOps
(LAMBDA (X)	(* mjs " 7-Oct-85 16:04")	After
(if (LESSP X 2)		Before
then 1		Delete
else (TIMES X		Replace
(FACT (SUB1 X))))		Switch
		()
		() out
		Undo
		Find
		Swap
		Reprint
		Edit
		EditCom
		Break
		Eval
		Exit
Edit buffer		
(FACT (DIFFERENCE X 1))		

Selecting Objects and Lists

Selection in a DEdit window is as follows:

The left button selects the object being directly pointed at.

The middle button selects the containing list.

The right button extends the current selection to the lowest common ancestor of that selection and the current position.

The only things that may be pointed at are atomic objects (symbols, numbers, etc) and parentheses, which are considered to represent the list they delimit. White space cannot be selected or edited.

When a selection is made, it is pushed on a selection stack, which will be the source of operands for DEdit commands. As each new selection pushes down the selections made before it, this stack can grow arbitrarily deep, so only the top two selections on the stack are highlighted on the screen. This highlighting is done by underscoring the topmost (most recent) selection with a solid black line and the second topmost selection with a dashed line. The patterns used were chosen so that their overlappings would be both visible and distinct, since selecting a subpart of another selection is quite common.

For example, in the next figure, the last selection is the list (FACT (SUB1 X)), and the previous selection is the single symbol SUB1:

```

Dedit of function FACT
(LAMBDA (X) (* mjs " 7-Oct-85 16:04")
  (if (LESSP X 2)
    then 1
    else (TIMES X
              (FACT (SUB1 X))))))

```

Because you can invoke DEdit recursively, there may be several DEdit windows active on the screen at once. This is often useful when transferring material from one object to another (as when reallocating functionality within a set of programs). Selections may be made in any active DEdit window, in any order. When there is more than one DEdit window, the edit command menu (and the type-in buffer) will attach itself to the most recently opened (or current) DEdit window.

Typing Characters to DEdit

Characters may be typed at the keyboard at any time. This will create a type-in buffer window which will position itself under the current DEdit window and do a LISPXREAD (which must be terminated by a right parenthesis or a return) from the keyboard. During the read, any character editing subsystem (such as TTYIN) that is loaded can be used to do character level editing on the type-in. When the read is complete, the type-in will become the current selection (top of stack) and be available as an operand for the next command. Once the read is complete, objects displayed in the type-in buffer can be selected from, scrolled, or even edited, just like those in the main window.

You can also enter editing commands directly into the type-in buffer. Typing control-Z will interpret the rest of the line as a teletype editor command that will be interpreted when the line is closed. Likewise, control-S *OLD NEW* will substitute *NEW* for *OLD*, and control-F *X* will find the next occurrence of *X*.

Copy-Selection

Often, significant pieces of what you wish to type can be found in an active DEdit window. To aid in transferring the keystrokes that these objects represent into the type-in buffer, DEdit supports copy-selection. Whenever a selection is made in the DEdit window with either shift key or the COPY key down, the selection made is not pushed on the selection stack, but is instead unread into the keyboard input (and hence shows up in the type-in buffer). A characteristically different highlighting is used to indicate when copy selection (as opposed to normal selection) is taking place.

Note: Copy-selection remains active even when DEdit is not. Thus you can unread particularly choice pieces of text from DEdit windows into an Exec window.

Entering DEdit Commands

A DEdit command is invoked by selecting an item from the DEdit command menu. This can be done either directly, using the left mouse button in the usual way, or by selecting a subcommand. Subcommands are less frequently used commands than those on the main edit command menu and are grouped together in submenus under the main menu to which they are most closely related.

For example, the teletype editor defines six commands for adding and removing parentheses (defined in terms of transformations on the underlying list structure). Of these six commands, only two (inserting and removing parentheses as a pair) are commonly used, so DEdit provides the other four as subcommands of the common two.

The subcommands of a command are accessed by selecting the command from the commands menu with the middle button. This will bring up a menu of the subcommand options from which a choice can be made. Subcommands are flagged in the list below with the name of the top level command of which they are options.

If you have a large DEdit window, or several DEdit windows active at once, the edit command window may be far away from the area of the screen in which you are operating. To solve this problem, the DEdit command menu is in an attached window. Whenever the tab key is pressed, the command window will move over to the current cursor position and stay there as long as either the tab key remains down or the cursor is in the command window. Thus, you can pull the command window over, slide the cursor into it and then release the tab key (or not) while you make a command selection in the normal way. This eliminates a great deal of mouse movement.

Whenever a change is made, the pretty-printer reprints until the printing stabilizes. As the standard pretty print algorithm is used, and as it leaves no information behind on how it makes its choices, this is a somewhat heuristic process. The REPRINT command can be used to tidy the result up if it is not, in fact, "pretty."

DEdit Functions

The functions used to start an editor are documented in the "Edit Interface" section of the *Lisp Release Notes*.

(RESETDEDIT)

[Function]

Completely reinitializes DEdit. Closes all DEdit windows, so that you must specify the window the next time DEdit is invoked. RESETDEDIT is also used to make DEdit recognize the new values of variables such as DEDITTYPEINCOMS (see "DEdit Parameters," below), when you change them.

DEdit Commands

All commands take their operands from the selection stack, and may push a result back on the stack. In general, the rule is to select target selections first and source selections second. Thus, a REPLACE command is done by selecting the thing to be replaced, selecting (or typing) the new material, and then selecting the REPLACE command in the command menu.

Using *TOP* to denote the topmost (most recent) element of the stack and *NXT* the second element, the DEdit commands are:

AFTER	[DEdit Command]
Inserts a copy of <i>TOP</i> after <i>NXT</i> .	
BEFORE	[DEdit Command]
Inserts a copy of <i>TOP</i> before <i>NXT</i> .	
DELETE	[DEdit Command]
Deletes <i>TOP</i> from the structure being edited. (A copy of) <i>TOP</i> remains on the stack and will appear, selected, in the edit buffer.	
REPLACE	[DEdit Command]
Replaces <i>NXT</i> with a copy of <i>TOP</i> obtained by substituting a copy of <i>NXT</i> wherever the value of the atom EDITEMBEDTOKEN (initially, the & character) appears in <i>TOP</i> . This provides a facility like the MBD edit command in Lisp; see EXTRACT, EMBED and IDIOMS below.	
SWITCH	[DEdit Command]
Exchanges <i>TOP</i> and <i>NXT</i> in the structure being edited.	
()	[DEdit Command]
(IN	[DEdit Command]
Subcommands of (). Inserts (before <i>TOP</i> (like the LI EDIT command; see "Commands That Edit Parentheses," below).	
) IN	[DEdit Command]
Subcommand of (). Inserts) after <i>TOP</i> (like the RI EDIT command; see "Commands That Edit Parentheses," below).	
() OUT	[DEdit Command]
Removes parentheses from <i>TOP</i> .	
(OUT	[DEdit Command]
Subcommand of () OUT. Removes (from before <i>TOP</i> (like the LO EDIT command; see "Commands That Edit Parentheses," below).	
) OUT	[DEdit Command]
Subcommand of () OUT. Removes) from after <i>TOP</i> (like the RO EDIT command; see "Commands That Edit Parentheses," below).	
UNDO	[DEdit Command]
Undoes last command.	

!UNDO [DEdit Command]

Subcommand of UNDO. Undoes all changes since the start of this call on DEdit. This command can be undone.

?UNDO [DEdit Command]

&UNDO [DEdit Command]

Subcommands of UNDO that allow selective undoing of other than the last command. Both of these commands bring up a menu of all the commands issued during this call on DEdit. When you select an item from this menu, the corresponding command (and if &UNDO, all commands since that point) will be undone.

FIND [DEdit Command]

Selects, in place of *TOP*, the first place after *TOP* that matches *NXT*. Uses the edit subsystem's search routine, so supports the full wildcarding conventions of EDIT.

SWAP [DEdit Command]

Exchanges *TOP* and *NXT* on the stack, i.e. the stack is changed, the structure being edited isn't.

SWAP and its subcommands affect the stack and the selections, rather than the structure being edited.

CENTER [DEdit Command]

Subcommand of SWAP. Scrolls until *TOP* is visible in its window.

CLEAR [DEdit Command]

Subcommand of SWAP. Discards all selections (i.e., clears the stack).

COPY [DEdit Command]

Subcommand of SWAP. Puts a copy of *TOP* into the edit buffer and makes it the new *TOP*.

POP [DEdit Command]

Subcommand of SWAP. Pops *TOP* off the selection stack.

REPRINT [DEdit Command]

Reprints *TOP*.

EDIT [DEdit Command]

Runs DEdit on the definition of the atom *TOP* (or *CAR* of list *TOP*). Uses TYPESOF to determine what definitions exist for *TOP* and, if there is more than one, asks you, via a menu, which one to use. If *TOP* is defined and is a non-list, calls INSPECT on that value. Edit also has a variety of subcommands which allow choice of editor (DEdit, TTYEdit, etc.) and whether to invoke that editor on the definition of *TOP* or the form itself.

Note: DEdit caches each subordinate edit window in the window from which it was entered for as long as the higher window is active. Thus, multiple DEdit commands

do not incur the cost of repeatedly allocating a new window.

EDITCOM

[DEdit Command]

Allows you to run arbitrary EDIT commands on the structure being DEdited (there are far too many of these for them all to appear on the main menu). *TOP* should be an EDIT command, which will be applied to *NXT* as the current edit expression. On return to DEdit, the (possibly changed) current EDIT expression will be selected as the new *TOP*. Thus, selecting some expression, typing (R FOO BAZ), and selecting EDITCOM will cause FOO to be replaced with BAZ in the expression selected.

In addition, a variety of common EDIT commands are available as subcommands of EDITCOM. Currently, these include ? = , GETD, CL, DW, REPACK, CAP, LOWER, and RAISE.

BREAK

[DEdit Command]

Does a BREAKIN AROUND the current expression *TOP*. (See BREAKIN function in *IRM*).

EVAL

[DEdit Command]

Evaluates *TOP*, whose value is pushed onto the stack in place of *TOP*, and which will therefore appear, selected, in the edit buffer.

EXIT

[DEdit Command]

Exits from DEdit (equivalent to Edit OK; see "Commands For Leaving The Editor," below).

OK

[DEdit Command]

STOP

[DEdit Command]

Subcommands of EXIT. OK exits without an error; STOP exits with an error. Equivalent to the EDIT commands with the same names.

DEdit Parameters

There are several global variables that can be used to affect various aspects of DEdit's operation.

EDITEMBEDTOKEN

[Variable]

Initially &. Used in both DEdit and the teletype editor to indicate the special atom used as the embed token.

DEDITLINGER

[Variable]

Initially T. The default behavior of the topmost DEdit window is to remain active on the screen when exited. This is occasionally inconvenient for programs that call DEdit directly, so it can be made to close automatically when exited by setting this variable to NIL.

DEDITYPEINCOMS

[Variable]

Defines the control characters recognized as commands during DEdit type-in. The elements of this list are of the form (*LETTER COMMANDNAME FN*), where

LETTER is the alphabetic character corresponding to the control character desired (e.g., A for control-A),

COMMANDNAME is a symbol used both as a prompt and internal tag,

FN is a function applied to the expressions typed as arguments to the command.

See the current value of DEDITYPEINCOMS for examples. DEDITYPEINCOMS is only accessed when DEdit is initialized, so DEdit should be reinitialized with RESETDEDIT (see "Calling DEdit," above) if it is changed.

DT.EDITMACROS

[Variable]

Defines the behavior of the EDIT command when invoked on a form that is not a list or symbol, thus telling DEdit how to edit instances of certain datatypes. DT.EDITMACROS is an association list keyed by datatype name; entries are of the form

(*DATATYPE MAKESOURCEFN INSTALLEDITFN*).

When told to edit an object of type *DATATYPE*, DEdit calls *MAKESOURCEFN* with the object as its argument.

MAKESOURCEFN can either do the editing itself, in which case it returns NIL, or else it destructures the object into an editable list and returns that list.

In the latter case, DEdit is then invoked recursively on the list; when that edit is finished, DEdit calls *INSTALLEDITFN* with two arguments, the original object and the edited list. If *INSTALLEDITFN* causes an error, the recursive DEdit is invoked again, and the process repeats until the you either exit the lower editor with STOP, or exit with an expression that *INSTALLEDITFN* accepts.

For example, suppose the you have a datatype declared by (DATATYPE FOO (NAME AGE SEX)). To make sure that instances of FOO can be edited, an entry (FOO DESTRUCTUREFOO INSTALLFOO) is added to DT.EDITMACROS, where the functions are defined by

```
(DESTRUCTUREFOO (OBJECT)
  (LIST (fetch NAME of OBJECT)
        (fetch AGE of OBJECT)
        (fetch SEX of OBJECT)))
(INSTALLFOO (OBJECT CONTENTS)
  (if (EQLLENGTH CONTENTS 3)
    then (replace NAME of OBJECT with (CAR CONTENTS))
         (replace AGE of OBJECT with (CADR CONTENTS))
         (replace SEX of OBJECT with (CADDR CONTENTS))
    else (ERROR "Wrong number of fields for FOO" CONTENTS)))
```

User Interface — Advanced Features

Multiple DEdit Commands

It is occasionally useful to be able to give several commands at once — either because you think of them as a unit or because the intervening re-pretty-printing is distracting. The stack architecture of DEdit makes such multiple commands easy to construct. You just push whatever arguments are required for the complete suite of commands you have in mind. Multiple commands are specified by holding down the CONTROL key during command selection. As long as the control key is down, commands selected will not be executed, but merely saved on a list. Finally, when a command is selected without the control key down, the command sequence is terminated with that command being the last one in the sequence.

You would rarely construct long sequences of commands in this fashion, because the feedback of being able to inspect the intermediate results is usually worthwhile. Typically, just two or three step idioms are composed in this fashion.

DEdit Idioms

As with any interactive system, there are certain common idioms on which experienced users depend heavily. In the case of DEdit, many of these idioms concern easy ways to achieve the effects of specific commands from the Edit system, with which many users are already familiar. The DEdit idioms described below are the result of the experience of the early users of the system and are by no means exhaustive. In addition to those that each user will develop to fit his own particular style, there are many more to be discovered and you are encouraged to share your discoveries.

Because of the novel argument specification technique (postfix; target first) many of the DEdit idioms are very simple, but opaque until you have absorbed the "target-source-command" way of looking at the world. Thus, you select where type-in is to go before touching the keyboard. After typing, the target will be selected second and the type-in selected on top, so that an AFTER, BEFORE or REPLACE will have the desired effect. If the order is switched, the command will try to change the type-in (which may or may not succeed), or will require tiresome swapping or reselection. Although this discipline seems strange at first, it comes easily with practice.

Segment selection and manipulation are handled in DEdit by first making them into a sublist, so they can be handled in the usual way. Thus, if you want to remove the three elements between A and E in the list (A B C D E), you select B, then D (either order), then make them into a sublist with the () command. This will leave the sublist (B C D) selected, so a subsequent DELETE will remove it. This can be issued as a single "(); DELETE" command using multiple command selection as described above, in which case the intermediate state of (A (B C D) E) will not show on the screen.

Inserting a segment proceeds in a similar fashion. Once the location of the insertion is selected, the segment to be inserted is typed as a list (if it is a list of atoms, they can be typed without parentheses and the READ will make them into a list, as you would expect). Then, the command sequence "AFTER (or BEFORE or REPLACE); () OUT" (given either as a multiple command or as two separate commands) will insert the type-in and splice it in by removing its parentheses.

Moving an expression to another place in the structure being edited is easily accomplished by a DELETE followed by an INSERT. Select the location where the moved expression is to go to; select the expression to be moved; then give the command sequence "DELETE; AFTER (or BEFORE or REPLACE)". The expression will first be deleted into the edit buffer where it will remain selected. The subsequent insertion will insert it back into the structure at the selected location.

Embedding and extracting are done with the REPLACE command. Extraction is simply a special case of replacing something with a subpiece of itself:

Select the thing to be replaced.

Select the subpart that is to replace it.

REPLACE.

Embedding also uses Replace, in conjunction with the embed token (the value of EDITEMBEDTOKEN, initially the single character atom &). Thus, to embed some expression in a PROG,

Select the expression.

Type: (PROG VARSLST &)

REPLACE.

SWITCH can also be used to generate a whole variety of complex moves and embeds.

For example, switching an expression with type-in not only replaces that expression with the type-in, but provides a copy of the expression in the buffer, from where it can be edited or moved to somewhere else.

Finally, you can exploit the stack structure on selections to queue multiple arguments for a sequence of commands. Thus, to replace several expressions by one common replacement, select each of the expressions to be replaced (any number), then the replacing expression. Now hit the REPLACE command as many times as there are replacements to be done. Each REPLACE will pop one selection off the stack, leaving the most recently replaced expression selected. As the latter is now a copy of the original source, the next REPLACE will have the desired effect, and so on.

Limitations

DEdit is not error-protected. If you select the up-arrow to close a break window which resulted from using the EVAL command, the DEdit window is also closed.