Editing and Mail

Zmacs

Introduction

Zmacs, the Genera editor, is built on a large and powerful system of text-manipulation functions and data structures, called *Zwei*.

Zwei is not an editor itself, but rather a system on which other text editors are implemented. For example, in addition to Zmacs, the Zmail mail reading system also uses Zwei functions to allow editing of a mail message as it is being composed or after it has been received. The subsystems that are established upon Zwei are:

- Zmacs, the editor that manipulates text in files
- Dired, the editor that manipulates directories represented as text in files
- Zmail, the editor that manipulates text in mailboxes
- Converse, the editor that manipulates text in messages

Since these parts of the system are all based on Zwei, many of the commands available as Zmacs commands are available in other editing contexts as well. (In addition, many of the same editing commands are available in the Input Editor, which you use when typing commands or forms to other programs, such as the Lisp Listener. The Input Editor is not based on Zwei, however.)

Zmacs is used not only to create text for documents and programs, but also to compile programs, check them for correct structure, inspect parts of programs (including system programs), create commands, alphabetize lists, check spelling, and perform many other functions.

Scope

Zmacs is intended as a reference for all users of Zmacs in Genera. It contains both conceptual overview and reference material that together describe the Zmacs editor. We assume that you have already read the *Genera User's Guide*.

Organization

The first three chapters contain introductory material for users who are unfamiliar with Zmacs concepts. Experienced users can skim the remaining chapters, which are organized according to editing function, and use them as reference material.

"Introduction to the Zmacs Manual" gives an overview of Zmacs and describes Zmacs documentation conventions in this manual.

"Getting Started in Zmacs" introduces basic Zmacs concepts and commands.

"Getting Help in Zmacs" describes ways to get out of trouble and how to get Zmacs information during editing.

"Moving the Cursor in Zmacs" includes descriptions of both mouse and keyboard motion commands.

"Basic Text Editing in Zmacs" explains how to edit text (for example, how to insert and delete, how to search and replace, and how to use character styles). This section also explains how to use word abbreviations for Zmacs editing commands.

"Formatting Text in Zmacs" explains how to format text using Zmacs commands and environments.

"Working with Regions in Zmacs" tells how to manipulate blocks of text.

"Manipulating Buffers and Files in Zmacs" gives more information on manipulating blocks of text, inserting files, keeping track of everything, and editing your directory.

"Zwei Undo Facility" explains how to selectively undo any or all of the changes you have made in an editor buffer.

"Zmacs Speller" describes Zmacs spelling tools.

"Editing Lisp Programs in Zmacs" describes the ways in which Zmacs is tailored for use in writing and editing programs in Lisp.

"Customizing the Zmacs Environment" describes how to fine tune your Zmacs environment using major and minor modes to set it up, keyboard macros to perform special editing tasks, binding keys to the commands of your choice, setting Zmacs variables to alter your standard system defaults, and saving the customized environment.

Zmacs Manual Notation Conventions

The word *current*, when describing a word, line, paragraph, page, or any Zmacs-recognizable piece of text, refers to the text that currently contains (or immediately follows) the cursor.

The *invocation* of a command shows exactly what keys you must press to invoke, or call, a command. We use the following format to describe Zmacs commands:

```
invocation                                                    Name
alternate invocation
alternate invocation
```

Formal description of command

Since each extended (m-X) command contains its name as part of its invocation, we do not repeat the name again on that line.

Example 1 of Zmacs Notation Conventions

m-> Goto End

Moves point to the end of the buffer.

With a numeric argument *n* between 0 and 10, moves point to a place *n*/10 of the way from the end of the buffer to the beginning.

(The m-> command goes to the end of the buffer — its name is Goto End.)

Example 2 of Zmacs Notation Conventions

Dired (m-X)

Prompts for the name of a directory to edit with Dired.

(The Dired (m-X) command is an extended command that enters the directory editor.)

Example 3 of Zmacs Notation Conventions

m-M Back To Indentation
c-m-M
m-RETURN
c-m-RETURN

Positions point before the first nonblank character on the current line.

(Back to Indentation has several possible invocations that all move back to the first nonblank character on the line.)

Getting Started

Organization of the Screen

Zmacs divides its window into several areas: the editor window, the echo area, and the mode line, each of which contains its own type of information.

Zmacs Editor Window

The biggest area, the *editor window*, shows the text you are editing. You can edit several different items at once with Zmacs; each item is edited in a separate editing environment called a *buffer*.

Editor Window's Buffer

Zmacs gives every buffer a name. At any time you are editing only one of them, the *selected* buffer. When we speak of what some command does to "the buffer", we are talking about the currently selected buffer. Multiple buffers in Zmacs make it

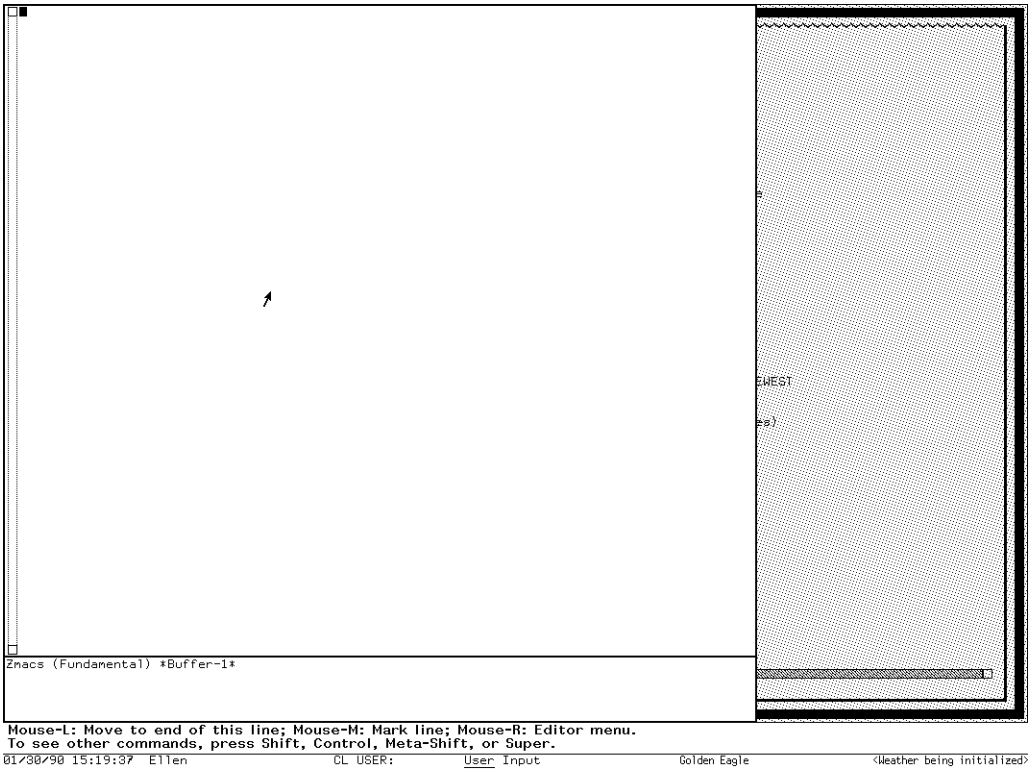


Figure 86. The initial Zmacs window

easy to switch among several files; the mode line tells you which one you are editing.

Editor Window's Cursor and Point

The small blinking rectangle, the *cursor*, usually appears somewhere within the buffer, showing the position of *point*, the location at which editing takes place. Although the cursor covers a single character, we consider point to be at the left edge of the cursor, between the character the cursor is blinking on and the previous character.

Editor Window's Typeout

When you request some other information from Zmacs (for example, if you ask for help or a listing of a file directory), Zmacs needs room to display this type of information. It prints this *typeout* in a *typeout window* (at the top of the editor window), which temporarily overlays the text in the editor window, using as much room as it needs.

Since the typeout is not part of the file you are editing, Zmacs delineates it from the editor buffer by drawing a line across the window between them (if both are present). The typeout window goes away if you type any command; if you want to make it go away immediately but not do anything else, you can press SPACE. The cursor, which appears at the end of the typeout, then moves back to its original location in the buffer.

Zmacs Echo Area

A few lines at the bottom of the screen make up what is called the *echo area*. *Echoing* means displaying the commands that you type. Zmacs commands are usually not echoed at all, but if you pause in the middle of a multicharacter command, all the characters (including numeric arguments and prefixes) typed so far are echoed. This is intended to prompt you for the rest of the command. The rest of the command is echoed, too, as you type it. This behavior is designed to give confident users optimum response, while giving hesitant users maximum feedback.

Echo Area's Minibuffer

Many Zmacs commands prompt you for additional information. This prompting happens in a small window within the echo area called the *minibuffer*.

When Zmacs prompts you, the cursor in the main editing window stops blinking and a blinking cursor appears in the minibuffer. Over the minibuffer, in the Zmacs mode line, some prompting text appears, indicating what information Zmacs is prompting you for.

When you type a response to the prompt, that response is inserted in the minibuffer. You can edit text in the minibuffer using the same Zmacs commands used in the main Zmacs window.

When you are done typing (and possibly editing) a response to the prompt, the RETURN key finishes your response.

Zmacs Mode Line

The line above the echo area is known as the *mode line*. It is the line that usually starts with ZMACS (Fundamental). Its purpose is to display information about the current buffer.

The mode line consists of:

- The name of the major mode
- The name of the minor mode(s), if any
- The name of the buffer

- The version number of the file
- The status of the buffer
- A message telling whether the buffer contents extend above and/or below the screen

The mode line has this format:

```
ZMACS (major-mode minor-mode(s)) buffer (version) buffer-status [position-flag]
```

Mode Line's *Major-mode*

major-mode is always the name of the *major mode* you are in. At any time, Zmacs is in one and only one of its possible major modes. The major modes available include:

- Fundamental mode (which Zmacs starts out in)
- Text mode
- Lisp mode
- MACSYMA mode

For full details about all the major modes, how they differ, and how to select one, see the section "Zmacs Major Modes".

Mode Line's *Minor-mode*

minor-mode is a list of the *minor modes* that are turned on at the moment. For example:

Fill	Auto Fill Mode
Electric Shift-lock	Electric Shift Lock Mode
Abbrev	Word Abbrev Mode
Overwrite	Overwrite Mode

For more information, see the section "Built-in Customization Using Zmacs Minor Modes".

Mode Line's *Buffer*

buffer is the name of the workspace that holds the text you are editing. A buffer can be named in one of two ways:

- By Zmacs, with a name that corresponds to the existing file that it contains or with its standard name for an empty buffer
- By you, with any name you like

When a buffer contains a file, the buffer name is the pathname of that file, rearranged with the file name first and the host and directory at the end. For a description of pathname components, see the section "Pathnames". When a buffer does not contain a file, the buffer name is a string.

Buffers that do not contain files are empty, newly created, or temporary buffers. When Zmacs creates and names a buffer, that name begins and ends with an asterisk. When you create and name a buffer, on the other hand, its name is of your choosing.

When you first start up and enter Zmacs, your buffer is either:

- An empty buffer called `*Buffer-1*`, which is the only one that exists when Zmacs starts up
- A buffer containing an existing file, which Zmacs appropriately calls by its name

For information on multiple buffers, see the section "Manipulating Buffers and Files in Zmacs".

Mode Line's *Version*

(version) is the version number most recently visited or saved. The mode line does not display any version number if the file is on a file system that does not support version numbers, such as UNIX.

Mode Line's *Buffer-status*

If the mode line displays *, changes have been made in the buffer that have not been saved in the file. If the buffer has not been changed since it was read in or saved, the mode line does not display a asterisk.

Mode Line's *Position-flag*

When the mode line displays the message `[More above]`, your screen shows the end of your buffer contents; when the mode line shows `[More below]`, your screen shows the beginning of your buffer contents. When it says `[More above and below]`, the buffer contents extend above and below the part that the screen displays. When the display shows the entire buffer contents, this message does not appear at all.

Mode Line Example

```
ZMACS (Text) text.text /dess/doc/books/ VAX: * [More above and below]
```

In this sample mode line, we are in Zmacs Text Mode, editing a file named text.text, which resides in the directory /dess/doc/books on the host named VAX. The file has been changed since we last saved it (indicated by the *), and the file contents extend above and below the portion that Zmacs displays on the screen.

Using Zmacs Help

Zmacs has many features that provide information about Zmacs commands, existing code, buffers, and files. Two features are generally useful: the HELP key and completion. See the section "Getting Help in Zmacs" and see the section "Using Zmacs Commands".

Pressing the HELP key in a Zmacs editing window gives information about Zmacs commands and variables. For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings". The kind of information it displays depends on the key you press after HELP.

HELP ?	Displays a summary of HELP options.
HELP A	Displays names, key bindings, and brief descriptions of commands if their names or the first lines of their help documentation contain a string you specify. The A refers to m-X Apropos, the command equivalent. If you enter the command with a numeric argument, only the names of the commands are searched for the string, and not the help documentation.
HELP C	Displays the name and description of a command bound to a key you specify.
HELP D	Displays documentation for a command whose name you specify.
HELP L	Displays a listing of the last 60 keys you pressed.
HELP U	Offers to undo the last major Zmacs operation, such as sorting or filling, when possible.
HELP V	Displays the names and values of Zmacs variables whose names contain a string you specify. For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".
HELP W	Displays the key binding for a command you specify. (The W refers to where.)
HELP SPACE	Repeats the last HELP command.

Entering Zmacs

You can enter, or invoke, the editor in several ways

- Press SELECT E.
- Select [Editor] from the System menu using the mouse.
- Use the Edit File CP command to enter Zmacs, specifying a particular file.
- Use the Select Activity command, specifying either Zmacs or Editor as its argument.

You can also enter Zmacs from a Lisp listener using the Lisp functions **ed** or **zwei:edit-functions**.

To run multiple copies of Zmacs. Press SELECT c-E to create a new editor process. Cycle from one editor process to the next by repeatedly pressing SELECT E.

Entering Zmacs with SELECT E

You can invoke the editor by pressing the SELECT key and then the letter E.

- If you have already been in the editor since booting the machine, Zmacs returns you to the same place in the same buffer that you last used.
- If this is the first time you are entering Zmacs since booting the machine, Zmacs puts you in an empty buffer named *Buffer-1*.

SELECT E enters or returns you to the editor from anyplace in the system, not just when you are talking to Lisp.

You can create multiple copies of Zmacs by pressing SELECT c-E. SELECT E returns you to the last copy of the Zmacs process you used. Repeatedly pressing SELECT E cycles through all the copies of Zmacs.

For information on other methods of invoking Zmacs, see the section "Entering Zmacs".

Entering Zmacs with the System Menu

You can invoke the editor using the System menu.

Summon a System menu by clicking sh-Right. Then click Left on the [Editor] option, which puts you into a Zmacs buffer. If you are returning to the editor Zmacs puts you back at the same place in the same buffer, and if you are entering Zmacs for the first time it puts you in an empty buffer.

Entering Zmacs with ed

The Lisp function **ed** enters Zmacs from a Lisp Listener. See the function **ed**.

When reentering Zmacs within a login session, **ed** enters the editor, preserving its state as it was when you left. When entering Zmacs for the first time during a login session, **ed** initializes Zmacs and creates an empty buffer.

You can enter **ed** with arguments with these values:

t	The **ed** function enters the editor, creates an empty buffer, and selects it.
Pathname or string	The **ed** function enters the editor and finds or creates a buffer with the specified file in it.
Defined symbol	The editor tries to find the source definition of that symbol for you to edit. A defined symbol can be, for example, a function, macro, variable, flavor, or system.
The symbol **zwei:reload**	The system reinitializes the editor. This destroys all existing buffers, so use this only if you have to.

Entering Zmacs with zwei:edit-functions

The Lisp function **zwei:edit-functions** also enters Zmacs from a Lisp Listener.

zwei:edit-functions *functions* *Function*

zwei:edit-functions is like **ed** in that inside the editor process it throws you back into the editor, whereas from another process it just sends a message to the editor and selects the editor's window. **zwei:edit-functions** gives *functions* to the editor in the same way that Edit Callers and similar editor commands would. See the section "The Zmacs Edit Callers Commands".

This command is useful when you have collected the names of things that you need to change, for example, using some program to generate the list. *spec-list* is a list of definitions; these are either function specs (if the definitions are functions) or symbols.

Zmacs sorts the list into an appropriate order, putting definitions from the same file together, and creates a support buffer called `*Function-Specs-to-Edit-`*n*`*`. It selects the editor buffer containing the first definition in the list.

Using Zmacs Commands

Commands

Zmacs *commands* are implemented by Lisp functions that perform the editing work. Every Zmacs command has a *name*, and many commands are bound to keys.

There are, in effect, three kinds of Zmacs commands, based roughly on how commonly used they are and how "serious" they are in their effects.

The first kind of Zmacs commands are the *keyboard accelerators*. Commonly used commands, such as Forward Word and Delete Forward, are bound to keys. Forward

Word is on m-F and Delete Forward is on c-D. It would be tiresome to have to type a command each time you wanted to move forward a word or delete the next character. These commands also take *numeric arguments*. If you want to move forward three words, press m-3 m-F and if you want to delete the next fourteen characters, press c-14 c-D.

The second kind of Zmacs commands are the c-X commands. These commands take *two* keystrokes to invoke them, such a c-X c-S to save a file buffer, or c-X I to insert a file into the buffer, or c-X RUBOUT to kill the previous sentence. These commands also take numeric arguments where appropriate. You can see the entire list of c-X commands by pressing HELP C and then c-X followed by *. (There is also an interesting set of c-Q commands. You can see that list by pressing HELP C and then c-Q.)

Finally, there are the *extended* commands, commonly called the m-X (meta-x) commands. These commands are invoked by pressing m-X and then typing the command name and entering it. In general, these commands are more rarely used or have more long-lasting effects and are therefore slightly less easy to enter. Examples of these commands include Show Character Styles or Add Patch. In general, numeric arguments to these commands cause some unusual behavior, such as sending command output to a printer.

Command tables assign keystrokes and names to commands. Each time you press a key, Zmacs looks up the function associated with that key. For ordinary characters, the function **com-standard**, in the standard command table, inserts the character once.

Zmacs Command Completion

Some Zmacs operations require you to provide names — for example, names of extended commands, Lisp objects, buffers, or files. Often you do not have to type all the characters of a name; Zmacs offers *completion* over some names. When completion is available, the word Completion appears in parentheses above the right side of the minibuffer.

You can request completion when you have typed enough characters to specify a unique word or name. For extended commands and most other names, completion works on initial substrings of each word. For example, m-X c SPACE b is sufficient to specify the extended command Compile Buffer. SPACE, COMPLETE, RETURN, and END complete names in different ways. Press HELP or click Right on the editor window or minibuffer to display a mouse-sensitive list of possible completions for the characters you have typed.

In addition, c-/ displays a mouse-sensitive list of every command that *contains* the substring and c-? displays a mouse-sensitive list of every command that *starts* with that string.

SPACE	Completes words up to the current word.
HELP or c-?	Displays possible completions in the typeout area.
Click Right	Pops up a menu of possible completions.

c-/	Displays a mouse-sensitive list of all commands *containing* the string you have typed so far.
c-?	Displays a mouse-sensitive list of all commands *starting with* the string you have typed so far.
COMPLETE	Completes as much as possible. This could be the full name.
RETURN or END	Confirms the name if possible, whether or not you have seen the full name.

Keystrokes

In general, commands that begin with a CONTROL (c-) key modifier operate on single characters, commands that begin with a META (m-) key modifier operate on words, sentences, paragraphs, and regions, and commands that begin with a CONTROL META (c-m-) modifier operate on Lisp code.

Prefix character commands consist of more than one keystroke per command. For example, to invoke the command c-X F, you first type the prefix character c-X and then the primary key F. Prefix character commands are not case-sensitive — that is, Zmacs converts a lowercase character following a prefix character command (like c-X) to uppercase. For example, c-X f is equivalent to c-X F.

Zmacs commands are self-delimiting. Unless otherwise specified, you do not need to type a carriage return or other terminating character to finish typing a command.

Extended Commands

Extended commands extend the range of commands past the one-or-two-keystroke limitation. You invoke Zmacs extended commands by name using the m-X command:

m-X Extended Command

Prompts for the name of a Zmacs command and executes that command.

Command completion is provided. See the section "Completion for Extended Commands (m-X Commands)".

Command Tables

There is always a currently active command table (*comtab*). When you invoke a command, Zmacs looks it up in the associated command table, checks to see if it is valid in the current context, and performs the function. Zmacs uses many comtabs, including the standard comtab, a comtab for commands that begin with the c-X prefix, and a comtab for reading pathnames in the minibuffer.

Many commands have no meaning outside their own limited context. Sometimes you might get a message or see online documentation about a command that says Not available in current context. Those commands that are not accessible via a

keystroke and not accessible via m-X are likely to be commands that do not work in the current context. For example, a command that is part of Dired is available on a key only when you are in Dired.

You can invoke a command that is not available in the current comtab with the c-m-X command. c-m-X works like m-X: you press the keys and then type the command name in the minibuffer. This is primarily intended for debugging new editor commands that have not yet been installed on any key. Using c-m-X to invoke a command that is not in the current comtab because it works only in some other context is a sure way to get into trouble.

c-m-X Any Extended Command

Prompts for the name of a Zmacs command and executes that command.

Command completion is provided.

Numeric Arguments

Many Zmacs commands take numeric arguments, which you type before the main command keystroke. Specify a numeric argument by pressing any combination of any of the modifier keys (c-, m-, s-, or h-) with the number. This way, you can type sequences of commands more easily without frequently alternating keys.

Numeric arguments to commands appear in the echo area when you do not type the command immediately. With no delay, the argument does not appear.

In general, use negative arguments to tell a command to move or act backwards. You can specify a negative argument by pressing any modifier key with the minus sign followed by the number. Most commands treat a numeric argument consisting of just a minus sign the same as -1.

For more details, see the section "Numeric Arguments and the Motion Commands".

Example of Numeric Arguments

c-F is the command to move the cursor forward one character. c-3 c-5 c-F moves point forward 35 characters; c-- c-3 c-5 c-F moves point backward 35 characters.

Throughout this manual, instead of writing out c-4 c-5 c-F or m-4 m-5 m-B, we usually abbreviate to c-45F or m-45B.

Defaults to Numeric Arguments

Many commands have default numeric arguments. This means that in the absence of a numeric argument, the command behaves as if the default argument were given. Most commands have a default argument of 1. This includes all the commands that interpret numeric arguments as repeat counts. Some commands have a different default and still others have no default: their behavior in the absence of a numeric argument is different from their behavior with a numeric argument.

c-U Quadruple Numeric Arg

This special command prefixes other commands, usually representing a numeric argument of 4. You can repeat c-U; it multiplies the numeric argument by 4 each time. For example, c-U c-U c-F moves point forward 16 characters. Sometimes instead of representing a numeric argument of 4, c-U alters the action of a command slightly; for example, when used with the command Set Pop Mark, c-U takes different actions with the mark. (For a description of the Set Pop Mark command, see the section "Working with Regions in Zmacs".)

Executing CP Commands From Zmacs

If you wish to execute a CP command while editing, you can press SUSPEND and issue the command in the typeout window, you can press SELECT L and issue the command to a Lisp Listener, or you can issue the m-X Execute CP Command command.

m-X Execute CP Command — Reads a Command Processor command line from the minibuffer and executes that command. All output from the command appears in the Zmacs typeout window.

Leaving Zmacs

Use a system-wide command to switch programs, such as SELECT, FUNCTION S, the System menu, or, if you have multiple windows on the screen, position the mouse on another window and click.

Leaving Zmacs with the SELECT Key

A set of windows is always available by pressing the SELECT key and then one of the following keys:

Key	*Program*
C	Converse, for messages to other users
D	Document Examiner, for reading online documentation
F	File system editor for access to files and directories
I	Inspector, for inspecting and modifying data structures
L	Lisp
M	Mail reading and sending system
N	Notifications, for rereading system notifications
P	Peek, a system status display
T	Telnet, a virtual terminal utility for logging in to other hosts
X	Flavor Examiner, for examining the structure of flavors that are defined in the Lisp environment

Leaving Zmacs via the System Menu

The System menu is a momentary menu that lists several choices for acting upon windows and calling programs (for example, a Lisp Listener, Zmacs, or the Inspector). You can always call the System menu by holding down the SHIFT key and clicking Right once). Use the System menu to do many things, among them:

- Create new windows.
- Select old windows.
- Change the size and placement of windows on the screen.
- Hardcopy a file.

Leaving Zmacs with c-Z

The Zmacs command c-Z returns you to the window in which the **ed** function was most recently called, usually the Lisp Listener.

Getting Help

Getting Out of Trouble

Sometimes you type the wrong command. Mostly it is obvious what you have done wrong, and it is a simple matter to undo it. There are, however, some kinds of trouble you can get into that require special remedies. For example, you might accidentally delete large chunks of text you need or you might begin to type a command and then change your mind.

This section tells you how to recover from these situations.

Undoing

The Zwei Undo facility remembers all the changes that you have made in an editor buffer and allows you to selectively undo any or all of the changes you have made. The Undo facility is available from Zmacs, Converse, the Zmail draft editor, and other editor buffers based on the Zwei substrate. (It is not available from the Input Editor or in the minibuffer.)

The simplest operation of the Undo facility is to undo the most recent change to the editor buffer. Go to a buffer, type something in, delete it, and then press c-sh-U. The deletion is undone. Region marking shows what was undone. Now press c-sh-R. You're back where you started. It is always safe to undo, because you can always redo, and vice versa.

The Undo (m-X) and Redo (m-X) commands are similar to c-sh-U and c-sh-R with the added feature that a display in the minibuffer shows you what will be undone or redone before any action is taken. HELP U also displays the change before undoing it.

Keep pressing c-sh-U. Previous changes to the buffer are undone. You can keep doing this until the buffer is returned to its original state. When you reach this point, if the buffer contains a file, it's no longer marked as needing to be saved. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing to be compiled.

Repeated pressing of c-sh-R will successively restore the buffer until all the undo commands have been cancelled out.

If you read in a file with no intention of changing it and accidentally type some characters into it, use c-sh-U rather than RUBOUT to get rid of them. That way, the buffer is no longer considered to be modified.

Undo commands operate only on the current buffer. Each buffer has an undo history, and a separate redo history. The undo history can be displayed with c-0 c-sh-U. Likewise, the redo history can be displayed with c-0 c-sh-R. Items in the history are mouse-sensitive. You can undo or redo all changes *up through a given change* or you can undo or redo *any single change* in the history. By default, both histories are discarded when you save the buffer. See the section "Discard Change History".

Of course, subsequent changes may depend on the single change that you are undoing or redoing, so no guarantee can be made that undoing change number 13 in a 29-change history will have no effect on changes 14 through 29. (On the other hand, you can always back out of any undo or redo.)

This sounds more complicated in writing than it is when you are doing it. A few minutes experimentation in an editor buffer will make you a competent and confident user of the most important and common undoing and redoing operations.

After an undo or redo, the text that was modified is highlighted the same as if you had marked a region, but in this case there is no region, and the highlighting disappears when you type the next command. The history also shows you what constitutes each change. See the section "What is a Change to the Undo Facility?".

Large Deletions

Do not delete large pieces of text by repeatedly pressing RUBOUT and c-D. Apart from being slow, text deleted character-by-character is gone for good.

Instead, use delete and kill commands that save deleted regions in the kill history. c-K, m-K, and the commands that deal with *regions* easily wipe out and save larger chunks. Also, RUBOUT or c-D with a numeric argument erases that many characters all at once and saves them in the kill history. For full descriptions of these delete and kill commands, see the section "Deleting and Transposing Text in Zmacs".

Getting Text Back

The system has different histories for different contexts. One of these is always the *current history*. The two histories that you need to use for yanking in Zmacs are the *kill history* and the *command history*. The kill history remembers pieces of

text that you killed or copied into it. In the context of Zmacs, the command history remembers all the editor commands that use the minibuffer in any way.

Additions to the histories are placed at the top of the list, so that history elements are stored in reverse chronological order — the newer elements at the top of the history, the older elements toward the bottom. A history remembers everything that has been typed to it since the last cold boot — it has no size limit.

Yanking commands pull in the elements of the history. *Top-level commands* start a yanking sequence; for example, c-Y yanks back the last text killed from the kill history, and c-m-Y yanks back the last command performed in the minibuffer. m-Y performs all subsequent yanks in the same sequence; for example, pressing m-Y while the kill history is the current history yanks the next item from that history.

A yanking sequence ends when you type new text, execute a form or command, or start another yanking sequence.

For complete descriptions of killing and yanking, see the section "Working with Regions in Zmacs".

Finding Out About Zmacs Commands

Sometimes you want to know if a Zmacs command exists that performs a certain function. Or, you might think that you know what a certain keystroke does, but you still want to make sure, or refresh your memory about its exact usage. This manual is one resource you might use in these circumstances. Zmacs itself has a number of built-in self-documentation facilities. This section describes some ways to get at this documentation.

Finding Out About Zmacs Commands with HELP

The HELP key is a prefix to a useful group of commands giving various kinds of online help. If you forget what a command does, which keystrokes perform an action, or have no idea how to accomplish something, press HELP.

Whenever you have a question of any kind, press HELP. Zmacs prompts you in the minibuffer for details on what kind of help. If you don't know, press HELP again and it tells you, in the *typeout window*, how to find what you're looking for. The typeout window displays right over the editor window. The actual contents of the buffer are not affected, and the next command you type restores the buffer display.

Pressing the HELP key in a Zmacs editing window gives information about Zmacs commands and variables. For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings". The kind of information it displays depends on the key you press after HELP.

HELP ?	Displays a summary of HELP options.
HELP A	Displays names, key bindings, and brief descriptions of commands if their names or the first lines of their help documentation contain a string you specify. The A refers to m-X Apro-

pos, the command equivalent. If you enter the command with a numeric argument, only the names of the commands are searched for the string, and not the help documentation.

HELP C	Displays the name and description of a command bound to a key you specify.
HELP D	Displays documentation for a command whose name you specify.
HELP L	Displays a listing of the last 60 keys you pressed.
HELP U	Offers to undo the last major Zmacs operation, such as sorting or filling, when possible.
HELP V	Displays the names and values of Zmacs variables whose names contain a string you specify. For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".
HELP W	Displays the key binding for a command you specify. (The W refers to where.)
HELP SPACE	Repeats the last HELP command.

Finding Out What a Zmacs Command Does

HELP C

The command HELP C displays "Document Command:" below the mode line and waits for you to type a command. When you do, Zmacs displays the internal documentation for that command.

If you press HELP C followed by c-F, the response is:

```
c-F is Forward, implemented by ZWEI:COM-FORWARD:
Moves forward one character.
With a numeric argument (n), it moves forward n characters.
```

The first line above tells you the name of the command (in this case Forward), and the name of the internal Lisp function that actually does the work (in this case **zwei:com-forward**). (You don't need to know these internal names for basic editing.) The COM-xxx name displayed by HELP C is mouse-sensitive: clicking Left on it edits the COM-xxx function, and clicking Right displays a menu with choices that include Arglist, Edit Definition, Disassemble, and Show Documentation.

The next line is a very short description of what the command does; it usually tells you what the command does without a numeric argument and how a numeric argument modifies that behavior.

Finding Out What a Prefix Command Does

When you ask (with HELP C) for documentation on a prefix command like c-X, Zmacs prompts you, in the typeout window, to complete the command. Zmacs displays the documentation for the prefix command in the typeout window.

Finding Out What an Extended Command Does

HELP D

When you want to find out what an extended command does, you can display the documentation for the command by pressing HELP D, which prompts in the minibuffer "Describe command:", to which you type the command's name.

Searching for Appropriate Zmacs Commands

HELP A

m-X Apropos

When you can only guess at part of the name or function of a command by the action it performs, there is a command, HELP A, to help you scan information about all the available Zmacs commands to find the one you want. All you have to do is type in a string, such as "buffer", and all command names plus the first line of all help documentation are scanned for the string you specify.

Each Zmacs command has a name. The name is almost always exactly what you would expect; that is, the name describes the function of the command in reasonably plain English. If not, the word you're looking for is almost surely in the first line of the help documentation.

With a numeric argument, HELP A searches only the command names.

The A stands for apropos. The m-X Apropos command works the same way.

Example of a Search String for HELP A

The command you perform when you use m-Q is called "Fill Paragraph", so you might expect a command that counts the number of paragraphs in the buffer to be called something like "Count Paragraphs" or "Paragraphs Count". No matter what, the word *paragraph* is going to be in the name or the first line of the help documentation.

Type HELP A and then paragraph. You'll see a list of help entries about paragraphs. Type c-U HELP A and then paragraph and you'll see a shorter list of help entries about commands that have "paragraph" as part of their names.

Finding Out What You Have Typed

HELP L

As you are editing you might find yourself in a confused state and not know how to recover.

If this happens, it is often useful to press HELP L to list the last 60 keystrokes you typed. By examining your own recent activity, it is often possible to find out where you went wrong and how to save work.

You should also consider the Undo Facility in these circumstances. See the section "Undoing".

More HELP Commands for Finding Out About Zmacs Commands

HELP U

Offers to undo the last "major" operation (such as fill or sort).

HELP V

Displays all the Zmacs variables whose names contain a certain substring. For descriptions of Zmacs variables. See the section "How to Specify Zmacs Variable Settings".

HELP W

Finds out whether an extended command is bound to a key.

Zmacs Help Command Summary List

This section lists the names of the available help commands grouped according to the context in which they are available. The purpose of this section is to summarize the capabilities and to help you determine both the overall contexts for which you can find help and a particular function that might be what you are looking for.

Zmacs Commands for Finding Out About the State of Buffers

Edit Changed Definitions (m-X)
Edit Changed Definitions Of Buffer (m-X)
List Buffers (c-X c-B)
List Changed Definitions (m-X)
List Changed Definitions Of Buffer (m-X)
List Definitions (m-X)
List Matching Lines (m-X)
Print Modifications (m-X)

Zmacs Commands for Finding Out About the State of Zmacs

Apropos (m-X)
Describe Variable (m-X)
Edit Zmacs Command (m-X)
HELP L

List Commands (m-X)
List Registers (m-X)
List Some Word Abbrevs (m-X)
List Tag Tables (m-X)
List Variables (m-X)
List Word Abbrevs (m-X)
Show Keyboard Macro (m-X)

Zmacs Commands for Finding Out About Lisp

Describe Variable At Point (c-sh-V)
Edit Callers (m-X)
Edit Callers In Package (m-X)
Edit Callers In System (m-X)
Edit CP Command (m-X)
Edit Definition (m-.)
Edit File Warnings (m-X)
Function Apropos (m-X)
List Callers (m-X)
List Matching Symbols (m-X)
Long Documentation (c-sh-D)
Multiple Edit Callers (m-X)
Multiple List Callers (m-X)
Quick Arglist (c-sh-A)
Show Documentation (m-sh-D)
Show Documentation Function (m-sh-A)
Show Documentation Variable (m-sh-V)
Where Is Symbol (m-X)

Zmacs Commands for Finding Out About Flavors

Edit Combined Methods (m-X)
Edit Methods (m-X)
List Combined Methods (m-X)
List Methods (m-X)
Show Documentation Flavor (m-sh-F)
Show Flavor Initializations (c-sh-F)

Zmacs Commands for Interacting with Lisp

Break (SUSPEND)
Compile And Exit (m-Z)
Compile Buffer (m-X)
Compile Changed Definitions (m-X)

Compile Changed Definitions Of Buffer (m-X), m-sh-C
Compile File (m-X)
Compile Region (m-X), c-sh-C
Compiler Warnings (m-X)
Edit Compiler Warnings (m-X)
Evaluate And Exit (c-m-Z)
Evaluate And Replace Into Buffer (m-X)
Evaluate Buffer (m-X)
Evaluate Changed Definitions (m-X)
Evaluate Changed Definitions Of Buffer (m-X), m-sh-E
Evaluate Into Buffer (m-X)
Evaluate Minibuffer (ESCAPE)
Evaluate Region (m-X), c-sh-E
Evaluate Region Hack (m-X)
Evaluate Region Verbose (c-m-sh-E)
Load Compiler Warnings (m-X)
Macro Expand Expression (m-X), c-sh-M
Quit (c-Z)
Trace (m-X)

General Information-giving Zmacs Commands

The following commands display:

- Information about the location of point
- Documentation about a specified Lisp function
- Argument list for the specified Lisp function
- Information about the current Lisp variable
- The number of lines in the region or page
- Possible parenthesis mismatches
- Trace information about the specified Lisp function

The word *current*, when describing a Lisp function or a Lisp variable, refers to (approximately) the function or variable whose name is nearest to the cursor.

c-X = Where Am I

Displays information about the location of point. It displays the X and Y positions, the octal code for the following character, the current line number and its percentage of the total file size. If there is a region, it displays the number of lines in it. Fast Where Am I (c-=) displays a subset of this information more quickly.

c-= Fast Where Am I

Quickly displays information about where point is. It displays the X and Y positions and the octal code for the following character. If there is a region, it displays the number of lines in it. Where Am I displays the same things and more.

m-sh-D Show Documentation

Show Documentation (m-X)

Displays the documentation for the given topic. It prompts for a topic name offering completion only on topics in the documentation database.

With a numeric argument, Show Documentation hardcopies the documentation on a printer of your choice.

This command is also available in Document Examiner. When you read documentation in the editor, a bookmark for that topic is inserted in the Background viewer in Document Examiner. See the section "Show Documentation Document Examiner Command".

See the section "Document Examiner".

Show Candidates Zmacs Command

Show Candidates (m-X)

Show Candidates prompts for a word or words to search for. By default it performs a heuristic or "smart" search for matching candidates.

With a numeric argument, Show Candidates prompts for a word or word to search for and then asks you to specify the style of matching. Your choices are:

- *Heuristic* matching, which is the default. This uses the words you have supplied as *stems*, so that searching for "local" also finds "locative" and "location", for example.
- *Exact* string matching, which means that "local" finds only "local".
- *Initial* exact string matching, which means that "local" finds "local" only in the initial position.
- *Substring* exact string matching, which means that "local" anywhere in the string is matched.

With a numeric argument, you are also asked if you want *adjacent*-word-order matching or *any*-word-order matching if you type more than one word. Any order matching is the default.

The difference between "adjacent" and "any" in the above is that a adjacent-word-order search on "input editor" will find "Using the Input Editor" and "The Input Editor Program Interface", but not "Editor Input". With any-word-order, it will find all three.

This command is also available in Document Examiner. See the section "Show Candidates Command". See the section "Document Examiner".

c-sh-D Long Documentation

Displays the documentation string for the specified function. It prompts for a function name, which you can either type in or select with the mouse. The default is the current function. The documentation string is part of the function definition. There may also be documentation of the function in the documentation database.

When this command does not find a documentation string, it suggests you use Show Documentation (m-X) (or m-sh-D) or Document Examiner to see the function's online documentation. Show Candidates (m-X) searches the online documentation.

c-sh-A Quick Arglist

Displays the argument list for the current function. With a numeric argument, it reads the function name from the minibuffer.

Arglist (m-X)

Displays the argument list of the specified function. It reads the name of the function (from the minibuffer) and displays the argument list in the echo area.

c-sh-V Describe Variable At Point

Displays information in the echo area about the current Lisp variable. The information displayed shows whether it is declared special, whether it has a value, and whether it has documentation put on by **defvar**. When nothing is available, it checks for lookalike symbols in other packages.

m-= Count Lines Region

Displays the number of lines in the region.

c-X L Count Lines Page

Displays the number of lines on the current page (or the buffer, if there are no page delimiters). In parentheses, it displays the number of lines up to the line containing point and the number of lines after the line containing point.

Find Unbalanced Parentheses (m-X)

Finds any parenthesis mismatch error in the buffer. It reads through the entire current buffer and tries to find places in which the parentheses do not balance. It positions point to possible trouble spots, printing out a message that says what the trouble appears to be. This command finds only one such error; if you suspect more errors, run it again.

Trace (m-X)

Traces or untraces a function. It reads the name of the function from the minibuffer and then it pops up a menu of trace options. With an argument, it omits the menu step.

See the special form **trace**.

See the section "Options to **trace**".

Using the Editor Menu

Click Right in Zmacs to display the *Editor menu*, a momentary menu containing editor commands, each of which is a possible choice. Position the mouse cursor over an item and then click the appropriate button to make the

The Editor menu commands are:

Command	*Description*
Arglist	Prints the argument list of the specified function. See the section "General Information-giving Zmacs Commands".
Edit Definition	Prepares to edit the definition of a specified function. See the section "Editing Lisp Programs in Zmacs".
List Callers	Lists all functions that call the specified function See the section "Editing Lisp Programs in Zmacs".
List Definitions	Displays the definitions in a specified buffer. See the section "Editing Lisp Programs in Zmacs".
List Buffers	Prints a list of all the buffers and their associated files See the section "Manipulating Buffers and Files in Zmacs".
Kill Or Save Buffers	Offers a menu of modified files with choices to kill, save, or remove the modification flag from the file. See the section "Manipulating Buffers and Files in Zmacs".
Split Screen	Makes several windows split among the buffers as specified. See the section "Manipulating Buffers and Files in Zmacs".
Compile Region	Compiles the region, or if no region is defined, the current definition. See the section "Editing Lisp Programs in Zmacs".
Indent Region	Indents each line in the region. See the section "Changing Case and Indentation in Zmacs".
Change Typein Style	Sets the default character style for typein. See the section "Working with Regions in Zmacs".
Change Style Region	Changes the character style for the region. See the section "Working with Regions in Zmacs".

Uppercase Region	Changes any lowercase characters in the region to uppercase. See the section "Working with Regions in Zmacs".
Lowercase Region	Changes any uppercase characters in the region to lowercase. See the section "Working with Regions in Zmacs".
Indent Rigidly	Shifts text in the region sideways as a unit. See the section "Changing Case and Indentation in Zmacs".
Indent Under	Fixes indentation to align under either a character that you click on with the mouse cursor or a string read from the minibuffer. See the section "Aligning Indentation in Zmacs".

Using the Minibuffer

Minibuffer Response Format

Most commands expect only one line of response. In these cases, the END key has the same meaning as the RETURN key, terminating the response.

However, for commands that expect one or more lines of response, RETURN has its usual significance, inserting a newline in the minibuffer, and END marks the end of the response.

Minibuffer Response Help

While responding to a prompt, you can press HELP to get documentation describing the current situation. Zmacs tells you exactly what input it expects and what the possible responses are.

More Ways to Enter Minibuffer Responses

Yanking and clicking with the mouse provide quick and simple ways to enter minibuffer responses without having to type them out. Both of these methods are context-sensitive. Yanking works only when you have previously entered a minibuffer response. Clicking works when you click on a name that makes sense in the context of the minibuffer prompt.

Yanking in the Minibuffer

c-m-Y — Repeat Last Minibuffer Command

Repeats a recent minibuffer command. It yanks the displayed default if there is

one; otherwise, it yanks the last thing typed in this context. A numeric argument *n* yanks the *n*th previous one. An argument of 0 lists the history of elements typed in the minibuffer.

For a similar command with string-matching, see the section "Repeat Last Matching MiniBuffer Command (m-X) Zmail Command".

After c-m-Y, m-Y replaces what was yanked with a previous element of the same history, in this case, another minibuffer command. For more details, see the section "Retrieving History Elements".

m-Y Yank Pop

Corrects a yank to use a different element of its history. The most recent command must be a yanking command (c-Y, m-Y, c-sh-Y, m-sh-Y or c-m-Y). The retrieved item (text or command) that was yanked by that command is replaced by the previous element of the corresponding history. The history is rotated (that is, the elements remain in the same order, but the pointer to the *current* element moves with each successive m-Y) to bring this element to the top.

A numeric argument of zero displays the history. A positive numeric argument of *n* moves *n* elements back in the history. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

Getting Information About Buffers and Regions

A good deal of information is available about each Zmacs buffer or region. You can get a count of characters, words, lines, paragraphs, or pages, as well as a count of substrings or Lisp objects.

Count Chars

m-X Count Chars

Counts the characters in the region or in the buffer if there is no region.

Count Words

m-X Count Words

Counts the words in the region or in the buffer if there is no region.

Count Lines

m-X Count Lines

Counts the lines in the region or in the buffer if there is no region.

Count Paragraphs

m-X Count Paragraphs

Counts the paragraphs in the region or in the buffer if there is no region.

Count Pages

m-X Count Pages

Counts the pages in the region, or in the buffer if there is no region.

Count Occurrences

m-X Count Occurrences

Counts how many times a certain substring occurs in the region or in the buffer following point if there is no region.

See the section "How Many".

How Many

m-X How Many

Counts how many times a certain substring occurs in the buffer following point. You are prompted for the substring.

See the section "Count Occurrences".

Moving the Cursor

To make changes at a particular place in a Zmacs buffer, you must move the cursor to that place, since most commands that modify the buffer do so immediately around the cursor.

The cursor movement or *motion* commands:

- View the contents of the buffer
- Redisplay the editor window
- Move the cursor around the buffer using mouse commands
- Move the cursor around the buffer using keystroke commands

For more details, see the section "Motion Commands".

The Editor Window and the Buffer

The *editor window* displays either a portion of your buffer or the whole buffer, depending on the size of the buffer and your current location in it.

When the current buffer is smaller than the exact size of the editor window, Zmacs displays the contents of the buffer at the top of the window and leaves the bottom of the window blank. You cannot tell whether the buffer actually comes to an end where the text stops, since there could be white space and newline characters after the last visible piece of text.

When the buffer is too large to fit on the screen, the editor window shows only a section of the buffer. The part that shows always contains the cursor, so it never vanishes off the top or bottom of the editor window. Zmacs changes the position of the editor window inside the buffer as seldom as possible — usually only when you try to move the cursor off the top or bottom of the screen.

Wraparound Lines in the Editor Window

Lines that are too long to fit across the editor window are displayed on as many physical lines as are necessary. An exclamation point (!) in the (normally blank) last column means that the next physical line is part of the same logical line.

Redisplaying the Window

Whenever you modify the buffer's contents or move point or the mark, Zmacs updates the display to reflect the change. (For a discussion of the mark, see the section "Working with Regions in Zmacs".) This updating can be as simple as moving the cursor or as involved as figuring out the whole display from scratch. These operations are called *redisplay* and Zmacs performs them automatically.

For example, when you move the cursor off the top or bottom of the editor window, a complete redisplay is required. The window has to shift to show a different part of the buffer in order to keep the cursor visible.

You can explicitly tell Zmacs to do a redisplay with the Recenter Window command, invoked by `c-L`. You might want to do this if the cursor gets too close to the top or the bottom of the editor window, and you want to redisplay with the cursor closer to the center so that you can see more context in one direction or the other.

It is important to remember that redisplay operations change only the *display*, not the actual contents of the buffer.

Recentering the Window

`c-L` Recenter Window

Completely redisplays the screen, leaving the cursor near the middle of the editor window.

With a numeric argument of *n*, it leaves the cursor *n* lines from the top of the window. With a negative numeric argument of *-n*, it leaves the cursor *n* lines from the bottom of the window.

Displaying the Next Screen

c-V, SCROLL — Next Screen

Moves the cursor to the beginning of the last visible line in the editor window and redisplays the screen with that line at the top of the window.

With a numeric argument of *n*, it moves the text up *n* lines. With a negative numeric argument *-n*, it moves the text down *n* lines. The cursor does not move (with respect to the text) unless the numeric argument is large enough to slide it off the screen. In that case the cursor remains at the top.

Displaying the Previous Screen

m-V, m-SCROLL — Previous Screen

Moves the cursor to the beginning of the first visible line in the editor window and redisplays the screen with that line at the bottom of the window.

With a numeric argument of *n*, it moves the text down *n* lines. With a negative numeric argument *-n*, it moves the text up *n* lines. The cursor does not move (with respect to the text) unless the numeric argument is large enough to slide it off the screen. In that case the cursor remains at the bottom.

Positioning the Window Around a Definition

c-m-R — Reposition Window

Redisplays, trying to get all of the current function definition in the window. It puts the beginning of the current definition at the top of the window with the current position of the cursor still visible. Doing c-m-R twice pushes comments off the top of the window, making more of the code of a large function visible.

Moving to a Specified Line

m-R — Move To Screen Edge

Moves to the beginning of a specified line on the screen. With no argument, it moves to the beginning of a line near the middle of the screen. The exact line is controlled by the Zmacs variable Center Fraction. A numeric argument specifies a particular line to move to. Negative arguments count up from the bottom of the window. (For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".)

Moving the Cursor with the Mouse

The easiest way to get the cursor where you want it is with the *mouse*. See the section "The Mouse".

Mouse Documentation Line in Zmacs

The mouse documentation line at the bottom of the screen tells you what will happen when you click the mouse. The small arrow cursor tells you where the mouse is and what it is over. What you can do with the mouse depends on what the mouse is over.

There are several sets of possible mouse actions, corresponding to the things the mouse can be over in a Zmacs buffer:

Text or blank space

Scroll bar

The mouse documentation line is two lines high. The top line tells you what you can do with the three mouse buttons. The bottom line tells you what keys you can press to change the action of the mouse buttons. When you press one of these keys, the top mouse documentation line tells you what you can do with the three mouse buttons while that key is held down.

The three mouse buttons are called L for left, M for middle and R for right.

The keyboard keys are called **c** for CONTROL, sh for SHIFT, s for SUPER and m-sh for META-SHIFT.

CONTROL and SHIFT affect editing operations. Their actions are described in this section.

META-SHIFT with the right mouse button gives you access to the *Window Operation menu*, which allows you to move, reshape, expand, bury, kill, or hardcopy the window or pane.

SUPER with the right mouse button gives you access to the *Presentation Debugging menu*.

Mouse Over Text or Blank Space

Here is a description of what you can do with the mouse when it is over text or blank space in a Zmacs buffer.

Notation	*Description*
L:Move point	Performs two separate actions, depending on whether you click Left or hold Left down. • Relocates the cursor: position the mouse cursor to the desired location and click Left. If the cursor is over blank space, point is moved to the end of the line. • Marks a region: position mouse cursor to desired location, hold left button down, move mouse cursor to end point of desired region and release the button.
sh-L:Move to point	Relocates the mouse arrow cursor to point (where the blinking cursor appears).

M:Mark thing

Marks (makes into a region) the object on which you click. Clicking after the end of a line or before the first nonblank character of a line marks the whole line. Clicking on a word marks that word.

In Lisp mode, however, if that word is part of what could be a symbol's printname, it marks that whole symbol name. Clicking on an open or close parenthesis marks all the text between that parenthesis and its matching parenthesis, including the parentheses. Clicking on an open or close quotation mark (") marks the whole quoted string. Clicking between words marks all text up to the end of the next word or possible symbol printname, depending on mode. For a complete description of *marking* regions, see the section "Working with Regions in Zmacs".

c-M:Copy Mouse

Inserts the object on which you click, as though you had typed it. This allows you to build a program or document by selecting things already appearing on your screen, in the manner of a menu. Hold down the CONTROL key and click Middle on the object you want to copy: it is inserted as though you had just typed it. If you change your mind, and want to remove what you have just inserted, press c-W, and it is removed.

The object copied can be a word, a printed representation of a Lisp symbol, a parenthesized or quoted group of words, a printed representation of a lisp list or string, or a line. What object is picked up by clicking c-(M) on it is determined by the same rules as clicking Middle on Mark Thing in Lisp Mode. That is,

- Clicking after the end of a line or before the first nonblank character of a line copies the whole line. Clicking on a word picks up that whole word, or possible Lisp symbol printname of which that word could be part.

- Clicking on an open or close parenthesis copies the text between that parenthesis and its matching parenthesis, including the parentheses. Clicking on an open or close quotation mark (") copies the whole quoted string. Clicking between words copies all text up to the end of the next word (or possible symbol printname).

Appropriate spaces are placed before the inserted object.

sh-M:Save/Kill/Yank

Performs one of four related actions:

- If there is a region, it saves the region in the kill history while leaving it in the buffer (like m-W).

- If the last command saved the region, it removes it from the buffer (like c-W, except it does not save).
- If the above two conditions do not apply, it yanks the first element from the kill history (like c-Y).
- If the last command was a yank command, it yanks the next item from the kill history (like m-Y).

For a complete description of *saving*, *killing*, and *yanking* regions, see the section "Working with Regions in Zmacs".

R:Editor menu — Displays a Zmacs menu offering mouse-sensitive Zmacs commands.

sh-R:System menu — Displays a System menu.

Mouse Over Scroll Bar

The scroll bar consists of a pair of dotted lines with a shaded section between them. The shaded section, or elevator, represents the portion of the buffer visible on the screen and the dotted lines, or shaft, represent the entire buffer. In general, you see only the shaft, but not the elevator. Move the mouse cursor over the shaft and the elevator appears. The elevator remains visible until the buffer changes in size, whereupon it disappears until you move the mouse cursor over the shaft again.

Motion Commands

Zmacs word, sentence, and paragraph motion commands all have strict definitions for where words, sentences, and paragraphs begin and end. You can modify all these definitions.

Summary of Cursor Motion Commands

These are the Zmacs commands that you can use to move the cursor:

c-A — Beginning of Line
Moves to the beginning of the line.

c-E — End of Line
Moves to the end of the line.

c-F — Forward
Moves forward one character.

c-B — Backward
Moves backward one character.

m-F — Forward Word
Moves forward one word.

m-B — Backward Word
Moves backward one word.

m-E — Forward Sentence
Moves to the end of the sentence in text mode.

m-A — Backward Sentence
Moves to the beginning of the sentence in text mode.

c-N — Down Real Line
Moves down one line.

c-P — Up Real Line
Moves up one line.

m-] — Forward Paragraph
Moves to the start of the next paragraph.

m-[— Backward Paragraph
Moves to the start of the current (or last) paragraph.

c-X] — Next Page
Moves to the next page.

c-X [— Previous Page
Moves to the previous page.

c-V, SCROLL — Next Screen
Moves down to display the next screenful of text.

m-V, m-SCROLL — Previous Screen
Moves up to display the previous screenful of text.

m-< — Goto Beginning
Moves to the beginning of the buffer.

m-> — Goto End
Moves to the end of the buffer.

Numeric Arguments and the Motion Commands

All the motion commands allow numeric arguments. For the most part, these numeric arguments are interpreted as repeat counts.

Example of Numeric Arguments with Motion Commands

m-F moves the cursor forward one word; m-13F moves the cursor forward 13 words.

Negative Numeric Arguments and Motion Commands

Most of the motion commands come in pairs, with one command for forward motion over a particular unit and one command for backward motion. Both kinds of commands often interpret negative numeric arguments by reversing the direction of motion.

These conventions — that Zmacs interprets numeric arguments as repeat counts, and that negative numeric arguments reverse the direction of motion — together make up the *motion convention*.

Example of Negative Numeric Arguments with Motion Commands

`m- -13F` moves point backward 13 words. `m-13B` has exactly the same effect.

Motion by Character

A Zmacs *character* can be any letter, number, or punctuation character.

Forward Character

`c-F` Forward

Moves the cursor forward over one character. `c-F` interprets numeric arguments as repeat counts.

Negative numeric arguments reverse the direction of motion. For example, `c-3B` and `c- -3F` both move the cursor backwards three characters.

Backward Character

`c-B` Backward

Moves the cursor backward over one character. `c-B` interprets numeric arguments as repeat counts.

Negative numeric arguments reverse the direction of motion. For example, `c-3 c-B` and `c-- c-3 c-F` both move the cursor backwards three characters.

Motion by Word

Zmacs generally considers a *word* to consist of a sequential string of alphanumeric characters, that is, any combination of the characters a-z, A-Z, and 0-9. Different major modes define their own delimiter characters. For example, in Text Mode an apostrophe (') is part of a word, but in other modes it is a delimiter. (For mode description, see the section "Zmacs Major Modes".)

Forward Word

`m-F` Forward Word

Moves the cursor forward one word. Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

`m-F` always places the cursor at the end of a word. If the cursor is in the middle of a word, `m-F` moves the cursor to the end of that word.

Backward Word

m-B Backward Word

Moves the cursor backward one word. Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-B always places the cursor at the beginning of a word. If the cursor is in the middle of a word, m-B moves the cursor to the beginning of that word.

Motion by Sentence

Description of Zmacs Sentence Delimiters

According to Zmacs, sentences can end with question marks, periods, and exclamation points. Furthermore, these punctuation marks only end a sentence when followed by:

- A newline
- A space followed by either a newline or another space

However, Zmacs allows any number of *closing characters*, which are ", ',), and], between the sentence-ending punctuation and the white space that follows it. A sentence also starts after a blank line.

This corresponds closely to standard typing conventions. Zmacs does not recognize a period followed by one space as the end of a sentence, for example, as in "e.g. " or "Dr. ".

Forward Sentence

m-E Forward Sentence

Moves the cursor forward one sentence.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-E always places the cursor at the end of a sentence. If the cursor is in the middle of a sentence, m-E moves the cursor to the end of that sentence.

Backward Sentence

m-A Backward Sentence

Moves the cursor backward one sentence.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-A always places the cursor at the beginning of a sentence. If the cursor is in the middle of a sentence, m-A moves the cursor to the beginning of that sentence.

Motion by Line

Lines are delimited by special characters called *newlines*.

Down Line

c-N Down Real Line

Moves the cursor straight down to the corresponding column of the next line. If the cursor is positioned in the middle of the line, c-N moves it to the middle of the next one.

With a numeric argument *n*, it moves the cursor down *n* lines. Moving down a negative number of lines is the same as moving up.

Up Line

c-P Up Real Line

Moves the cursor straight up to the corresponding column of the previous line. If the cursor is positioned in the middle of the line, c-P moves it to the middle of the previous one.

With a numeric argument of *n*, it moves the cursor up *n* lines. Moving up a negative number of lines is the same as moving down.

Beginning of Line

c-A Beginning of Line

Moves the cursor to the beginning of the current line.

With a numeric argument of *n*, it moves the cursor to the beginning of the *n*th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

End of Line

c-E End Of Line

Moves the cursor to the end of the current line.

With a numeric argument of *n*, it moves the cursor to the end of the *n*th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

Goal Column and the Motion Commands

Set Goal Column

c-X c-N Set Goal Column

Sets the default column position (*goal column*). The goal column sets point position for c-N and c-P. It disables the default action of matching the goal column to point's current column and sets the goal column to zero instead. With a numeric argument *n*, sets the goal column to *n*. c-U turns it off (sets it back to the default state of keeping cursor in same horizontal position for c-N and c-P).

Motion by Lisp Expression

Motion by Lisp expression repositions the cursor according to Lisp code delimiters: *lists* and *expressions*. A list is something enclosed in balanced parentheses. A Lisp expression is any readable printed representation of a Lisp object.

Forward List

c-m-N Forward List

Moves forward over one list. It accepts a numeric argument for repetition count.

Backward List

c-m-P Backward List

Moves backward over one list. It accepts a numeric argument for repetition count.

Motion Along One Nesting Level

Point always sits either between two expressions or in the middle of a Lisp object (excluding a list or **nil**).

c-m-F Forward Sexp

Moves point to the end of a surrounding Lisp object (excluding a list or **nil**) if there is one, or past the Lisp expression immediately to the right if not.

If parentheses are unbalanced to such an extent that it doesn't make sense to talk about "the expression on the right", this command gives an error message and does not move point at all.

c-m-F observes the motion convention for numeric arguments.

c-m-B Backward Sexp

Moves point to the beginning of a surrounding Lisp object (excluding a list or **nil**) if there is one, or to the beginning of the Lisp expression immediately to the left if not.

If parentheses are unbalanced to such an extent that it doesn't make sense to talk about "the expression on the left", this command gives an error message and does not move point at all.

c-m-B observes the motion convention for numeric arguments.

Motion Up and Down Nesting Levels

c-m-D — Down List

Moves point forward past any intervening Lisp object (excluding a list or **nil**) to the level of list structure and leaves point just to the right of the open parenthesis of that expression.

With a numeric argument of *n*, it moves down *n* nesting levels.

c-m-U — Backward Up List
c-m-(

Backs up out of nesting levels. It moves backward one level of list structure. It searches for an open parenthesis and leaves point to the left of that open parenthesis. Also, if called inside of a string, it moves back up out of that string, leaving point to the left of its starting quote. It accepts numeric arguments for repetition count.

With a numeric argument of *n*, it moves up *n* nesting levels.

c-m-) — Forward Up List

Moves forward out of nesting levels. It moves forward one level of list structure. It searches for a close parenthesis and leaves point to the right of that close parenthesis. Also, if called inside of a string, it moves up out of that string, leaving point to the right of its ending quote. It accepts numeric arguments for repetition count.

With a numeric argument of *n*, it moves up *n* nesting levels.

Motion Among Top-Level Expressions

A Lisp file contains a sequence of expressions that we call *top-level expressions*, to distinguish them from their own subexpressions. Zmacs assumes that top-level expressions begin with an open parenthesis against the left margin. It does *not* parse top-level expressions by balancing parentheses, since parentheses do not always balance while programs are being written. The indentation represents the *programmer's* conception of program structure, and provides a better guide. So by *top-level expression*, we mean a section of text delimited by open parentheses at the beginning of two lines.

In code that includes a string containing a carriage return followed by an open parenthesis, show you that the open parenthesis does not start a top-level expression by putting a slash in front of it.

c-m-A — Beginning Of Definition
c-m-[

Moves point to the beginning of the current top-level expression.

With a positive numeric argument *n*, it moves back *n* top-level expressions. With a negative numeric argument *-n*, it moves forward *n* top-level expressions.

`c-m-E` End Of Definition
`c-m-]`

Moves point to the end of the current top-level expression.

With a positive numeric argument *n*, it moves forward *n* top-level expressions. With a negative numeric argument *-n*, it moves back *n* top-level expressions.

`m-)` Move Over)

Moves past the next close parenthesis, then does Indent New Line. It removes any whitespace between point and the close parenthesis before moving over it. With a positive argument *n*, after finding the next close parenthesis and deleting whitespace before it, it moves past *n*-1 additional close parentheses before doing Indent New Line. It ignores numeric arguments that are less than 1.

Motion by Paragraph

A paragraph is delimited by:

- A newline followed by blanks (spaces or tabs)
- A blank line
- A Page character alone on a line
- Various other mode-dependent factors (for example, a line that does not begin with the fill-prefix). See the section "Filling a Region".

Forward Paragraph

`m-]` Forward Paragraph

Moves the cursor forward one paragraph.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

`m-]` always places the cursor at the end of a paragraph. If the cursor is in the middle of a paragraph, `m-]` moves the cursor to the end of that paragraph.

Backward Paragraph

`m-[` Backward Paragraph

Moves the cursor one paragraph backward.

Numeric arguments are interpreted as repeat counts; negative numeric arguments reverse the direction of motion.

m-[always places the cursor at the beginning of a paragraph. If the cursor is in the middle of a paragraph, m-[moves the cursor to the beginning of that paragraph.

Motion by Page

Pages are delimited by Page characters. You can insert a Page character by pressing the PAGE key. The Page delimiter belongs to the page that precedes it and is therefore the last character on that page.

Forward Page

c-X] Next Page

Moves the cursor to the beginning of the next page; that is, puts the cursor immediately after the nearest following Page delimiter. If the buffer does not contain a Page delimiter, it goes to the end of the buffer.

With a positive numeric argument *n*, it repeats this operation *n* times to move forward *n* pages. A negative numeric argument *-n* moves the cursor backward instead.

c-X [always places the cursor immediately to the right of the next Page delimiter. If the cursor is immediately to the left of the Page delimiter, c-X] goes to the beginning of the page after next rather than just moving forward one character.

Backward Page

c-X [Previous Page

Moves the cursor to the beginning of the previous page; that is, puts the cursor immediately after the nearest preceding Page delimiter. If the buffer does not contain a Page delimiter, it goes to the beginning of the buffer.

With a positive numeric argument *n*, it repeats this operation *n* times to move backward *n* pages. A negative numeric argument *-n* moves the cursor forward instead.

c-X [always places the cursor at the beginning of a page. If the cursor is already at the beginning of the page, c-X [moves it to the beginning of the previous page.

Motion with Respect to the Whole Buffer

Use the following commands to go to the Beginning/End of the buffer.

Goto Beginning

M-< Goto Beginning

Moves the cursor to the beginning of the buffer.

With a numeric argument *n* between 0 and 10, it moves the cursor to a place *n*/10 of the way (counted in lines) from the beginning of the buffer towards the end.

Goto End

M-> Goto End

Moves the cursor to the end of the buffer. You can use M-> if you are in doubt as to the exact place on the screen where the buffer stops.

With a numeric argument *n* between 0 and 10, it moves the cursor to a place *n*/10 of the way (counted in lines) from the end of the buffer towards the beginning.

Basic Text Editing

This section describes how to edit text in Zmacs. It shows how to insert and delete, and search and replace character strings, as well as how to change case and indentation, use word abbreviations for Zmacs commands, and how to use character styles.

Inserting Text in Zmacs

Zmacs is always ready to accept an insertion. To insert new text anywhere in the buffer, position the cursor at the place you want the new text to go and type the new text. Zmacs always inserts characters at the cursor. The text to the right of the cursor is pushed along ahead of the text being inserted.

Inserting Characters

When you type in new text, you are actually issuing Zmacs commands. Ordinary printing characters are called *self-inserting* because when you type one, it inserts itself into the text in your buffer.

You can give numeric arguments to the keystrokes that insert printing characters into the buffer; Zmacs interprets these arguments as repeat counts. See the section "Numeric Arguments".

Example: C-80 * inserts a line of 80 asterisks at the cursor.

Starting a New Line

Newline characters delimit lines of text. They have no visible printed form, but are present at each line break. You can break one line into two lines by inserting a newline (pressing RETURN) where desired. Similarly, you can merge two lines into one by deleting the intervening newline.

Correcting Typos

To correct text you have just inserted, use the RUBOUT key. RUBOUT deletes the character *before* the cursor (not the one over which the cursor is positioned; that is the character *after* the cursor). The cursor and the rest of that line move to the left.

When given a numeric argument, RUBOUT saves the succession of deleted characters.

Example: c-20 RUBOUT kills the previous 20 characters and saves them together.

See the section "Deleting vs. Killing Text".

When the cursor is positioned on the first character on a line and you press RUBOUT, the preceding newline character is deleted and Zmacs appends the text on that line to the end of the previous line.

Wrapping Lines

When you add too many characters to one line without breaking it with a RETURN, the line grows to occupy two (or more) lines on the screen, with an exclamation point at the extreme right margin of all but the last of them. The ! means that the following screen line is not really a distinct line in the file, but just the continuation of a line too long to fit the screen.

Inserting Formatting Characters

You can insert most characters directly into the buffer by simply typing them, but other characters act as editing commands and do not insert themselves. If you need to insert a character that is normally a command (for example, TAB or RUBOUT), use the c-Q (Quoted Insert) command first to tell Zmacs to insert the following character into the buffer literally. c-Q prompts in the echo area for the character to be inserted and inserts it into the text.

Deleting and Transposing Text in Zmacs

Deleting vs. Killing Text

Deleting text merely gets rid of it, but Zmacs deletion commands not only *kill* text but also get it back. These commands save killed text in a *history* stack. Other commands, called *yanking* commands, retrieve elements from the history.

Deletion commands that operate on single characters do not save what they delete. However, by giving them a numeric argument, thus telling them to delete several characters, they too save the deleted text.

The commands that delete only white space do not save it.

Deleting Text

Most commands that erase text from the buffer save it so that you can get it back if you change your mind, or move or copy it to other parts of the buffer. These commands are known as *kill* commands. The rest of the commands that erase text do not save it; they are known as *delete* commands. The delete commands include c-D and RUBOUT, which delete only one character at a time, and those commands that delete only spaces or line separators. (However, when given a numeric argument, c-D and RUBOUT do save that sequence of deleted characters on the kill ring.) Commands that can destroy significant amounts of information generally kill. The commands' names and individual descriptions use the words "kill" and "delete" to say which they do.

If you issue a kill command by mistake, you can retrieve the text with c-Y, the Yank command. For details on killing and retrieving text, see the section "Working with Regions in Zmacs".

Zmacs Commands for Deleting Text

This section summarizes Zmacs commands for deleting text.

c-D — Delete Forward
Deletes the character after point.

RUBOUT — Rubout
Deletes the character before point.

m-D — Kill Word
Kills forward one word.

m-RUBOUT — Backward Kill Word
Kills backward one word.

m-K — Kill Sentence
Kills forward one sentence.

c-X RUBOUT — Backward Kill Sentence
Kills backward one sentence.

c-K — Kill Line
Kills to the end of the line or kills an end of line.

c-W — Kill Region
Kills region (from point to mark).

c-m-K — Kill Sexp
Kills forward over exactly one Lisp expression.

c-m-RUBOUT — Backward Kill Sexp
Kills backward over exactly one Lisp expression.

m-\ — Delete Horizontal Space
Deletes any spaces or tabs around point.

c-X c-O — Delete Blank Lines
Deletes any blank lines following the end of the current line.

M-^ Delete Indentation

Deletes RETURN and any indentation at front of line.

Deleting and Transposing Characters

Deleting the Last Character

RUBOUT Rubout

Deletes the character immediately to the left of the cursor.

If the cursor is at the beginning of a line, RUBOUT deletes the newline character at the end of the previous line, thus appending the current line to the previous one.

With a positive numeric argument of *n*, RUBOUT deletes the *n* characters immediately to the left of the cursor. With a negative numeric argument of *-n*, it deletes the *n* characters immediately to the right of the cursor. With any numeric argument, it saves the deleted characters on the kill history.

Deleting the Current Character

C-D Delete Forward

Deletes the character at the cursor.

If the cursor is at the end of a line, C-D deletes the newline character at the end of the line, thus appending the next line to the current one.

With a positive numeric argument of *n*, C-D deletes the *n* characters immediately to the right of cursor. With a negative numeric argument of *-n*, it deletes the *n* characters immediately to the left of cursor. With any numeric argument, it saves the deleted characters on the kill history.

Transposing Characters

C-T Exchange Characters

Transposes two characters (the ones on each side of the cursor).

If the cursor is not at the end of a line, C-T transposes the character at the cursor and the character to the left of the cursor and advances the cursor one character. The result is that the character to the left of the cursor has been "dragged" one character position to the right. Repeated use of C-T continues to pull that character forward. This is useful when you are typing and enter two characters in the wrong order (for example, teh for the).

If the cursor is at the end of a line, C-T transposes the two preceding characters.

With a nonzero numeric argument of *n*, C-T deletes the character to the left of the cursor, moves forward *n* characters, and reinserts the deleted character. When *n* is negative, the cursor moves backwards.

c-T can only be given a numeric argument of zero when the mark is active. In this case, it exchanges the characters at point and mark.

Deleting and Transposing Words

Introduction

For a complete description of how words are delimited, see the section "Motion by Word".

Deleting the Current Word

m-D Kill Word

Kills the word after the cursor and saves it on the kill history. If the cursor is in the middle of a word, m-D kills from the cursor to the end of that word.

With a numeric argument *n*, it kills *n* words forward from the cursor. If *n* is negative, it kills backward.

Deleting the Previous Word

m-RUBOUT Backward Kill Word

Kills the word before the cursor and saves it on the kill history. If the cursor is in the middle of a word, m-RUBOUT kills from the cursor to the beginning of that word.

With a numeric argument *n*, it kills *n* words backward from the cursor. If *n* is negative, it kills forward.

Transposing Words

m-T Exchange Words

Transposes the current word and the previous one. If the cursor is at the end of a line, m-T transposes the last word on that line and the first one on the next, regardless of the amount or type of white space between them.

With a nonzero numeric argument *n*, m-T goes to the beginning of the current word, deletes the previous word, goes forward *n* words, and reinserts the deleted word. Moving forward a negative amount is equivalent to moving backward. An argument of zero transposes the words at point and mark.

Deleting and Transposing Lisp Expressions

Motion by Lisp expression repositions the cursor according to Lisp code delimiters: *lists* and *expressions*. A list is something enclosed in balanced parentheses. A Lisp expression is any readable printed representation of a Lisp object.

Deleting the Current Lisp Expression

c-m-K Kill Sexp

Kills the Lisp expression immediately to the right of point and saves it on the kill history.

With a numeric argument of *n*, it kills the *n* succeeding expressions. It is an error to kill off the end of a containing expression. When the numeric argument is negative, it kills backwards from point the same way.

Deleting the Previous Lisp Expression

c-m-RUBOUT Backward Kill Sexp

Kills the Lisp expression immediately to the left of point and saves it on the kill history.

With a numeric argument of *n*, it kills the *n* preceding expressions. It is an error to kill off the beginning of a containing expression. When the numeric argument is negative, it kills forward from point the same way.

Deleting the List Containing the Current Lisp Expression

Kill Backward Up List (c-m-X)

Deletes the list that contains the Lisp expression after point, but leaves that expression itself.

Transposing Lisp Expressions

c-m-T Exchange Sexps

Point must be between two expressions to use this command.

Exchanges the two expressions on either side of point, preserving current indentation.

With a numeric argument of *n*, it deletes the expression to the left of point, moves forward *n* expressions, and reinserts the deleted expression. With a negative numeric argument, it exchanges expressions in the opposite direction. An argument of zero transposes the expressions at point and mark.

Deleting and Transposing Lines

Lines are delimited by special characters called *newlines*.

Down Line

c-N Down Real Line

Moves the cursor straight down to the corresponding column of the next line. If the cursor is positioned in the middle of the line, c-N moves it to the middle of the next one.

With a numeric argument *n*, it moves the cursor down *n* lines. Moving down a negative number of lines is the same as moving up.

Up Line

c-P Up Real Line

Moves the cursor straight up to the corresponding column of the previous line. If the cursor is positioned in the middle of the line, c-P moves it to the middle of the previous one.

With a numeric argument of *n*, it moves the cursor up *n* lines. Moving up a negative number of lines is the same as moving down.

Beginning of Line

c-A Beginning of Line

Moves the cursor to the beginning of the current line.

With a numeric argument of *n*, it moves the cursor to the beginning of the *n*th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

End of Line

c-E End Of Line

Moves the cursor to the end of the current line.

With a numeric argument of *n*, it moves the cursor to the end of the *n*th line after the current one, where the current line is numbered 1, the preceding line is numbered 0, and so on.

Deleting the Current Line

c-K Kill Line

Kills a line at a time and saves it on the kill history.

If the cursor is at the end of a line, c-K kills the newline, merging the current line with the next one. If the cursor is elsewhere on the line, c-K kills the text between the cursor and the end of the current line.

With a numeric argument *n*, c-K kills up to the *n*th newline following the cursor. When *n* is negative or zero, c-K kills back to the *1-n*th newline before the cursor. c-0 c-K kills from the cursor back to the beginning of the line that it is on.

Deleting Backward on the Line

CLEAR INPUT Clear

Kills backward to the start of the current line and saves it on the kill history. If point is already at the beginning of the line, it kills the previous line. With a numeric argument *n*, it kills between point and the start of the *n*th line *above* the current line. Use CLEAR INPUT when entering a new line of text, to delete the whole line.

Transposing Lines of Text

c-X c-T Exchange Lines

Exchanges the current line with the previous one and leaves the cursor at the beginning of the next line.

With a nonzero numeric argument *n*, c-X c-T deletes the previous line (including the following newline), moves down *n* lines, and reinserts the deleted line.

With a numeric argument of zero, c-X c-T exchanges the lines at point and mark, advancing both point and mark to the beginning of the next line.

Deleting Sentences

According to Zmacs, sentences can end with question marks, periods, and exclamation points. Furthermore, these punctuation marks only end a sentence when followed by:

- A newline
- A space followed by either a newline or another space

However, Zmacs allows any number of *closing characters*, which are ", ',), and], between the sentence-ending punctuation and the white space that follows it. A sentence also starts after a blank line.

This corresponds closely to standard typing conventions. Zmacs does not recognize a period followed by one space as the end of a sentence, for example, as in "e.g. " or "Dr. ".

Deleting the Current Sentence

m-K Kill Sentence

Kills the text between the cursor and the end of the current sentence, and saves it on the kill history.

With a numeric argument of *n*, m-K kills the text between the cursor and the end of the *n*th sentence after the cursor, *counting* the current sentence. If the argument is negative, m-K kills -*n* sentences *before* the cursor, counting the current sentence.

Deleting the Previous Sentence

c-X RUBOUT — Backward Kill Sentence

Kills backward one sentence and saves it on the kill history.

With a negative argument, c-X RUBOUT kills forward one sentence in a similar manner.

Killing and Yanking Text

The kill history contains deleted text and is the history that saves the results of the commands described in this chapter. It allows you to move text from one editor window to another, for example, from the editor to a Lisp Listener. The *yanking* commands described below retrieve elements from the kill history.

Yanking is getting back previously killed text or previously typed minibuffer commands. Using a few keystrokes, you can retrieve editing commands you have previously typed or restore any piece of text you have previously killed. *Popping* is moving through the *history* of previous commands or killed text.

To get back the last piece of text killed, press c-Y. To work your way back through the *kill history*, press m-Y repeatedly.

To get back the last command typed, press c-m-Y. To work your way back through the *command history*, press m-Y repeatedly. This technique works only on editor commands that use the minibuffer in some way. Commands such as c-N or c-P are not kept in the history.

You can also select items from the entire kill history or command history, or search a part of these histories using string-matching.

Here are the commands for yanking in the kill history:

c-Y	Yanks the first element in the kill history.
m-Y	After c-Y, m-Y yanks the previous element in the kill history. Subsequent m-Ys move down the kill history.
c-0 c-Y	Displays the elements of the kill history. You can click left on any item in the history to insert it in your buffer. Only the first 25 elements of the history are shown. Click left on (*N* more elements in history.) to display the rest of the history.
c-sh-Y	Yanks the first element in the kill history that matches a string you supply.
m-sh-Y	After c-sh-Y, m-sh-Y yanks a previous element in the kill history that matches the string you supplied. Subsequent m-sh-Ys move down the kill history matching the same string. If pressed after a c-Y, m-sh-Y prompts for a string to match.
c-0 c-sh-Y	Displays the elements of the kill history that match a string you supply. You can click left on any item in the history to insert it in your buffer. Only the first 25 matching elements of

the history are shown. Click left on (*N* more elements in history.) to display the rest of the matching history.

What Histories Save

Zmacs uses several histories:

Type	*Description*
Kill	History of text deleted or saved. The kill history is shared with the input editor, thus allowing you to move text between files and the Lisp Listener.
Replace	History of arguments to Query Replace (m-X) and related commands. See the section "Searching, Replacing, and Sorting in Zmacs".
Buffer	History of editor buffers visited in this window. See the section "Manipulating Buffers and Files in Zmacs".
Pathname	History of file names that have been typed.
Command	History of editor commands that use the minibuffer, and their arguments. Commands that do not use the minibuffer, for example, m-RUBOUT, are not recorded in the history.
Definition	History of names of definitions that have been typed.

There is no limit to the length of a history, but the typeout window displays only the first 25 elements of the history. When the history contains more than 25 elements, the screen displays a mouse-sensitive line: *n* more elements in history. Clicking Left displays the rest of the history.

Viewing the Kill History

You can view and retrieve either the complete kill history or a history of all text including a string you specify.

c-0 c-Y

Displays the elements of the kill history:

```
Kill history:
  1: last piece of killed text
  2: next-to-last piece of killed text
  3: this one is a very long piece of killed text...
  .
  .
  .
(End of history.)
```

Point with the mouse to any element of the history and click Left to insert it into the current buffer. Only the first 25 elements of the history are shown. Click Left on (N more elements in history.) to display the rest of the history.

c-0 c-sh-Y

Displays the elements of the kill history that match a string you supply. For example, if you supply the string "shift", you might see the following:

```
Kill history:
  3: first piece of killed text to click on and shift to someplace else
  5: Yon Cassius has a lean and shifty look
  9: Using the shift key with yank commands adds string searching
  .
  .
  .
(End of history.)
```

Point with the mouse to any element of the history and click Left to insert it into the current buffer. Only the first 25 elements of the matching history are shown. Click Left on (N more elements in history.) to display the rest of the matching history.

Viewing the Editor Command History

You can view and retrieve either the complete command history or a history of all commands including a string you specify.

c-0 c-m-Y

Displays the elements of the editor command history:

```
Command history:
  1: Control-X Control-F last-file-read-in
  2: Help A
  3: Control-X Control-F other-file-read-in
  .
  .
  .
(End of history.)
```

This command is context-sensitive. When typed at a Lisp Listener, it lists the recent commands typed there. When typed at the minibuffer, it lists the history appropriate to what is being read in the minibuffer, for example, a pathname or the name of a definition.

The editor history contains only commands that use the minibuffer. Commands such as c-N and c-P are not in the history.

c-0 c-m-sh-Y

Displays the elements of the editor command history that match a string you supply. For example, if you supply the string "sh", you might see the following:

```
Command history:
  3: 0 Control-Meta-Shift-Y oregano
  5: Show Mail
  7: Show Spell Dictionaries
112: Meta-X Finish Patch
  .
  .
(End of history.)
```

This command is context-sensitive. When typed at a Lisp Listener, it lists the matching commands typed there. When typed at the minibuffer, it lists the matching history appropriate to what is being read in the minibuffer, for example, a pathname or the name of a definition.

The editor history contains only commands that use the minibuffer. Commands such as c-N and c-P are not in the history.

Using the Mouse on History Elements

History elements are mouse-sensitive. Click on an element of the kill history to yank it to point. Click on an element of the command history to reexecute the command.

Retrieving History Elements

c-Y — Yank

Yanks back and inserts the last text killed or saved. If you have moved point since you killed the text, put point where you want the killed text to go before pressing c-Y. Point ends up after the text, and mark before the text. An argument of c-U puts point before the text instead. A numeric argument of 0 displays the kill history and does not yank anything. A nonzero numeric argument selects an element of the kill history.

Note that when you yank text, Concordia automatically places a blank space before the text you are yanking. You can disable this feature by setting the **si:*ie-fixup-whitespace*** variable to nil.

c-sh-Y — Yank Matching

Yanks back and inserts the last text killed or saved that matches a string you supply. If you have moved point since you killed the text, put point where you want the killed text to go before pressing c-Y. Point ends up after the text, and mark before the text. An argument of c-U puts point before the text instead. A numeric argument of 0 displays the kill history and does not yank anything. A nonzero numeric argument selects an element of the kill history.

c-m-Y — Repeat Last Minibuffer Command

Repeats a recent minibuffer command. It yanks the displayed default if there is one; otherwise, it yanks the last thing typed in this context. A numeric argument *n* yanks the *n*th previous one. An argument of 0 lists the history of elements typed in the minibuffer.

For a similar command with string-matching, see the section "Repeat Last Matching MiniBuffer Command (m-X) Zmail Command".

c-m-sh-Y Repeat Last Matching Minibuffer Command

Yanks back and repeats the last minibuffer command that includes a string you specify. m-sh-Y yanks back previous commands that contain the same string.

m-Y Yank Pop

Corrects a yank to use a different element of its history. The most recent command must be a yanking command (c-Y, m-Y, c-sh-Y, m-sh-Y or c-m-Y). The retrieved item (text or command) that was yanked by that command is replaced by the previous element of the corresponding history. The history is rotated (that is, the elements remain in the same order, but the pointer to the *current* element moves with each successive m-Y) to bring this element to the top.

A numeric argument of zero displays the history. A positive numeric argument of *n* moves *n* elements back in the history. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

m-sh-Y Yank Pop Matching

Corrects a yank to use a different element of its history. The most recent command must be a yanking command (c-Y, m-Y, c-sh-Y, m-sh-Y or c-m-Y). If you supplied a matching string in the previous command, that string is used. Otherwise, m-sh-Y prompts for a string. The retrieved item (text or command) that was yanked by that command is replaced by the previous element of the corresponding history. The history is rotated (that is, the elements remain in the same order, but the pointer to the *current* element moves with each successive m-sh-Y) to bring this element to the top.

A numeric argument of 0 displays the history. A positive numeric argument of *n* moves *n* elements back in the history. A negative numeric argument moves to a newer history element; this only makes sense after you rotate the history.

Kill Merging

Normally, each kill command pushes a new block onto the kill history. However, two or more kill commands in a row combine their text into a single element on the history, so that a single c-Y command gets it all back as it was before it was killed. This means that you do not have to kill all the text in one command; you can keep killing line after line, or word after word, until you have killed it all, and you can still get it all back at once.

Commands that kill forward from point add onto the end of the previous killed text. Commands that kill backward from point add onto the beginning. This way, any sequence of mixed forward and backward kill commands puts all the killed text into one element without rearrangement.

If a kill command is separated from the last kill command by other commands, it starts a new element on the kill history, unless you tell it not to by saying c-m-W (Append Next Kill) in front of it. The c-m-W tells the following command, if it is a kill command, to append the text it kills to the last killed text, instead of starting a new element. With c-m-W, you can kill several discrete pieces of text and accumulate them to be yanked back in one place.

c-m-W Append Next Kill

Makes the next kill command append text to the newest element of the kill history.

Searching, Replacing, and Sorting in Zmacs

Searching in Zmacs

Like other editors, Zmacs has commands for searching for an occurrence of a string. Zmacs search commands are *incremental*; that is, they begin to search as soon as you type the first character.

This section describes how to search incrementally forward and backward in the buffer, as well as a method for specifying a complete search string first and then specifying a direction in which to search.

Incremental Search

The command to search is c-S (Incremental Search). c-S reads in characters and positions the cursor at the first occurrence of the characters that you have typed. If you type c-S and then t, the cursor moves right after the first t. Type an r, and see the cursor move to after the first tr. Add a y and the cursor moves to the first try after the place where you started the search. At the same time, the try has echoed at the bottom of the screen. Stop typing when you have typed enough characters to identify the place you want.

You can rub out any character you type. After the try, pressing RUBOUT makes the y disappear from the bottom of the screen, leaving only tr. The cursor moves back to the tr. Rubbing out the r and t moves the cursor back to where you started the search. To exit from the search, press END or ESCAPE. You can also use ABORT to exit from the search. To abort out of the search and return to the original starting point in the buffer, use c-G.

If you want to search for something else, press CLEAR INPUT to erase the current search string. You are still in the search loop, so type another search string.

If the string cannot be found with c-S, type c-R to search backward for the default string. Zmacs remembers the default search string — you can reinvoke the search at any time using c-S c-S, to search forward for it, or c-R c-R to search backward.

c-S Incremental Search

Searches for a character string while you type it, searching forward to the end of the buffer. It prompts for a string in the echo area with I-Search:. As you type characters in, c-S displays the accumulating string in the echo area and searches for it at the same time. If no string is found, it displays Failing I-Search:. When it locates the string, it puts the cursor after it so that repeated c-Ss locate subsequent occurrences of the default string in the buffer.

RUBOUT	Removes a character and backs up the search to the last match.
ESCAPE	When typed before any search characters, switches to String Search. See the section "Zmacs String Search".
END or ESCAPE	Exits the search.
c-G	Exits the search and returns to original starting point in the buffer.
c-Q	Quotes the next character, to prevent it from terminating the search.
c-S	Repeats the search.
c-R	Reverses the search to search backwards.

If c-S or c-R is the first character typed, the previous search string is used again as the default. Entering any other command character terminates the search (and then executes that command).

Reverse Incremental Search

c-R, Reverse Incremental Search, works exactly the same way as c-S, Incremental Search, except that it searches *backward* towards the top of the buffer from point, instead of forward.

c-R Reverse Incremental Search

Searches for a character string while you type it, searching backward to the beginning of the buffer. It prompts for a string in the echo area with Reverse I-Search:. As you type characters in, c-R displays the accumulating string in the echo area and searches for it at the same time. If no string is found, it displays Failing Reverse I-Search:. When it locates the string, it puts the cursor in front of it so that repeated c-Rs locate previous occurrences of the default string in the buffer.

RUBOUT	Removes a character and backs up the search to the last match.

ESCAPE	When typed before any search characters, switches to Reverse String Search. See the section "Zmacs String Search".
END or ESCAPE	Exits the search.
c-G	Exits the search and returns to original starting point in the buffer.
c-Q	Quotes the next character, to prevent it from terminating the search.
c-S	Reverses the search to search forward.
c-R	Repeats the search.

If c-S or c-R is the first character typed, the previous search string is used again as the default. Entering any other command character terminates the search (and then executes that command).

String Search

The string search command, invoked by c-S ESCAPE, lets you type in the entire string and specify the direction in which to search before starting the search.

c-S ESCAPE String Search

Searches for a specified string, according to the arguments given with the special characters below. Another c-S always begins the search. It prompts in the echo area String Search:. It saves previous string search commands on a ring, retrievable with c-D.

The ring contains three elements by default and can be rotated with repeated c-Ds. The number of elements is controlled by the value of the Zwei variable Search Ring Max (**zwei:*search-ring-max***). You can change it via the m-X Set Variable command or as follows:

```
(zwei:set-zwei-variable "Search Ring Max" n)
```

While you are entering the search string, the following characters have special meanings:

c-B	Searches forward from the beginning of the buffer.
c-E	Searches backwards from the end of the buffer.
c-F	Leaves point at the top of the window, if the window must be recentered.
c-G	Aborts the search.
c-D	Gets a string to search for from the ring of previous search strings.
c-L	Redisplays the typein line.
c-M	Appends any marked text in the buffer to the search string.

c-Q	Quotes the next character.
c-R	Reverses the direction of the search.
c-S	Does the search, then comes back to the search command loop.
c-U or CLEAR INPUT	Erases all characters typed so far.
c-V	Delimited Search: Searches for occurrences of the string surrounded by delimiters.
c-W	Word Search: Searches for words in this sequence regardless of intervening punctuation, white space, newlines, and other delimiters.
c-Y	Appends the string on top of the search ring to the search string.
RUBOUT	Rubs out the previous character typed.
END or ESCAPE	Does the search and exits.
c-Middle	Searches for what the mouse is pointing at.

If you search for an empty string, it uses the default. Otherwise, the string you type becomes the default, and the default is saved unless it is a single character.

Locating and Replacing Strings Automatically

c-?, Replace String, searches forward for a string and replaces that string with another. c-? prompts for the string to be replaced, reads the string from the minibuffer, and then reads the replacement string. After it goes through the buffer trying to make the replacements, it tells you how many replacements it made (*1. replacements.*), or that it made none (Point pushed. Query replace done) .

You can also substitute one string for another *selectively* throughout the buffer, with m-?, Query Replace. m-? prompts first for the string to be replaced (Query-replace some occurrences of:), and then for the string to replace it with (Query-replace some occurrences of "string" with:). Terminate each string you specify with RETURN. m-? locates each occurrence and lets you decide what to do about each one.

Making Global Replacements in Zmacs

c-? Replace String

Replace String (m-X)

Replaces all occurrences of a given string with another, where the string can be characters, words, or phrases. It prompts first for the string to remove and second for the string to replace it with. A numeric argument *n* means to make *n* replace-

ments. If you do not specify a region, it begins at point and replaces all occurrences of the first string that occur *after* point in the buffer. Usually it attempts to match the case of the replacements with the case of the string being replaced. This behavior is controlled by the Zmacs variable Case Replace P (default **t**). When it is null, case matching does not take place. (For descriptions of Zmacs variables,

see the section "How to Specify Zmacs Variable Settings".)

Querying While Making Global Replacements in Zmacs

m-Z Query Replace

Query Replace (m-X)

If you do not specify a region, this command starts at point and replaces a string through the rest of the buffer, asking about each occurrence, where the string can be characters, words, or phrases. It prompts for each string. You first give it STRING1, then STRING2, and it finds the first STRING1, displaying it in context. You respond with one of the following characters:

SPACE	Replaces this STRING1 with STRING2 and shows next STRING1.
RUBOUT	Leaves this STRING1, but shows next STRING1.
,	Replaces this STRING1 and shows result, waiting for a SPACE, c-R, or ESCAPE.
Period	Replaces this STRING1 and ends query replace.
c-G	Leaves this occurrence of STRING1 unchanged and terminates the query replace.
ESCAPE	Same as c-G.
^	Returns to site of previous STRING1.
c-W	Kills this STRING1 and enters recursive edit.
c-R	Enters recursive editing mode immediately. Press END to return to Query Replace.
c-L	Redisplays screen.
!	Replaces all remaining STRING1s without asking.

Entering any other character terminates the command. Usually the command attempts to match the case of the replacements with the case of the string being replaced. This behavior is controlled by the Zmacs variable **zwei:*case-replace-p*** (default **t**). When it is null, case matching does not take place. (For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".)

If you give a numeric argument, it does not consider STRING1s that are not bounded on both sides by delimiter characters.

Querying While Making Multiple Global Replacements

You can specify single or multiple global replacements. While doing multiple query replacements, you can specify the replacement strings either from the minibuffer or from another buffer altogether.

Multiple Query Replace (m-X)

Performs query replace using many pairs of strings at the same time, where the strings can be characters, words, or phrases. (See the section "Querying While Making Global Replacements in Zmacs".) Strings are read in alternate minibuffers; when you finish entering all strings, press RETURN twice. An argument means that the strings must be surrounded by delimiter characters. A negative argument means that the strings must be delimited Lisp objects (not lists), rather than words.

Multiple Query Replace From Buffer (m-X)

Performs query replace using many pairs of strings *supplied from the specified buffer*. (See the section "Querying While Making Global Replacements in Zmacs".) The current buffer should contain a STRING1, a space, and a STRING2. Put quotation marks around any string that contains a space, tab, backspace, semicolon, or new-line character. Lines in the buffer that begin with a semicolon or are blank are ignored. In other words, each string in the buffer is a Lisp string, but quotation marks can be omitted if the string contains no special characters.

Other Types of Replacement Operations in Zmacs

Besides making string replacements in text, Zmacs commands replace:

- Tabs with spaces, and vice versa
- A region into the kill history
- Evaluated code into the buffer
- The value of LET into its variable
- A string for delimited Lisp objects (not lists or **nil**)

Tabify

Tabify (m-X)

Replace as many space characters in the buffer as possible with equivalent tabs. If a region is marked, this command tabifies the region.

Untabify

Untabify (m-X)

Replace all tab characters in the buffer as possible with equivalent space characters. If a region is marked, this command untabifies the region.

Query Replace Last Kill

Query Replace Last Kill (m-X)

Replaces the first item in the kill history with the region.

Evaluate And Replace Into Buffer

Evaluate And Replace Into Buffer (m-X)

Evaluates the Lisp object following point in the buffer and replaces it with its result.

Query Replace LET Binding

Query Replace Let Binding (m-X)

Replaces variable of LET with its value. Point must be after or within the binding to be modified.

Atom Query Replace

Atom Query Replace (m-X)

Performs query replace for delimited Lisp objects (except for lists or **nil**). (See the section "Querying While Making Global Replacements in Zmacs".)

Tag Tables and Search Domains

Tag tables, a means of global searching and replacing, allow you to make sweeping changes to groups of files without having to explicitly locate each file. A tag *table* is a temporary Zmacs support buffer that contains the names of sets of buffers and files, which you specify. You can edit these specified buffers and files as a unit, once you have specified them as items in a tag table. Tag tables remain active for the duration of the Zmacs session; you cannot permanently save tag tables.

You can use tag tables, for example, to:

- Search for all references to a certain variable and alter them consistently
- Search for all occurrences of an obsolete term and update it
- Search for all functions that give a certain error message

How Tag Tables Work

First, you specify the buffers or files that will make up the tag table. See the section "Specifying and Listing Tag Tables". Then you can perform an operation. See the section "Performing Operations with Tag Tables". Zmacs performs the operation on the files within the tag table that you have specified.

Example

Suppose you want to perform a tag query replace in several files. Use Tags Query Replace (m-X) to begin, see the section "Performing Operations with Tag Tables". The minibuffer prompts as in Query Replace (m-X) for the string to be replaced and the replacement string. The operation begins and Zmacs displays `Control-. is now Continue query replacement of "string-old" with "string-new"`; as it displays each occurrence, you deal with each one using the appropriate response characters. Tags Query Replace goes through all the files specified in the tag table, listing their names in the minibuffer and stopping at each occurrence of `"string-old"`. When it finishes searching all the files, it displays `No more files.`

Specifying and Listing Tag Tables

Select All Buffers As Tag Table (m-X)

Selects all existing file buffers. With a numeric argument, prompts for a string and only buffers whose name contains that string are considered. It creates a support buffer called *Tag-Table-*N*, which contains a list of the names of all the buffers. See the section "Support Buffers".

Select Some Buffers As Tag Table (m-X)

Selects some existing file buffers, querying about each one. With a numeric argument, prompts for a string and only buffers whose name contains that string are considered.

Select Some Files As Tag Table (m-X)

Creates a support buffer with the names of the selected files.

Select Tag Table (m-X)

Makes a tag table current for commands like tag search. It prompts in the minibuffer for the name of the tag table to use.

Select System As Tag Table (m-X)

Creates a tag table for all files in a system (and its subsystems) defined by the system definition. It prompts in the minibuffer for the name of a system — press HELP to see a display of system names. It selects the system but does not read the files in (use Find Files in Tag Table (m-X) to read them in).

Giving a numeric argument (c-U) to Select System As Tag Table includes all component systems of the system in the tag table.

List Tag Tables (m-X)

Lists in the typeout window the names of all the tag tables, and for each one shows the files it contains.

Performing Operations with Tag Tables

Tags Search (m-X)

Searches for the specified string within files of the tag table. It prompts in the minibuffer for the search string. If there is no current tag table, it prompts for one.

Zmacs displays in the echo area the name of each of the files in the tag table as it searches each file for the specified string. As Zmacs begins the operation and finds the first occurrence, it displays `Point pushed.` in the minibuffer and moves the cursor to the occurrence. After you deal with that occurrence, use c-., the Next Possibility command, to locate the next occurrence. (See the section "Displaying the Next Possibility".) Go through the specified files using c-. to the end.

Tags Query Replace (m-X)

You can also specify a region for these to operate on, asking about each occurrence. It prompts first for the string to remove and second for the string to replace it with. You first give it STRING1, then STRING2, and it finds the first STRING1, displaying it in context. You respond with one of the following characters:

SPACE	Replaces this STRING1 with STRING2 and shows next STRING1.
RUBOUT	Does not replace this occurrence, but shows next STRING1.
,	Replaces this STRING1 and shows result, waiting for a SPACE, c-R, or ESCAPE.
Period	Replaces this STRING1 and terminates the query replace.
c-G	Leaves this occurrence of STRING1 unchanged and terminates the query replace.
ESCAPE	Same as c-G.
^	Returns to site of previous STRING1 (actually, pops the point-pdl).
c-W	Kills this STRING1 and enters recursive edit.
c-R	Enters recursive editing mode immediately. Press END to return to Query Replace.
c-L	Redisplays screen.
!	Replaces all remaining STRING1s without asking.

Entering any other command character terminates the command. Usually the command attempts to match the case of the replacements with the case of the string being replaced. This behavior is controlled by the Zmacs variable Case Replace P (default **t**). When it is null, case matching does not take place. (For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".)

If you give a numeric argument, it does not consider STRING1s that are not bounded on both sides by delimiter characters.

Tags Multiple Query Replace (m-X)

Performs tags query replace using many pairs of strings at the same time, where the strings can be characters, words, or phrases. Strings are read in alternate minibuffers; when you finish entering all strings, press RETURN twice. An argument means that the strings must be surrounded by delimiter characters. A negative argument means that the strings must be delimited Lisp objects (excluding lists and **nil**), rather than words.

Tags Multiple Query Replace From Buffer (m-X)

Replaces occurrences of any number of strings with other strings within the tag table files, asking about each change. The current buffer should contain a STRING1, a space, and a STRING2. Put quotation marks around any string that contains a space, tab, backspace, semicolon, or newline character. Lines in the buffer that begin with a semicolon or are blank are ignored. In other words, each string in the buffer is a Lisp string, but quotation marks can be omitted if the string contains no special characters.

A positive numeric argument means to consider only the cases where the strings to replace occur as a word (rather than within a word). A negative numeric argument means to consider only delimited Lisp objects (excluding lists and **nil**), rather than words.

This command has the same options as Tags Query Replace.

Find Files in Tag Table (m-X)

Reads every file in the selected tag table into the editor. If there is no current tag table, it prompts for the name of one, which you can specify as a file (F), all editor buffers (B), or a system (S).

Visit Tag Table (m-X)

Creates a tag table by reading in a PDP-10 EMACS tag *file*. Tag files provide a list of the names of files that belong together as part of a system and a list of names and locations of definitions within the files. The file names are made into a tag table; the definition names are added to the completion table.

First, it reads in the specified tag file. It prompts for a file name from the minibuffer. Next, it goes through the tag file and marks the name of each tag as being a possible section of its file. The Edit Definition command (m-.) uses these marks to figure out which file to use.

It uses a support buffer to hold the elements of the tag table and another support buffer to hold the state of a pending operation involving all the files in the tag table. See the section "Support Buffers". Each contains the names of the files.

Support Buffers

Zmacs creates *support buffers* to save lists that it creates as part of the execution of some commands:

- Tag table commands.
- Edit Buffers (m-X).
- Source Comare (m-X).
- Lists for Edit Definition (m-.), when more than one definition exists.
- Buffers for Dired (m-X).
- Everything that edits a sequence of definitions, as in List Callers (m-X) or List Methods (m-X).

This means that you can examine the buffers containing the lists even after you have done some editing.

c-X c-B, the List Buffers command, displays these support buffers in the listing of buffers. Their names are, for example, *Definitions-1*, *Tags-Search-1*, and *Tags-Query-Replace-1*.

To avoid proliferation of editor buffers, Zmacs reuses support buffers in most cases, so that it usually saves no more than two of each type of support buffer at a time.

Possibility Buffers

Each time you use a command that generates a set of possibilities (for example, Tags Search (m-X) and Tags Query Replace (m-X)), it creates a possibility buffer for that set and pushes the set of possibilities onto a stack. c-., Next Possibility, extracts the next item from the set at the top of the stack. The set is popped from the stack when no more items remain in it. Several informational messages are associated with this facility. When the whole possibilities stack is empty and you have nothing more pending it displays:

```
No more sets of possibilities.
```

Displaying the Next Possibility

c-. Next Possibility

Selects the next possibility for the current set of possibilities. With a negative argument, pops off a set of possibilities. An argument of c-U or any positive number displays the remaining possibilities in the current set. With an argument of 0, selects the current buffer of possibilities.

For a description of the Edit Definition and Edit Callers commands, see the section "Editing Lisp Programs in Zmacs".

Example

Suppose you had been using c-. to move through the set provided by Tags Search and you then used Tags Query Replace to push a new set of possibilities onto the stack. When you finished the set provided by Tags Query Replace, you would see a message like the following to notify you that the empty set had been popped off the stack and the set of possibilities for Tags Search had been reinstated:

```
c-. is now Search for next occurrence of "string"
```

The position of point in the support buffer indicates the next item for Next Possibility (c-.). You can select the support buffer and move point manually in order to skip or redo possibilities.

Typing c-. while in a support buffer that is not at the top of the possibilities stack moves it to the top, prints an appropriate message, then takes the next possibility from that support buffer.

Sorting

The Zmacs sorting commands alphabetically sort a region by line, paragraph, or whatever *sort key* you specify.

Zmacs Sorting Commands

Sort Lines (m-X)

Sorts the region alphabetically by lines.

Sort Paragraphs (m-X)

Sorts the region alphabetically by paragraphs.

Sort Via Keyboard Macros (m-X)

Sorts the region, prompting for actions to define the *records* (the units of the region to be rearranged) and the sort keys (the fields in the records that are compared alphabetically to determine the new order of records). It prompts you to define the records and sort keys by performing positioning commands. It prompts for three actions:

1. Move to the beginning of the sort key (that is, move the cursor to the beginning of the field upon which to sort).

2. Move to the end of the sort key (that is, move to the end of the sort field).

3. Move to the end of the sort record (that is, move to the end of the record containing that field).

For each, it records the keystrokes that you use (as keyboard macros) and plays those back to find and sort the records in the region.

Changing Case and Indentation in Zmacs

Changing Case

Zmacs offers extended commands that convert the case of the code for words, regions, and buffers.

Changing Case of Words

`m-C` Uppercase Initial

Puts next word in lowercase, but capitalizes initial character. With an argument, it capitalizes that many words.

`m-L` Lowercase Word

Puts next word in lowercase. With an argument, it puts that many words in lowercase.

`m-U` Uppercase Word

Puts next word in uppercase. With an argument, it puts that many words in uppercase.

Changing Case of Regions

`c-X c-U` Uppercase Region

Uppercases the region.

`c-X c-L` Lowercase Region

Lowercases the region.

Uppercase Code in Region (`m-X`)

Converts all code (not comments, strings, or quoted characters) to uppercase. This gives the same effect as retyping that text while in Electric Shift Lock Mode. It operates on the region if there is one, otherwise it operates on the current definition.

Lowercase Code in Region (`m-X`)

Converts all code (not comments, strings, or quoted characters) to lowercase. It operates on the region if there is one, otherwise it operates on the current definition.

Changing Case of Buffers

Uppercase Code in Buffer (m-X)

Converts all code (not comments, strings, or quoted characters) to uppercase. This gives the same effect as retyping that text while in Electric Shift Lock Mode. It queries for a buffer name (the default is the current buffer) and operates on that buffer.

Lowercase Code in Buffer (m-X)

Converts all code (not comments, strings, or quoted characters) to lowercase. It queries for a buffer name (the default is the current buffer) and operates on that buffer.

Indentation

Indentation makes Lisp programs readable. Proper indentation reflects the structure of a program. Expressions should be indented so that their subforms are easily identifiable, and so that a function can be related to its arguments by eye, without counting parentheses.

The indentation commands work in any Zmacs major mode. However, the TAB key indents differently depending on the mode.

You can control the way Zmacs indents language forms. This is useful when you write macros, which parse their arguments differently from functions. See the section "Controlling Indentation of Lisp Forms in Zmacs".

Indenting Current Line

TAB

In Lisp mode, the TAB key indents the current line of Lisp code correctly with respect to the line above it. (In most other modes, TAB inserts a Tab character.) Point remains fixed with respect to the code.

With a numeric argument *n*, it indents the next *n* lines including the current one, and leaves point at the *n*+1st line.

c-TAB — Indent Differently

Tries to indent this line differently. If called repeatedly, it makes multiple attempts.

m-TAB — Insert Tab

Inserts a Tab character, even in Lisp Mode, in the buffer at point.

c-m-TAB — Indent For Lisp

Indents this line to make indented Lisp code, even in a mode other than Lisp Mode. A numeric argument specifies the number of lines to indent.

Indenting New Line

The keystroke combination RETURN TAB gets you into the right position to start typing the next line of code. LINE is the abbreviation for that combination.

LINE — Indent New Line

If the next two lines are blank, goes to the next line; otherwise, it creates a new blank line following the current one. In any case, it does a TAB on that blank line.

Reindenting Expression

c-m-Q — Indent Sexp

Corrects the indentation of the expression following point by adjusting the amount of space before each line in the expression. Position point in front of the incorrectly indented expression and press c-m-Q. This does not affect the indentation of the current line, but only fixes the indentation of following lines with respect to the current line. Use after modifying an expression.

With a numeric argument of *n*, it fixes the indentation of the next *n* expressions.

Indenting Region

c-m-\ — Indent Region

Indents each line in the region. With no argument, it calls the current Tab command to indent. With an argument of *n*, it indents each line *n* spaces in the current font.

Indenting Region Uniformly

c-X TAB — Indent Rigidly
c-X c-I

Shifts text in the region sideways as a unit. The default is to shift text one space (in the current character style) to the right. A numeric argument (positive or negative) shifts the region that many spaces to the right or left, respectively.

Aligning Indentation

Indent Under (c-m-X)

Aligns indentation under *string*. You specify *string* by clicking on it or typing it in the minibuffer.

When you use the mouse to specify the alignment string, begin by putting the cursor on the line you want to indent, then click right, click on Indent Under, then either point the cursor (a down-arrow pointing at a box) at a character that you want to line up with and click left, or type in a string for which it searches.

When you type the alignment string in the minibuffer, it searches back, line by line, forward in each line, for a string that matches the one read and that is farther to the right than the cursor already is. It indents to align with the string found, removing any previous indentation first.

Going Back to First Indented Character

m-M — Back To Indentation
c-m-M
m-RETURN
c-m-RETURN

Positions point before the first nonblank character on the current line.

Deleting Indentation

m-^ — Delete Indentation
c-m-^

Deletes the newline character and any indentation at the beginning of the current line. It tacks the current line onto the end of the previous line, leaving one space between them when appropriate, for example, at the beginning of a sentence.

With any numeric argument, it moves down a line first, killing the whitespace at the end of the current line.

New Line with This Indentation

m-O — This Indentation

Makes a new line after the current one, deducing the new line's indentation from point's position on the current line. If point is to the left of the first nonblank character on the current line, it indents the new line exactly like the current one. But if point is to the right of the first nonblank character, it indents the new line to the current position of point. Regardless, it leaves point at the end of the newly created line.

With a numeric argument, the new line is always indented like the current one, no matter where point is. With an argument of zero, it indents current line to point.

Moving Rest of Line Down

c-m-O — Split Line

Moves rest of current line down one line. It inserts a carriage return and indents new line directly beneath point. With a numeric argument *n*, it moves down *n* lines.

Inserting Blank Line

c-O Make Room

Inserts a blank line after point. With a numeric argument *n*, it inserts *n* blank lines.

Deleting Blank Line

c-X c-O Delete Blank Lines

Deletes any blank lines around the end of the current line.

Centering the Current Line

m-S Center Line

Centers the text of the current line within the line. With an argument *n*, it centers *n* lines and moves past them. Do not use this command for indenting Lisp code.

Controlling Indentation of Lisp Forms

This section shows you how to control the way that Zmacs indents Lisp forms when you use LINE, TAB, c-m-\, c-m-Q and so forth. This information is most useful when you are creating your own macros, which parse their arguments differently from functions.

This information applies to Zmacs in Lisp Mode only.

In indenting forms, Zmacs makes no distinction between functions, macros, or just data. Indentation is controlled by a property of the first symbol of the form. For convenience, this discussion refers to these symbols as functions if the technique applies to symbols of any type and as macros when the technique applies to macros only.

There are four methods of controlling indentation:

1. Start the name of your function with **def**...
2. Use **&body** in your macro's argument list
3. Use the **zwei:indentation** declaration
4. Use the **zwei:defindentation** special form

All four methods can be used to control the indentation of macros. The **def...** method and the **zwei:defindentation** method can be used to control the indentation of other forms as well. If you use none of these methods, forms beginning with your symbol will indent like ordinary functions.

It should be noted that Zmacs will not know that you have used **zwei:indentation** or **&body** until you compile your **defmacro** or evaluate the **zwei:defindentation**.

This discussion of indentation uses some special terminology.

For instance, *indentation n* means that Zmacs indents the first character of that argument *n* characters to the right of the first character of the function name. Thus, if Zmacs indents a form like this

```
.... (elbow-macro
      charm
       "grace"
        (beauty))
```

then the argument `charm` is getting indentation 0, the argument `"grace"` is getting indentation 1, and the argument `(beauty)` is getting indentation 2.

Standard indentation refers to the indentation behavior of elements of normal lists. The first form on a line indents to the same column as the first argument on the previous line which belongs to the same function. When the first argument of a function is not on the same line as the function name, it gets indentation 1. Here are some examples of standard indentation:

```
(list 0
      1
      2)

(list
  0
  1
  2)

(list 0 1
      2 3 4)
```

Controlling Indentation by Naming Your Function def...

If the name of your function begins with **def...**, argument 2 (the "third" argument) is given indentation 1, and all other arguments get standard indentation, like **defun**.

For example:

```
(defmacro define-thing (name args thing-maker thing-destroyer)
  `(progn (defun (:property ,name maker) ,args ,thing-maker)
          (defun (:property ,name destroyer) ,args ,thing-destroyer)))

(define-thing sylon
              (x y z)
  (...)
  (...))
```

This indentation behavior is overridden by use of **&body**, **zwei:indentation**, or **zwei:defindentation**.

Controlling Indentation Using &body

When you use **&body** in the argument list of a macro, the first **&body** argument gets an indentation of 1. All other arguments get standard indentation.

For example:

```
(defmacro macro-with-body-arg (a b &body body)
  `(list ,a ,b ,@body))

(macro-with-body-arg 0
                     1
  2
  3)
```

This indentation behavior is overridden by use of **zwei:indentation** or **zwei:defindentation**.

&rest and **&optional** have no effect on indentation.

Controlling Indentation Using zwei:indentation

zwei:indentation *Declaration*

zwei:indentation *subform-index indentation subform-index indentation...* declaration

zwei:indentation . *indentor-function* declaration

The **zwei:indentation** declaration (and the **zwei:defindentation** special form) give you the most control over the way Zmacs indents calls to your macros.

zwei:indentation is placed in the declaration part of the **defmacro** which defines your macro. When using the first syntax of **zwei:indentation**, the declaration should be given an even number of arguments. Each pair of arguments assigns a specific indentation to a particular argument. Note that, for the purposes of this declaration, *the arguments of your macro are numbered from 0.*

Here is an example using **zwei:indentation**:

```
(defmacro strangely-indented-macro (arg0 arg1 arg2 arg3 &rest rest)
  (declare (zwei:indentation 2 3 5 0 6 2))
  `(list ,arg0 ,arg1 ,arg2 ,arg3 ,@rest))
```

This causes argument 2 to get indentation 3, and argument 5 to get indentation 0. All other arguments get standard indentation, which causes argument 1 to follow argument 0, argument 3 to follow argument 2, etc.

Here is how this example macro is indented:

```
;0123456  <- indentation
(strangely-indented-macro 0
                          1
    2
    3
    4
 5
   6
   7
   8
   9)
```

zwei:indentation has a second syntax. When the **cdr** of the declaration is a function or a symbol with a function definition, that function is called to do the indenting.

For example,

```
(declare (zwei:indentation . zwei:indent-prog))
```

would cause the macro to indent like **prog**.

In addition to **zwei:indent-prog**, you could also use **zwei:indent-prog-or-tagbody, zwei:indent-tagbody,** or **zwei:indent-loop.** There is no further documentation on these function definitions.

Controlling Indentation Using zwei:defindentation

zwei:defindentation *Special Form*

zwei:defindentation (*name subform-index indentation subform-index indentation...*)

zwei:defindentation (*name . indentor-function*)

zwei:defindentation is similar to **zwei:indentation** in that it allows you to control the indentation of certain forms. The *subform-index*, *indentation* and *indentor-function* arguments work in the same way as for **zwei:indentation.**

zwei:defindentation differs from **zwei:indentation** in two ways:

- **zwei:defindentation** can be used to control the indentation of forms beginning with any symbol, not just macros. The *name* argument specifies the symbol whose indentation you wish to control.

- **zwei:defindentation** is a special form which must be compiled or evaluated after the definition of your macro, function, or other symbol.

When defining a macro, you will probably find **zwei:indentation** more convenient than **zwei:defindentation** , because there will be one less definition in your source file, and because recompiling your macro will undo any previous indentation specifications for that macro.

This example shows how to use **zwei:defindentation** to control the indentation of a function:

```
(defun funny-function (&rest rest)
  (length rest))

(zwei:defindentation (funny-function 3 5))

(funny-function 0
                1
                2
      3
      4)
```

Using Word Abbreviations in Zmacs

In Word Abbrev Mode, you can use word abbreviations, typing short abbreviations in an editing buffer to be expanded into text blocks of any length. Thus, you can substitute short, easily typed made-up words and have commonly used words, phrases, paragraphs, or program elements appear in their place.

Word Abbrev Mode is a Zmacs minor mode. Turn it on with the m-X Word Abbrev Mode command. Turn it off with another m-X Word Abbrev Mode command. When Word Abbrev Mode is on, you see the word Abbrev in the mode line.

Here is an example of using Word Abbrev Mode. Go to an editing buffer. For this example, make it a text buffer. Enter the m-X Word Abbrev Mode command. Type a word into the buffer, such as "Tex". Press c-X c-A. You are prompted in the minibuffer, Text mode abbrev for "Tex":. Type "t1" in the minibuffer. Now type "t1" and press the space bar. The word "Tex" appears in the buffer. From now on in this session, any time you type "t1", "Tex" will appear. If you want to have "t1" appear in the buffer instead of "Tex", type c-X U to Undo the last word abbreviation.

The expansion is triggered by a space, RETURN, or any punctuation mark, including asterisks, ampersands, and other symbols on the top row of the keyboard. The trigger is also inserted into the buffer.

If you want to have an abbreviation expand to more than one word, you can either select a region before typing c-X c-A or use a numerical argument with c-X for the number of words you want to include in the abbreviation.

c-X c-A makes word abbreviations only for the current Zmacs major mode. Using c-X c-A in a Lisp buffer makes abbreviations that work in all Lisp Mode Buffers. c-X c-A in a Text Mode buffer makes abbreviations that work in all Text Mode buffers, and so forth. These are called *mode word abbrevs*. You can make *global word abbrevs* with c-X +. Global word abbrevs work in any buffer except those where a mode word abbrev is defined using the same abbreviation. That is, you could have "t1" with a global definition of "(FUNCALL STREAM :STRING-OUT STRING)" and still have "t1" expand to "Tex" in text buffers.

If you want to see a list of word abbreviations, use the m-X List Word Abbrevs command.

If you want to save a file of word abbreviations, use the m-X Write Word Abbrev File command. To use the word abbreviations in a file, use the m-X Read Word Abbrev File command.

You can edit the current word abbreviations with the m-X Edit Word Abbrevs command. You can put all your word abbreviations in a buffer with the m-X Insert Word Abbrevs command and then write out the buffer so you can have all your word abbreviations in a readable form. Word abbreviation files are written in a special format with the "qwabl" file type and are not readable.

To stop using word abbreviations, you can either turn the mode off with the m-X Word Abbrev Mode command, or use the m-X Kill All Word Abbrevs command, which eliminates all defined word abbreviations.

You have some control over capitalization in Word Abbrev Mode. For instance, with "wabv" as an abbreviation for "word abbreviation", "Wabv" expands to "Word abbreviation" and "WABV" expands to "WORD ABBREVIATION". Capitalization is also controlled by how you enter the word abbreviation in the first place.

Word abbreviations work in the minibuffer, so you can even abbreviate commands. If you always type "otehr" when you mean "other", then make "otehr" the abbreviation for "other". You can also use word abbreviation in programming, for writing reports, or in many other editing contexts.

For an alternate method of storing and inserting blocks of text, see the section "Saving and Inserting Regions in Registers".

Word Abbreviation Commands

The word abbreviation commands in this section are listed in alphabetical order.

Add Global Word Abbrev

c-X + Add Global Word Abbrev

Prompts for a global word abbreviation. The default is to make a global abbreviation for the word preceding point. With a numerical argument, the command makes a global abbreviation for that many words before the point. If a region is defined, the command makes a global abbreviation for the region.

Global abbreviations work in all buffers unless a mode word abbreviation is defined for the current buffer mode. See the section "Add Mode Word Abbrev".

Add Mode Word Abbrev

c-X c-A Add Mode Word Abbrev

Adds a mode word abbreviation. The default is to make a mode abbreviation for the word preceding point. With a numerical argument, the command makes a mode abbreviation for that many words before the point. If a region is defined, the command makes a mode abbreviation for the region.

Mode abbreviations work in all buffers of the same mode as they were created in. Lisp mode abbreviations work in all Lisp Mode buffers. Text mode abbreviations work in all Text Mode buffers, and so forth. Mode word abbreviations override global word abbreviations for buffers of the same mode. That is, you can have a global abbreviation that works in all buffers except Text Mode, and have the same abbreviation expand differently in Text Mode buffers. See the section "Add Global Word Abbrev".

For a prompting form of this command, see the section "Make Word Abbrev".

Edit Word Abbrevs

Edit Word Abbrevs (m-X)

Allows you to edit word abbreviations. Displays word abbreviations in a Word-Abbrev buffer that you can edit in the usual fashion.

Insert Word Abbrevs

Insert Word Abbrevs (m-X)

Inserts a list of word abbreviations and their expansions into the buffer. You can make a file of this buffer so you will have a readable list of the word abbreviations you are using. Regular word abbreviation files (.qwabl type) are not readable.

Kill All Word Abbrevs

Kill All Word Abbrevs (m-X)

Eliminates all word abbreviations, whether read in from a file or created interactively with c-X commands.

List Some Word Abbrevs

List Some Word Abbrevs (m-X)

Lists word abbreviations or expansions that contain a given string. Prompts for the string. If you have "t1" as an abbreviation for "Tex", either "1" or "ex" will list the abbreviation.

The command m-X List Word Abbrevs, lists all abbreviations and expansions. See the section "List Word Abbrevs".

List Word Abbrevs

List Word Abbrevs (m-X)

Lists all word abbreviations and expansions.

Make Word Abbrev

Make Word Abbrev (m-X)

Prompts for and creates a new mode word abbreviation. Note that this command has the same effect as Add Mode Word Abbrev (c-X c-A), but prompts instead of picking up the word or phrase from text. It does not make a global abbreviation. See the section "Add Mode Word Abbrev".

See the section "Add Global Word Abbrev".

Read Word Abbrev File

Read Word Abbrev File (m-X)

Reads in a word abbreviation file created with Write Word Abbrev File (m-X). Abbreviations in the file override previously abbreviations, but you can add new abbreviations interactively.

Word abbreviation files are not in a readable form. They have the file type .qwabl.

Unexpand Last Word

c-X U Unexpand Last Word

Undoes the last expansion of a word abbreviation, leaving the unexpanded abbreviation in the buffer. Thus, if you have "t1" as an abbreviation for "Tex", but should, for some reason, want to have "t1" in your text, you could type "t1", resulting in "Tex", and then type c-X U, to get rid of "Tex" and leave "t1" behind.

Word Abbrev Mode

Word Abbrev Mode (m-X)

Turns on Word Abbrev Mode. If Word Abbrev Mode is already on, this command turns it off. Word Abbrev Mode allows you to define word abbreviations that expand as you type them. This command displays Abbrev in the mode line.

Write Word Abbrev File

Write Word Abbrev File (m-X)

Writes all the current word abbreviations to a word abbreviation file. Word abbreviation files are not in a readable form. They have the file type .qwabl.

Use the m-X Read Word Abbrev File command to read in a previously created file of abbreviations.

See the section "Read Word Abbrev File".

Using Character Styles in Zmacs

A number of Zmacs commands allow you to use different character styles. Using character styles, you can indicate program structure with different character styles, or you can do certain kinds of text formatting.

For another kind of text formatting, see the section "Formatting Text in Zmacs".

The information in this chapter deals with these commands only. For more information on character styles, see the section "Character Styles". For more information on using character styles in programs, see the section "Character Environment Facilities".

A number of choices of character styles are available on the Genera system. Here are a few examples:

Dutch.Bold-Italic.Small

Swiss.Bold-Condensed-Caps.Normal

Eurex.Italic.Huge

Dutch.Roman.Normal

Zmacs commands allow you to specify the character style for a character, word, region, or buffer. (Not all character styles work equally well on all printers.)

Character styles are identified by three characteristics that affect how the character appears. These are family, face, and size. The names of character styles incorporate these three characteristics. Thus, as in the example, you see the **family** Eurex, the **face** Italic, and the **size** Huge; this is expressed in commands as EUREX.ITALIC.HUGE. This is a *fully specified* character style.

Where you see a character style named as NIL.NIL.NIL (with *nils* in its name), this indicates that the character style is being merged against a default style. For instance, the default character style for Zmacs buffers is FIX.ROMAN.NORMAL. Thus, if you wish a character in a Zmacs buffer to be **bold**, it can be entered as NIL.BOLD.NIL. This means that in the Zmacs buffer, its style is merged against FIX.ROMAN.NORMAL to produce FIX.BOLD.NORMAL. In some other context, NIL.BOLD.NIL might be interpreted as SWISS.BOLD.NORMAL because it was merged against SWISS.ROMAN.NORMAL. A character style specification in the form NIL.BOLD.NIL is called a *character face* specification (in contrast to the fully qualified specification).

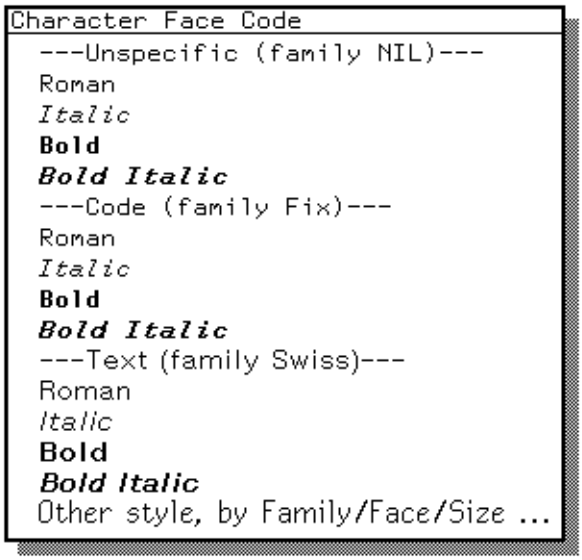
Here is an example of using character styles. In a Zmacs buffer, type a word. Move the cursor back to the beginning of the word. Press m-J. You are prompted

```
Change character style of word to [default NIL.NIL.NIL]:
```

Type in NIL.BOLD.NIL and press RETURN. The word you have selected appears in **bold**.

Now move to the beginning of another word. Press m-J. You see the same prompt. This time move the mouse over the word you just made bold. You'll see Mouse-L: NIL.BOLD.NIL in the mouse documentation line. Click Left. The word you selected appears in bold.

Now move to the beginning of another word. Press m-J. You see the same prompt. The arrow cursor points straight up and you'll see Mouse-R: Character style menu in the mouse documentation line. (If you don't see it, move the mouse a bit until it appears.) Click Right, and a menu appears.



Move the mouse over the first word **bold** and click any button. The word you selected appears in **bold**.

The menu allows you many choices of character styles. If you click on the item

```
Other style by Family/Face/Size ...
```

you get a series of menus that allows you to select the family, faces within that family, and styles available for that combination of family and face. Not all families have all faces and sizes available. Sizes are relative, for example, SMALL or SMALLER, not absolute.

These are the three ways of selecting a character style. All commands for changing character styles allow you to use these three methods of specifying the character style you want.

A common desire is to include a word or phrase in the same family, but **bold** or *italic* face. If you are typing in a normal Zmacs buffer (FIX.ROMAN.NORMAL) you can make a word bold by pressing m-J and then simply entering "bold" or just "b" in response to the prompt. This is specifying a character face.

There are six commands for changing character styles.

- c-J (Change Style Character) changes the style of a character, or several characters if you use a numeric argument.

- m-J (Change Style Word) changes the style of a word, or several words if you use a numeric argument.

- c-X c-J (Change Style Region) changes the style of a region.

- m-X Change One Style Region changes one style in a region, but not any other. Thus, if you had a region with both **bold** and *italic* in it, you could change the *italic* characters without affecting the **bold** ones.

- m-X Set Default Character Style sets the default character style for the whole buffer. The command also prompts to ask if you want to set the default character style in the attribute list as well.

- c-m-J (Change Typein Style) changes the character style for newly inserted characters.

There are two commands for getting information about character styles in a buffer.

- m-X Show Character Styles displays all character styles in the buffer or in the region, if there is one. The display includes the character style, that is, what you type in to select a character style; the Lisp name of the character style; the font equivalent; and samples of the character style. When samples are displayed, you can click on the samples to select that character style for any of the commands for changing character styles.

- m-X Find Character in Style searches forward for the next character in a given style. You supply the name of the character style as in the commands for changing or specifying character styles.

Character Style Commands in Zmacs

The character style commands in this section are listed in alphabetical order.

Change One Style Region

Change One Style Region (m-X)
Allows you to change one character style in a region without affecting other character styles in the region. This command prompts you for the style you want to change and the style to which you want it changed. You can identify the old and new styles by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

To change all the characters in a region to a single style, see the section "Change Style Region".

c-J Change Style Character

c-J Change Style Character
Changes the character style of a single character, or, with a numeric argument, more than one character. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Change Style Region

c-X c-J **Change Style Region**
Allows you to change the character style of a region to a single character style. That is, if there are two character styles in the region, both will be changed to the character style you choose with this command. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

To change only one character style in a region, see the section "Change One Style Region".

m-J Change Style Word

m-J Change Style Word
Changes the character style of a single word, or, with a numeric argument, more than one word. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

c-m-J Change Typein Style

c-m-J Change Typein Style
Sets the character style for newly inserted characters. It does not affect the default style for the buffer. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Find Character in Style

Find Character in Style (m-X)
Searches forward for the next character in a given style. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Set Default Character Style

Set Default Character Style (m-X)
: Sets or changes the character style associated with the buffer. You can identify the style you want by typing in the name, by clicking with the mouse on the style you wish, or by clicking through the character styles menu.

Show Character Styles

Show Character Styles (m-X)
This command displays all the character styles in a buffer. The display includes the character style, that is, what you type in to select a character style; the Lisp name of the character style; the font equivalent; and samples of the character style. When samples are displayed, you can click on the samples to select that character style for any of the commands for changing character styles.

Improved User Interface for Changing Character Styles in Zmacs

Genera now includes a new, optional user-interface mechanism for specifying a new character style, for all the Zwei commands that change the character style of text, such as c-J, m-J, c-m-J, and c-X c-J.

The new mechanism is an option that is selected according to the value of a new Zwei variable, **zwei:*change-style-mode***, or Change Style Mode. The variable has two possible values, Quick and Prompt For Name. Prompt For Name is the default, and selects the existing interface for specifying character styles in the minibuffer. You do not activate the new behavior until you specify Quick mode.

Previously, you had to type a line of keyboard input to the minibuffer to specify a character style. The new alternative (Quick) mode allows you to define your own one-character abbreviations for commonly used styles, which can then be specified by typing just one character at the prompt. It comes with an initial set of abbreviations (I for italic, B for bold, and so on). It provides both an escape mechanism to be prompted for a full character style and a mechanism for redefining existing abbreviations and defining new ones on the fly.

You set the variable by a Zmacs command: m-X Set Variable / Change Style Mode / Quick. If you want to establish this setting routinely in your init file, use this form:

```
(ZWEI:SET-ZWEI-VARIABLE "Change Style Mode" :QUICK)
```

You can also make your own standard abbreviations in your init file. For example, the following establishes T as an abbreviation for NIL.NIL.TINY:

```
(ZWEI:SET-CHARACTER-STYLE-DISPATCH #\T '(NIL NIL :TINY))
```

Typing any of the style-changing commands prompts for one character. Possibilities are:

Character	*Description*
c-G, ABORT, RUBOUT	Aborts
ESCAPE	Prompts for a style (like the old Zwei behavior)
RETURN or SPACE	Accept the default (from most recent change)
char	First time, asks for a definition and uses it; after that, uses the existing definition
m-*char*	(Re)Defines the action of *char*

You use **zwei:set-character-style-dispatch** to set the abbreviation defaults. The following definitions are predefined:

Abbrev.	*Style*
B	NIL.BOLD.NIL
I	NIL.ITALIC.NIL
P	NIL.BOLD-ITALIC.NIL
N	NIL.NIL.NIL
S	NIL.NIL.SMALLER
L	NIL.NIL.LARGER

The major incompatibility between Prompt for Name and Quick is that pressing m-J RETURN in Quick mode does *not* produce NIL.NIL.NIL; you must press m-J N to do that.

Another difference is that Quick mode makes it possible to use style-change commands in the minibuffer. This is not possible in Prompt for Name mode, because entering the minibuffer recursively is not allowed. Note that you do not see the prompt if the minibuffer is exposed.

Formatting Text

Producing formatted text requires two steps:

1. Entering the text and text formatting instructions. See the section "How to Use Text Formatting Commands".

2. Formatting that text with one of the Zmacs formatting commands. See the section "Using Zmacs Format Commands".

First you use the Zmacs editor to enter the text and embed formatting instructions. These instructions format the text by, for example, specifying fonts, creating bulleted lists, inserting headings, and describing formatting environments.

For example, to specify that you want to italicize a group of words, such as the title of a book, use the italicize environment. To emphasize a word, you might use the boldface environment.

This text:

```
@i(Gone With the Wind), by Margaret Mitchell, is a @b(great) book.
```

produces this, when formatted:

Gone With the Wind, by Margaret Mitchell, is a **great** book.

Formatting instructions all begin with an @. The "i" tells the formatter that you want the italicize environment, and the parentheses (*delimiters*) enclose the text within that environment. Other valid delimiters can be (), [], <>, {}, "", ”, or ‘’.

The extended commands Format Region (m-X), Format Buffer (m-X), and Format File (m-X) display text in a formatted style using formatting instructions that you embed in the text. You can send the formatted text to a supported printer by giving the Format command a numeric argument. This prompts for an output device.

Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Zmacs also supports a number of commands for using character styles, which allow text formatting of a somewhat different sort. See the section "Using Character Styles in Zmacs".

How to Use Text Formatting Commands

Formatting commands control the format of the text (such as blank spaces between lines, tab settings, line breaks) and whether the formatter centers the text or aligns it against one of the margins. You can also use formatting commands to specify particular formatting characteristics or environments.

For example:

```
@i(Gone With the Wind),@* by Margaret Mitchell @# is a @b(great) book.
```

produces:

Gone With the Wind,
by Margaret Mitchell is a **great** book.

The @* command forces a line break and the @# command leaves a blank em-space for a special character to be drawn in.

Some commands, such as the @* in the example, are complete by themselves. Others accept arguments, which must be enclosed in delimiters. There is no such thing as a long form for a command; you cannot say `@begin(blankspace)`, for example.

Example of Using Tabs to Format Text

This example shows how to use tab stops to:

- Divide text into four columns

- Center text
- Position text Flush right
- Reset tabs

```
@begin(format)
@tabdivide(4)
1.@\*@\*@\*
2.@=a@\@=b@\@=c@\@=d
3.@=e@=f@=g@=h
4.Left@=Center@>right
5.Left@=Center@>right@\
@tabclear()
6.Left@=Center@>right
@end(format)
```

produces:

```
1.               *               *               *
2.       a               b               c               d
3.       e               f               g               h
4.Left   Center                                              right
5.Left   Center               right
6.Left                             Center                    right
```

How to Create an Environment

You can create a Zmacs formatting environment in two ways. You can use use a short-form such as:

```
@i(italicize this)
```

or you can use a long-form environment (where the commands @begin(i) and @end(i) act as delimiters for the text that they enclose). The long form is particularly useful for specifying the format for an entire text passage.

For example,

```
@begin(i)
Environments can be either short form or long form.  The long
form uses the commands @@begin and @@end to act as delimiters
for the text that they enclose.
@end(i)
```

produces this:

Environments can be either short form or long form. The long form uses the commands @begin and @end to act as delimiters for the text that they enclose.

(The @s inside the environment must be doubled so the formatter does not interpret them as format commands.)

The following environment *enumerates*, that is, numbers sequentially each separate line of text within it:

```
@begin(enumerate)
Paragraph 1

Paragraph 2

Paragraph 3
@end(enumerate)
```

produces the following output:

1. Paragraph 1

2. Paragraph 2

3. Paragraph 3

Note that environments can be either filled or unfilled:

Filled	Fills each output line to capacity within the limits of the display.
Unfilled	Keeps output lines exactly as you entered them, as in an example.

Basic Text Formatting Commands and Environments

@+

Creates superscript displays of numbers or letters.

No documentation available for section + Environment Display.

@-

Creates subscript displays of numbers or letters.

No documentation available for section - Environment Display.

@B

Formats its contents using Facecode B, bold.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

This text is in B facecode.

@BI

Formats its contents using Facecode BI, bold italic.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

This text is in BI (bold italic) facecode.

@Blankspace

The Blankspace command leaves vertical blank space in the displayed record.

No documentation available for section Valid Vertical Distances.

@Box

Draws a box around the specified text. You can modify the LeftMargin and/or RightMargin attributes (respectively) to use relative document margins (a signed attribute value), or to use global margin settings (an unsigned attribute value).

No documentation available for section Box Environment Display.

@C

Formats its contents using SMALLCAPS.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

THIS TEXT IS IN C (SMALLCAPS) FACECODE.

@Caption

Specifies the caption for a figure or table. The "Tag Command" must follow this command to generate appropriate crossreference labels.

Figure ! shows a bug.

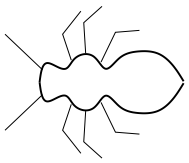


Figure 87. A Bug

@Center

An "unfilled environment" that causes each line within it to be centered in the displayed record.

@Checklist

A filled environment which produces a list marked by pointer icons on the screen. There are no markers when this environment is hardcopied.

Checklist is similar to the Itemize environment and Enumerate environment.

Using the default attributes, the Checklist environment generates a simple list.

No documentation available for section Checklist Environment Display.

@Commentary

Creates an environment for comments. The contents of this environment are not printed or displayed as part of the document.

@Description

Description is a "filled environment" that has a wider left margin after the first line. Because it resembles a two-column table, Description is often used to format lists of definitions. It is also useful for displaying paragraphs of text with headers.

You must insert a tab character to separate the header from the body of the paragraph. Press `s-TAB` or use the Tab-to-tab-stop command.

No documentation available for section Description Environment Display.

@Display

Display is an "unfilled environment" where each line in the Concordia file produces one line in the processed text. Both the left and the right margins are widened.

It is often used to show lists without tick-marks or numbers.

This text is in a Display environment.
This is a new line.

@Enumerate

Enumerate is a "filled environment" that produces a numbered list. It is very similar to the "Itemize Environment" and the "Checklist Environment".

Using the default attributes, the Enumerate environment generates a simple numbered list.

Nested Enumerate environments produce lists that are lettered and numbered, and so are commonly used to make outlines.

No documentation available for section Enumerate Environment Display.

@Equation

The Equation environment is used as a simple mathematical environment to produce numbered equations.

@Example

Example is an "unfilled environment" that creates a region with a wider margin and that uses a fixed-width font. It is designed to show examples of computer interaction.

Often, examples of actual programs contain lines that are too long to be printed on one line in the Example environment. Refer to the section "Longlines Attribute" for more information.

No documentation available for section Example Environment Display.

@F

Formats its contents using a facecode defined with the SpecialFont command.

@Figure

Allows you to specify a caption and a tag for a Figure. Figures are numbered. The caption appears in the List of Figures, generated automatically by the formatter, and the tag can be used with the "Ref Command" for crossreference purposes.

Figure ! shows a bug.

@Fileexample

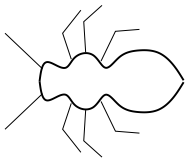


Figure 88. A Bug

An unfilled environment that has a wider margin and uses a fixed-width font. Fileexample, Inputexample, and Outputexample are Markups that are identical to Programexample.

No documentation available for section Fileexample Environment Display.

@Flushleft

Flushleft environment is an "unfilled environment" that causes each line within it to be aligned with the displayed record's left "global margin". The distance to the margin can be changed by altering the "Leftmargin Attribute".

No documentation available for section Flushleft Environment Display.

@Flushright

Flushright environment is an "unfilled environment" that causes each line within it to be aligned with the displayed record's right "global margin". The distance to the margin can be changed by altering the "Rightmargin Attribute".

No documentation available for section Flushright Environment Display.

@Foot

Inserts a numbered parenthetical note at the bottom of the formatted page. It does not create bottom-of-the-page footnotes or numbering online.

@Format

Format is an "unfilled environment" that produces text exactly as you type it using the current font, without moving margins or justifying text. It is used to show unusual formatting.

No documentation available for section Format Environment Display.

@FullPageFigure

Causes a figure to appear at the top of the next new page. Note that the formatter continues to fill the current page before placing the figure. Captions for FullPageFigures appear wherever you place the Caption command.

@FullPageTable

Causes a table to appear at the top of the next new page. Note that the formatter continues to fill the current page before placing the table. Captions for FullPageTables appear wherever you place the Caption command.

@G

Formats its contents using Facecode G, usually a Greek character set.

Note that creating this environment without a marked region changes the typein character style. (You can also use c-m-J to change the typein character style.)

Τηισ τεξτ ισ ιν Γ (Γρεεκ) φαχεχοδε.

@Group

Causes the text within the environment to be grouped together and to appear all on one page. If the grouping is too large to fit on the current page, a new page will be started, leaving white space at the bottom of the previous page. The Group environment should be used only when absolutely necessary. If the grouped text is more than a few lines, it is a good idea to identify one or more points where a page break would be acceptable and insert a Hinge command.

@Heading

Formats its contents in the same style as a first level heading (chapter) as defined in the book design. This heading is unnumbered.

This is in a Heading environment.

@Hinge

Indicates the places in a grouped environment where a new page can begin.

@I

Formats its contents using Facecode I, italic.

Note that creating this environment without a marked region changes the typein character style. (You can also use c-m-J to change the typein character style.)

This text is in I (Italics) facecode.

@Index

Makes the text given as the argument to this command an index entry.

@IndexPrimary

Makes the text given as the argument a primary index entry.

@IndexSecondary

Makes the text given as the argument a secondary index entry.

@InputExample

An unfilled environment that has a wider margin and uses a fixed-width font. Inputexample, Fileexample, and Outputexample are markups that are identical to Programexample.

No documentation available for section Inputexample Environment Display.

@Itemize

Itemize is a "filled environment" that produces a bulleted list. The items in the list are marked by bullets, or tick-marks. Separate each item by one blank line.

The Enumerate and Checklist environments are similar to Itemize.

No documentation available for section Itemize Environment Display.

@K

Formats its contents using Facecode K, a character set that looks like computer keyboard input.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

```
This text is in K (Keyboard) facecode.
```

@Label

Defines a crossreference label.

@Level

An environment for writing outlines. Level is similar to Enumerate, except that it uses the standard conventions for numbering outlines.

No documentation available for section Level Environment Display.

@LS

Formats its contents using Facecode LS, the same style as Lisp Objects.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

This text is in LS (Lisp object) facecode.

@M

Formats its contents using Facecode M, usually mathematics symbols.

```
This text is in M (Mathematical) facecode
```

@MajorHeading

Formats its contents in the same style as a Majorpart as defined by the book design. This command is retained for conversion compatibility. In general each section should be a record to provide good modularity for Document Examiner lookup.

This is in a MajorHeading environment.

@Multiple

Used inside Itemize, Enumerate, and Description environments to indicate that a series of paragraphs are all part of one item for purposes of formatting.

@NewPage

Begins a new page immediately. Its argument is an integer that controls the number of blank pages to leave (the default is 0).

@Note

Places the text given as an argument in a numbered footnote or endnote and leaves a numbered reference in the text.

@Outputexample

An unfilled environment that has a wider margin and uses a fixed-width font. Outputexample, Fileexample, and Inputexample are markups that are identical to Programexample.

```
This text is in an OutputExample environment.  Note the
wider margins.
```

@P

Formats its contents using Facecode P, bold italic.

Note that creating this environment without a marked region changes the typein character style. (You can also use c-m-J to change the typein character style.)

This text is in P (bold italic) facecode.

@PageFooting

Selects the text and format for the running text at the foot of a page or pages.

@PageHeading

Selects the text and format for the running text at the top of a page or pages.

@PageRef

Use with the Tag command for inserting page references.

@ProgramExample

ProgramExample is an unfilled environment which creates a region with a wider margin and which uses a fixed-width font. It is similar to the environments "Example Environment" and "Display Environment". It is designed to show examples of computer programs.

No documentation available for section Programexample Environment Display.

@Quotation

Formats its contents in a filled environment with left and right margins increased by one centimeter.

No documentation available for section Quotation Environment Display.

@R

Formats its contents using Facecode R, roman.

Note that creating this environment without a marked region changes the typein character style. (You can also use c-m-J to change the typein character style.)

This text is in R (Roman) facecode.

@Ref

Retrieves and prints the value associated with a counter used to count page numbers, figures, tables, chapters, sections, and appendixes.

@S

Formats its contents using Facecode S, usually a symbol character set.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

™ηισ τεξτ ισ ιν Σ ⟨Σψ—β•λ⟩ φαχεχ•δε°

@Set

Sets a counter to a value. You specify the counter as a keyword and then specify the value.

@SimpleTable

Formats its contents as a simple table. Enter the contents of the table with columns separated by tabs. Concordia calculates the tabstops for you based on the length of the entries. See the section "Creating Simple Tables".

@SimpleTableSpecs

Specifies horizontal and vertical rules for a simple table. See the section "Creating Simple Tables".

@String

Defines a text string to be used with the Value command.

You can use this command to set the specified Case selector variable to an ambient value. For more information, see the section "Setting Document Formatting Options in Symbolics Concordia".

@SubHeading

Formats its contents as a subheading, that is in the same style as a third level heading (subsection) is formatted as defined in the book design. This command is retained for conversion compatibility. In general each section should be a record to provide good modularity for Document Examiner lookup.

This is in a SubHeading environment.

@SubSection

Starts a new subsection in the document. This command is retained for conversion compatibility.

@SubSubHeading

Formats its contents as a subsubheading, that is in the same style as a fourth level heading (subsubsection) is formatted as defined by the book design. This command is retained for conversion compatibility. In general each section should be a record to provide good modularity for Document Examiner lookup.

This is in a SubSubHeading environment.

@SubSubSection

Starts a new subsubsection in the document. This command is retained for conversion compatibility.

@T

Formats its contents using Facecode T, a fixed width (typewriter) character set.

Note that creating this environment without a marked region changes the typein character style. (You can also use `c-m-J` to change the typein character style.)

```
This text is in T (Typewriter) facecode.
```

@TabClear

Eliminates any tabs that were previously set in a file. Usually used before any type of tab setting command, like "TabDivide Command" or "TabSet Command".

@TabDivide

Divides the formatting region into columns. Columns are of equal widths and sizes to fill the region horizontally. Text items are separated for formatting by using the "s-TAB Command" or by pressing `s-TAB`.

This command takes an integer as an argument.

Although it is not necessary to clear tabstops within an environment, doing so guarantees that the TabDivide command will work as you expect it to. It is a good work habit to put a TabClear command before each TabDivide command you use.

When equally sized columns are not desired, use the "TabSet Command".

Wrap tabbed text in a Format Environment to override any formatting defaults.

@Table

Allows you to specify a caption and a tag for a Table. Tables are numbered. The caption appears in the List of Tables, generated automatically by the formatter, and the tag can be used with the "Ref Command" for crossreference purposes.

@TabSet

Creates tabs within the text at distances you specify.

- Use a horizontal distance: number of characters, inches, or centimeters.
- Use the "s-TAB Command" or `s-TAB` to separate text items.
- Insert "TabClear Command" to clear previously set tabs.

It is a good work habit to put a TabClear command at the end of any environment where tabs are explicitly set. Clearing tabs that are no longer necessary, guarantees that the TabSet command will work as you expect it to.

Wrap tabbed text in a Format Environment to override any formatting defaults in the active environment.

@Tag

Specifies a *Codeword* (a string) as a crossreference label showing the position and number of an equation, theorem, figure, or table.

Used with the "Caption Command" to mark figures and tables.

Referenced by the "Ref Command".

@Text

It may seem strange to have a special environment called Text, because after all everything we type that is not in a special environment is processed just like stuff written in the text environment.

No documentation available for section Text Environment Display.

@Transparent

An environment that has no effect. It exists to permit you to adjust an attribute for a small part of a document without altering the design of the document as a whole. For example, you can scale a picture that is just inserted into text (not in a

figure or example environment) by adding the PictureScale attribute to a Transparent environment that you have wrapped around the picture.

@Unnumbered

Modifies environments that have the Numbered attribute so that paragraphs are unnumbered.

@Use

Takes an environment name as its argument. This definition the environment is included as part of the current environment.

@Value

Retrieves and prints the value of its argument, which should be a counter or a string defined by the String command.

You can also use the Value command to retrieve the value of "Sage System Variables".

@Verbatim

Verbatim produces everything you type exactly as you type it, without moving margins or justifying text. It is used to show unusual formatting.

Verbatim is an "unfilled environment" like "Format Environment" except that Verbatim uses a fixed-width font unlike regular text.

No documentation available for section Verbatim Environment Display.

@Verse

An filled environment for printing verse. Each line is treated as a paragraph, that is each line of verse is terminated by RETURN and if a line is too long for the linelength, it wraps with appropriate indentation on the continuation line.

No documentation available for section Verse Environment Display.

@W

Indicates that its contents are to be treated as a single word and should not be hyphenated.

These punctuation-character commands consist of an @ followed by one punctuation character. They take no arguments.

@# Leaves a blank space (quad space or em-space) for a special character.

@* Forces a line break.

@. Generates a period and forces a single significant space after it (used for abbreviations).

@=	Sets a tab at the left side of text to be centered. Do not use in a filled environment. Works with the tab commands (@\, @>, or @=).
@>	Sets a tab at the left side of text to be flushed right. Do not use in a filled environment. Works with the tab command (@\, @>, or @=).
@\	Moves the cursor to the next tab stop or marks the end of text being centered or flushed right. Do not use in a filled environment.
@^	Sets a tab at the current cursor position. Do not use in a filled environment.
@@	Inserts an @ in the text.
@~	Ignores all the white space between it and the next text in the source.

Using Zmacs Format Commands

The second (and final) step in formatting is to issue one of the formatting commands, which interprets the text and formatting instructions into the formatted text.

You can use the commands to format the text on your screen or on a printer. Check first on the screen before sending output to a printer.

Use a numeric argument to send the output to a supported printer. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Format Region

Format Region (m-X)

Displays the contents of the region formatted as a text environment. With a numeric argument, the command prompts for an output device. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Format Buffer

Format Buffer (m-X)

Displays the contents of the buffer, formatted as a text environment. With a numeric argument, the command prompts for an output device. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Format File

Format File (m-X)

Displays the contents of the file, formatted as a text environment. With a numeric argument, the command prompts for an output device. Page numbers are included by default on hardcopy output. You can turn off page-numbering with a numeric argument of 0.

Working with Regions

What is a Zmacs Region?

Many Zmacs commands deal with the region. A region consists of a block of information within the buffer that you want to manipulate as a single entity. You define the area of the region, which can be any size, from characters or chunks of code to pages or the entire buffer.

Zmacs keeps track of one or more locations in a buffer using buffer *pointers*. This section describes:

- The two buffer pointers named *point* and *mark*
- How Zmacs uses them to define the boundaries of a region
- The *point-pdl*, a ring of pointers to saved locations
- *Registers*, pointers to locations that you name and save
- The region-manipulating commands

Point and the Region

Point (shown by the cursor) is the most important buffer pointer. Most editor commands depend on the position of point. Many editor commands, invoked by either the mouse or the keyboard, can be used to position point to the desired location in the buffer. Point points to one end of the region.

Mark and the Region

Mark points to the other end of the region. To *mark* a piece of text means to position point and mark on either side of the text, making it the region. The simplest way to mark some text is to position point (using either the mouse or keystrokes) to one boundary (either the beginning or the end) of the text, set the mark there (using the Set Pop Mark command), and then reposition point at the other boundary. See the section "Setting/Popping the Mark".

Unlike point, the mark can be *active* or *inactive*. When mark is active, the region is shown on the screen by underlining. When mark is inactive, you cannot see it on the screen unless you reactivate it with c-X c-X. Although normally you cannot see an inactive mark, Zmacs keeps track of mark when it is inactive and sometimes uses mark in its inactive state. For example, c-Y leaves point and mark surrounding what it yanks, but does not activate mark. c-W immediately following c-Y kills the region even though it is not active. c-X c-X after c-Y activates mark, making the region visible. However, most commands will not use mark or the region unless it is active. You can set the mark three ways: when you create a region using the mouse, explicitly with the command Set Pop Mark (c-SPACE), or with one of the commands to mark regions. See the section "Commands to Mark Regions". When you set the mark, you activate it and make the region appear.

Creating a Region

You can create a region using either the mouse or keystrokes.

Creating a Region with the Mouse

The most common way to create a region is with the mouse. Hold down the left mouse button and drag the cursor. Release the button to mark the end of the region.

Holding down the middle mouse button creates a region, too. It marks the "thing" you point the mouse at, "thing" being mode-dependent (a word or Lisp expression if you point with the mouse at text, or a line if you point with the mouse at white space before or after all the text on the line).

Creating a Region With Keystrokes

You can also create a region using keystrokes. After setting the mark, you can move point either forward or backward to define a region in either direction; as you do so, Zmacs highlights the region with underlining.

Typing a self-inserting character or c-G deactivates the mark and removes the underlining that highlights the region. The mark does not have an associated cursor like point. When inactive, the mark is invisible, but you can go to it with c-X c-X, Swap Point And Mark.

The Point-pdl

Zmacs maintains a special stack of buffer pointers called the *point-pdl*, where *pdl* stands for *push-down list*, another name for a stack.

Zmacs automatically saves point on the point-pdl as it executes some commands (for example, m-<) that move point great distances. Whenever Zmacs pushes point onto the point-pdl, it displays "Point pushed" in the echo area, moves point to its new location, and pushes the previous point down onto the point-pdl.

By popping the point-pdl, that is, resetting point to its last location as recorded on the point-pdl, Zmacs returns point to where it was when the pdl was last pushed.

Setting/Popping the Mark

c-SPACE — Set Pop Mark

With no argument, c-SPACE does three things:

1. Puts mark where point is
2. Makes mark active
3. Pushes point onto the point-pdl

Other commands can do each of these operations separately. Creating a region with the mouse sets a mark and makes it active but does not push point.

This command does other things depending on how many c-Us are typed in front of it:

Argument	*Action Taken*
one c-U	Pops the location on the top of the point-pdl into point (typically puts point where it set the last mark).
two c-Us	Pops the location on the top of the point-pdl and throws it away.

Moving to Previous Points

c-m-SPACE — Move to Previous Point

Exchanges point and top of point-pdl. With a numeric argument *n*, it rotates a ring consisting of point and the top *n-1* elements of point-pdl; thus the default argument is 2. With a numeric argument of 1, it rotates the entire point-pdl. A negative numeric argument rotates the ring in the other direction.

c-X c-m-SPACE — Move to Default Previous Point

Rotates the point-pdl, the same as c-m-SPACE except that c-X c-m-SPACE has a default of 3. A numeric argument specifies the number of entries to rotate and sets the new default before rotating the point-pdl.

Showing the Mark

c-X c-X — Swap Point And Mark

Exchanges point and mark. It works even when no region is active. It highlights the text between point and mark.

With an argument, it does not exchange point and mark, but instead it highlights the text between point and mark.

Registers in Zmacs

Saving and Moving to Locations in Registers

You can assign one-character "names" to locations in the buffer, which can be helpful for setting up a series of places in your text to which you want to return for some reason — to double-check several items without interrupting your text entry or editing, if you are considering a format change that will affect several parallel points, or simply to return quickly and easily to rough spots that require further work.

c-X S — Save Position

Saves the current location in a register. It prompts for a one-character register name.

c-X J — Jump to Saved Position

Moves point to a position that was saved in a register. It prompts for a register name and switches buffers to move to the saved position, if necessary.

Saving and Inserting Regions in Registers

c-X X — Put Register

Copies the text of the region into a register. It prompts for a register name. With a numeric argument, it deletes the region from the buffer after copying it.

c-X G — Open Get Register

Inserts text from a specified register into the buffer. It prompts for the name of the register. It overwrites blank lines in the buffer the way RETURN does (using the command Insert Crs). It leaves the mark before the inserted text and point after it. With a numeric argument, it puts point before the text and the mark after.

List Registers (m-X)

Displays names and contents of all defined registers. It shows the name of the register and whether it contains a position or text. If the register contains a position, it tells which character on the line the position is at, and shows the first 50 characters on that line. If the register contains text, it shows the first 50 characters on the first line of that text.

```
List of all registers:
D (text)       This text was marked as a region and saved here
1 (position)   Char 0. in "another line containing a position"
Done.
```

Show Register (m-X)

Displays the contents of a register in the typeout window. It prompts for a register name and then tells whether the register contains a position or text:

```
Register A contains a position: Character 0 in this line:
this is the line
```
or
```
Register A contains text:
```

Kill Register (m-X)

Kills a register.

Commands to Mark Regions

To *mark* a piece of text means activating mark and then positioning point and mark on either side of the text, making it the region. The simplest way to mark some text is to go to one end of the text, set the mark there (using the Set Pop Mark command), and go to the other end of the text. See the section "Setting/Popping the Mark". However, several convenient commands mark different specific amounts of text:

m-@	Marks a word.
c-m-@	Marks an expression.
c-m-H	Marks a definition.
m-H	Marks a paragraph.
c-X c-P	Marks a page.
c-X H	Marks the whole buffer.
c->	Marks to the end of the buffer.
c-<	Marks to the beginning of the buffer.

Marking Words

m-@ Mark Word

Puts the mark at the end of the current word. With a numeric argument of *n*, m-@ puts the mark *n* words forward from point.

Marking Lisp Expressions

c-m-@ Mark Sexp

Marks the current expression by putting mark at the end.

With a numeric argument *n*, it moves forward *n* expressions and puts the mark there. For a more detailed description of how to move forward *n* expressions, see the section "Motion by Lisp Expression".

c-m-H Mark Definition

Puts point and mark around the current definition.

Marking Paragraphs

m-H Mark Paragraph

Puts the mark at the end of the current paragraph and moves point to the beginning, so that the current paragraph becomes the region. With a numeric argument *n*, m-H puts point at the beginning of the current paragraph and marks *n* paragraphs forward from there.

Example

m-3H marks the current paragraph and the following two; m- -1H marks the preceding paragraph. When marking preceding paragraphs, point is left at the end of the region, and when marking current and succeeding paragraphs, point is left at the beginning of the region.

Marking Pages

c-X c-P Mark Page

Puts the mark at the end of the current page and moves point to the beginning, so that the current page becomes the region.

With a numeric argument of *n*, c-X c-P marks the *n*th page after the current one. If *n* is zero, this is the current page; if *n* is negative, this page comes *before* the current page.

Marking Buffers

c-X H Mark Whole

Marks the whole buffer by putting point at the beginning and the mark at the end.

With any numeric argument, c-X H puts the mark at the beginning and point at the end.

Marking to End of Buffer

c-> Mark End

Marks from the cursor to the end of the buffer by putting the mark at the end of the buffer.

Marking to Beginning of Buffer

c-< Mark Beginning

Marks from the cursor to the beginning of the buffer by putting the mark at the beginning of the buffer.

Region-Manipulating Commands

Saving a Region

m-W Save Region

Puts region on kill history list without deleting it. For information on kill merging and the Append Next Kill command, c-m-W, see the section "Kill Merging".

Deleting a Region

c-W Kill Region

Deletes the region. If there is no region, c-W produces an error.

This command ignores numeric arguments and places the deleted text on the kill history list. For information on retrieving history elements and the Yank command, c-Y, see the section "Retrieving History Elements".

Compiling a Region

c-sh-C Compile Region
Compile Region (m-X)

Compiles the region, or if no region is defined, the current definition.

Transposing Regions

c-X T Exchange Regions

Exchanges two regions delimited by point and last three marks.

After transposing regions, you can undo the effect of this command by invoking it again.

Hardcopying a Region

Hardcopy Region (m-X)

Sends a region's contents to the local hardcopy device for printing.

For full information on Genera hardcopying, see the section "How to Get Output to a Printer".

Filling a Region

When Zmacs *fills* text it breaks it up so that it does not extend past the *fill column*. The fill column determines the right margin, and is the first column in which text is not to be placed by m-Q, m-G, or Auto Fill Mode formatting. In addition, the *fill prefix*, if set, is inserted:

- At the beginning of each new line typed in while in Auto Fill Mode
- At the beginning of each line in a paragraph for m-Q and each line in a region for m-G

The fill prefix determines the left margin, and is empty unless set to contain some combination of spaces and characters. If you do not set the fill prefix, the left margin is the left edge of your Zmacs window. For example, to insert five spaces at the beginning of every line, insert them at the beginning of the current line, and with point at column six, use c-X .. To turn this fill prefix off, put point at the beginning of a line, and use c-X . again.

Adjusting or *justifying* text inserts extra spaces between the words to make the right margin come out exactly even.

m-Q Fill Paragraph

Fills the current (or next) paragraph. A positive argument means to adjust rather than fill.

m-G Fill Region

Fills the current region. A positive argument means to adjust rather than fill.

c-X . Set Fill Prefix

Defines Fill Prefix from the current line. All of the current line up to point becomes the Fill Prefix. Fill Region starts each nonblank line with the prefix (which is ignored for filling purposes). To stop using a Fill Prefix, do a Set Fill Prefix at the beginning of a line.

m-sh-Q
Fill Differently

This command can be used successively, with no other commands intervening, to fill text to successively wider fill-widths. If the fill-width is 48, the first m-sh-Q will fill the text to 48 characters width. The sequence of commands m-sh-Q m-sh-Q m-sh-Q m-sh-Q and so on would fill to a width of 48, then 49, 50, 51 and so on.

You can also give a argument to this command to specify the increment. So m-sh-Q c-5 m-sh-Q would fill to 48, then to 53.

Other Region-related Commands

For descriptions of the following commands:

Name and Invocation

Uppercase Region c-X c-U

Lowercase Region c-X c-L

Uppercase Code in Region (m-X)

Lowercase Code in Region (m-X)

See the section "Changing Case of Regions in Zmacs".

Manipulating Buffers and Files

Working with Buffers and Files

Files are semipermanent collections of information stored safely outside the Zmacs environment. *Buffers*, on the other hand, are more dynamic, temporary collections of information, used by Zmacs for manipulating text. Buffers live in the active Zmacs environment. Each buffer has its own point and mark as well as other associated information.

We say we use Zmacs to "edit files", but what we really do is copy a file into a buffer created for the purpose, edit the buffer, and then write out a new version of the file from the edited buffer. The old version of the file is retained, to be deleted explicitly when appropriate. Successive versions of files are distinguished by *version number*, a component of the file name that is incremented with each new revised copy (except on file server hosts such as UNIX that do not have version numbers).

Zmacs allows multiple buffers, so that you can edit many files simultaneously. Usually only one buffer is visible on the screen at a time. You can, however, divide the screen into multiple windows so that you can view the contents of several buffers at once.

Zmacs keeps track of the association between files and buffers. If you are editing a file's contents in a buffer, Zmacs gives that buffer the same name as that of the file being edited.

Buffer and File Names

Both buffers and files have long names that indicate the host directory as well as the file name (and version, where supported). Hence completion is a necessary aid and is always provided for entering buffer and file names.

Buffer Flags for Existing Files

Each buffer has a *modification flag* that tells whether the buffer has been changed to be different from the associated file. You can see the modification flag by clicking on either the List Buffers command or the Kill or Save Buffers command in the editor menu (editor menu is click Right once), or by pressing c-X c-B for List Buffers.

The modification flag is cleared when:

- The file is read into the buffer from the file system.
- The buffer is *saved*, that is, whenever its contents are written out to the associated file. As soon as its contents are modified thereafter, the modification flag is set and Zmacs displays an asterisk (*): (1) in the mode line to the right of the buffer name, and (2) whenever it displays output from the List Buffers command.

Buffer Flags for New Files

The List Buffers (c-X c-B) command uses the plus sign (+) to mark new files that have not been saved. In addition, it uses + to mark new buffers, not associated with files, that have text in them. This helps when you put text into a new buffer and later want to be reminded to write that buffer to a file.

Creating and Saving Buffers and Files

You do all your text editing in Zmacs *buffers*, which are temporary workspaces that can hold text. To keep any text permanently you must put it in a *file*. Files store data for any length of time.

To edit the contents of a file using Zmacs, you create a buffer and copy the file contents into it. To add text to the end of the buffer, move point to the end of the buffer and type the new text. Editing proceeds in the buffer, not in the file. The file remains unchanged until you explicitly write the modified buffer contents to the file.

If you create multiple buffers, Zmacs keeps track of which files you are editing in which buffers. This association allows you to use completion to switch among buffers while you are editing them; you do not have to type the file name more than once. Zmacs always displays the name of the file you are currently editing.

The information in this section allows you to find or create and save a file. For complete information on buffers and files, see the section "Manipulating Buffers and Files in Zmacs".

These are the commands you can use to create and save Zmacs buffers and files:

c-X c-F Find File
Reads the specified file into a buffer.

c-X c-S Save File
Saves out the changes to the current file.

c-X B Select Buffer
Selects the specified buffer.

c-X c-W Write File
Writes out the buffer to the specified file.

Creating a Buffer

Zmacs creates your initial buffer when you first enter the editor. To create other buffers, use c-X c-F (Find File) to create either an empty buffer or a buffer containing a file. c-X c-F prompts for the name of a file, terminated by RETURN.

When you type c-X c-F for the first time in a Zmacs session, Zmacs offers you, as a default file name, an empty file (with the Lisp suffix native to your host computer) in your home directory on your host computer. For example:

System	*Empty Buffer Name*
Genera	bork.lisp
UNIX	bork.lisp
VAX/VMS	bork.lsp

For more information about c-X c-F, see the section "Editing Existing Files".

Base and Syntax Default Settings for Lisp

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Common-Lisp. See the section "Buffer and File Attributes in Zmacs".

Buffer Contents with c-X c-F

The first time you use c-X c-F, you can create an empty buffer using the Zmacs default file name, create an empty buffer using a name that you specify, or create a buffer containing an existing file.

- To create an empty buffer with the initial default file name as the one Zmacs associates with your buffer, press RETURN.

- To create a new empty buffer, respond with any name. Zmacs creates an empty buffer, gives the buffer the new name, and displays (New File) in the minibuffer.
- To create a new buffer containing an existing file, respond to the prompt with the name of that file. Zmacs switches to an empty buffer, reads that file in, and names the buffer appropriately.

Saving a File

Once you have the file in your buffer, you can make changes and then *save* the file with c-X c-S, the Save File command. This makes the changes permanent and actually changes the file. Until then, the changes are only inside your Zmacs buffer and the file itself is not really changed.

Creating a File

The first time you save or write the buffer, Zmacs creates the new file. You can create a new file with c-X c-S.

You can also write the buffer out with c-X c-W, Write File, if you want to change the name of the file from whatever you specified originally. Zmacs prompts in the minibuffer for the name of the place you want to write the buffer's contents. c-X c-W also offers a default pathname, in this case, the name you supplied with c-X c-F.

Editing Existing Files

To tell Zmacs to edit text in a file, use c-X c-F, the Find File command, and give Zmacs a file name. You can enter the *pathname* of any file on any host that is reachable by network connections from your Symbolics Machine. If the file already exists, Zmacs locates the file and reads it into your buffer.

Selecting, Listing, and Examining Buffers

Current Buffer

At all times when using Zmacs, you have one *selected* buffer, which is the buffer that you are actively editing. This is the buffer in which all current activity takes place until you switch buffers.

Buffer History

With a single Zmacs window on the screen, the editor keeps one buffer history, the *global history list*, which remembers the previous-buffer history (stack history)

of that window. The top buffer in the stack is the currently selected one. Usually, when a buffer is selected, it is pulled out of the stack and put on top. The buffers near the top are usually the most recently used. Each time you change buffers Zmacs offers the name of the most recently used buffer as the default buffer name.

When we refer to the *n*th buffer, we mean the *n*th buffer in Zmacs's stack of buffers.

Every additional window maintains its own buffer history, but the global history list continues to display an entry for every buffer in every window.

When you create a new window, Zmacs initially takes the history list for the new window from the global history list. From then on, as you switch from buffer to buffer within that window, the list for that window reflects the history of those changes in chronological order. This affects particularly c-m-L (Select Previous Buffer) and the default for c-X B (Select Buffer).

The global history list still exists and is used for name completion and c-X c-B (List Buffers).

Buffer Commands

Changing Buffers

c-X B — Select Buffer

Prompts for the name of a buffer and selects that buffer, displaying its contents on the screen. If you press END or RETURN instead of a name, it reselects the second most recently selected buffer.

Using completion, it takes the string you enter and tries to complete it to an existing buffer name:

- When completion is successful, it selects that buffer.
- When completion is unsuccessful, (there is no buffer with the name given), it either waits for you to type more characters (if there are multiple possible completions) or it beeps to give you a chance to correct a typing error (if there is no possible completion). A subsequent response of c-RETURN creates a new buffer with the specified name and selects it.

If you precede the c-X B command with a numeric argument, Zmacs prompts for the name of the buffer and then creates and selects it.

c-m-L — Select Previous Buffer

Selects a previously selected buffer. With a numeric argument *n*, it selects the *n*th previous buffer. The default argument is 2. When the argument is 1, it rotates the entire buffer history. A negative argument means to rotate the other way. An argument of zero displays the buffer history, which is mouse sensitive.

c-X c-m-L Select Default Previous Buffer

With a numeric argument *n*, this is exactly the same as c-m-L. Without a numeric argument, this command *remembers the last numeric argument it received* and uses that as its argument this time.

This is useful if you happen to be working with the top few buffers on the buffer stack and want to cycle among them without having to remember how many there are.

Listing Buffers

c-X c-B List Buffers

Lists all the currently existing buffers in the typeout window, along with the editor mode of the buffer and the name of the associated file, if any. For buffers with associated files, it displays the version number of the file, if any. If there is no associated file, c-X c-B gives the size of the buffer in lines instead. For Dired buffers, it displays the pathname used for creating the buffer. It lists modified buffers with an asterisk.

With an argument of c-U, it prompts for a substring and then lists only buffers whose names contain that substring.

The buffer names are mouse sensitive. Click right on the name of the buffer for a menu of operations (Kill, Not Modified, Save, Select) for that buffer. You can select one of the buffers by clicking left on its name.

The first line of a buffer listing resulting from List Buffers, which says "Buffers in Zmacs", is mouse sensitive. You can click on this line to execute Edit Buffers on the same set of buffers. This is especially useful when you have done c-U c-X c-B and then realize you wanted to do c-U c-X c-sh-B instead.

List Buffers lists the buffers sorted in stack order. You can inhibit this sorting by setting the global variable **zwei:*sort-zmacs-buffer-list*** to **nil** (default is **t**).

Example

```
Buffers in Zmacs:
  Buffer name:                 File Version:          Major mode:

+ file1 /dess/zmacs VIXEN:                             (Fundamental)
= *Dired-1*                    VIXEN: /dess/zmacs/*    (Dired)
* doc.mss /dess/zmacs VIXEN:                           (Text)
  *Buffer-1*                   [1 line]                (Fundamental)

+ means new file or non-empty non-file buffer.  * means modified file.
= means read-only.
```

Editing Buffers

c-m-X Edit Buffers is not part of the standard comtab. It is similar to List Buffers (c-X c-B), except that the buffer listing that Edit Buffers produces is a buffer in its own right. (For an example showing how to make c-X c-B call Edit Buffers instead of List Buffers, see the section "Setting Editor Variables in Init Files".) It contains one line for each of the buffers in the editor.

(c-X c-sh-B) Edit Buffers

Displays a list of all buffers, allowing you to save or delete buffers and to select a new buffer. A set of single character subcommands lets you specify various operations for the buffers. For example, you can mark buffers to be deleted, saved, or not modified. The buffer is read-only; like the Directory editor (Dired) buffer, you can move around in it by searching and with commands like c-N and c-P.

The lines in the list are not mouse sensitive. With the cursor on the line for a buffer, the following single character commands apply to that buffer:

With an argument of c-U, it prompts for a substring and then lists only buffers whose names contain that substring.

RUBOUT	Undeletes buffer above the cursor.
SPACE	Selects the buffer on the current line, performing any marked actions.
D	Marks the buffer for deletion (K, c-D, c-K are synonyms).
E	Immediately selects the buffer at the point for editing, without performing any marked actions. (You can use c-m-L to return to Edit Buffers.)
U	Undeletes either the buffer on the current line or the buffer on the line above.
S	Marks the buffer for saving.
=	Compares the buffer to its corresponding file.
~	Marks the buffer for setting not modified.
X	Executes an extended command (same as m-X).

Showing a Buffer

Use Show Buffer to just look at a buffer without editing it.

c-X V Show Buffer
Show Buffer (m-X)

Prompts for the name of a buffer and prints out the buffer contents for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful, displaying a --MORE-- message at the bottom.

SPACE, c-V, SCROLL	Displays the next screenful.
BACKSPACE, m-V	Displays the previous screenful.
RUBOUT	Exits.

Anything else exits and is executed as a command.

Inserting Command Output into the Buffer

You might want to save some output produced by a command into the buffer, rather than seeing it displayed on the typeout window and then erased.

Execute Command Into Buffer

Execute Command Into Buffer (m-X)

Sends output from a command into the buffer. It prompts you for a command, either a key or an extended command. It inserts any typeout produced by the command into the buffer at point, rather than displaying it on the typeout window. Macro Expand Expression All (m-X) is a good example of a command whose output can be usefully saved in this manner.

Hardcopying the Buffer

Hardcopy Buffer (m-X)

Prompts for the name of a buffer and then prints the specified buffer on the local hardcopy device.

For full information on Genera hardcopying, see the section "How to Get Output to a Printer".

Renaming the Buffer

Rename Buffer (m-X)

Prompts for a new name for the current buffer and changes the name to the specified string. Note that, if you want a buffer's name to be a file name, rather than just a string, use m-X "Set Visited File Name" instead. This operation removes any file association that the buffer had.

Saving Buffers

Save File Buffers (m-X)

Offers to write out each buffer that is associated with a file. It prompts in the typeout window with the name of each buffer:

```
Save file cheatin-heart.lisp >hwilliams> L: ? (Y or N) Yes.
Save word abbrevs on file L:>hwilliams>jambalaya.qwabl? (Y or N) Yes.
Save file rooty-tooty.text >hwilliams L: ? (Y or N)
```

Encrypting and Decrypting the Buffer

Encrypt Buffer (m-X)

Encrypts the contents of the buffer. It prompts for a key and does not echo it as you type it. It prompts for the same key again, just in case you mistyped it because of the lack of echoing, and makes sure you typed it the same both times. This key can consist of plain alphanumeric text only. Punctuation or other funny characters are ignored. Upper and Lower case are equivalent. The encryption algorithm is the same one used by the Hermes mail-reading system.

Decrypt Buffer (m-X)

Decrypts the contents of an encrypted buffer. It prompts for a key and does not echo it as you type it. The encryption key given for decrypting must match the one used for encrypting. The encryption algorithm is the same one used by the Hermes mail-reading system.

Reading a File Into a New Buffer

Edit File (m-X)

c-X c-F Find File

Prompts for the name of a file and looks for a buffer currently associated with that file. If one is found, it selects it. Otherwise, it creates a new buffer and reads that file into it.

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Common-Lisp. See the section "Buffer and File Attributes in Zmacs".

Reading a File Into an Existing Buffer

The c-X c-V command, Visit File, is primarily useful when you type in a mistaken file name after c-X c-F and Zmacs responds (New File). You can simultaneously read in the correct file and get rid of the unwanted buffer with Visit File. Note that this is not the same as c-X I or "Insert File" m-X.

c-X c-V Visit File

Prompts for the name of a file and reads that file into *the current buffer*. This action associates the current buffer with the specified file.

This command can be used only if the current buffer is not already associated with an existing file.

Writing the Buffer Contents to a File

c-X c-W Write File

Prompts for the name of a file and writes out the contents of the current buffer to the specified file. This changes the current buffer's name and associates it with the specified file. Subsequent saves using c-X c-S save to the newly specified file. This operation clears the modification flag.

Saving the Buffer Contents to the File

c-X c-S Save File

Writes the contents of the current buffer out to the associated file and clears the modification flag. It does not write the file if the buffer is unchanged from when the file was last visited or saved. It reads a file name from the minibuffer if the current buffer does not have an associated file.

Re-reading a File Into the Buffer

Revert Buffer (m-X)

Re-reads information into the buffer that it is associated with. For example, you can revert a Dired buffer to see the most current listing of that directory. You can also read in the most up-to-date version of a file. The command prompts for a buffer name, defaulting to the current buffer. The prompt serves as a confirmation, since Revert Buffer (m-X) throws away any modifications made to the buffer since you last saved or read the file or other information. This command is useful if you have damaged the buffer and want to start over or if the associated file is more current than the buffer. This operation clears the modification flag.

Refind File (m-X)

Re-reads a specified file into its associated buffer only if that file has changed on disk. The command prompts for a buffer name, defaulting to the current buffer. If the associated file on disk has changed, it re-reads the file into the buffer. If the associated file on disk has not changed, it tells you that it is not necessary to refind that file. This command is useful when more than one person works on the same program.

Refind All Files (m-X)

Re-reads only those files that have changed on disk into their associated buffers, asking about each one. If the associated file on disk has not changed, the command tells you that it is not necessary to refind that file. This command is useful when more than one person works on the same program.

With a numeric argument, Zmacs asks you for a string, which it matches with any part of the buffer names and operates only over buffers whose names contain that string.

The behavior of these three commands can be controlled with the variable **zwei:*revert-unedited-buffers-for-new-versions***. See the section "Zmacs Customization in Init Files".

Creating a Fundamental Mode Buffer

Find File In Fundamental Mode (m-X)

Creates a fundamental mode buffer containing the file. This is useful because Zmacs does not parse the file while reading it in, thus the names of the functions in the file do not conflict with those already known to completion in m-. and similar commands. This command is necessary if the normal parsing of a Lisp Mode file signals an error, preventing it from being read into the editor to correct the cause of the error.

Associating a File with a Buffer

Set Visited File Name (m-X)

Prompts for the name of a file and associates the current buffer with that file. (Use "Rename Buffer" to change the name of a buffer to a string). This command does *not* read the specified file into the buffer. Effectively, the current contents of the buffer are declared to be the new intended contents of the specified file.

This command should be used with caution to avoid unintentionally destroying the old contents of the specified file.

Destroying Buffers

c-X K Kill Buffer

Prompts for the name of a buffer and destroys that buffer. If you press END or RETURN instead of a name, c-K destroys the current buffer and prompts for the name of a buffer to select instead.

Kill Some Buffers (m-X)

For each existing buffer, tells you something about the status of the buffer and asks whether or not to delete it. If you elect to delete a buffer that has been modified since it was last saved, the command offers to save it first.

Kill Or Save Buffers (m-X)

Displays a menu listing all existing buffers. Modified buffers are initially marked for saving. Choices are: Save, Kill, Unmodify, and Hardcopy. Specify these options next to the buffer names in the menu. This command is bound to c-X c-m-B and appears on the editor menu. Specifying a numerical argument to c-X c-m-B inhibits the initial marking of the menu.

Appending, Prepending, and Inserting Text

Appending a Region to a Buffer

c-X A — Append To Buffer

Prompts for the name of a buffer and appends the contents of the region onto the end of the specified buffer.

Appending a Region to a File

Append To File (m-X)

Prompts for the name of a file (Append region to end of file:) and appends the contents of the region onto the end of the specified file, writing a new version of that file.

Prepending a Region to a File

Prepend To File (m-X)

Prompts for the name of a file and prepends the contents of the region onto the beginning of the specified file.

Inserting a Buffer Into Another Buffer

Insert Buffer (m-X)

Prompts for the name of a buffer and inserts the entire contents of that buffer into the current buffer at the cursor.

Inserting a File Into a Buffer

Insert File (m-X)

Prompts for the name of a file and inserts the contents of that file into the current buffer at the cursor. Note that this is not the same as c-X c-V or "Visit File" m-X.

Comparing Files and Buffers

Source Compare

Source Compare (m-X)

Compares two quantities of text. Prompts to determine the type of each quantity to be compared. You may be further prompted as appropriate to the quantity type you select. A single keystroke is all that is required to identify the quantity type.

Possible quantity types, and their associated keys are:

Key	*Quantity*	*Notes*
B	Buffer	Prompts for a buffer name.
D	Definition	Uses the definition at the cursor.
F	File	Prompts for a file name.
K or c-Y	Last Kill	Does not prompt. Uses most recently killed quantity (what c-Y would yank).
P or m-Y	Previous Kill	Does not prompt. Uses second most recently killed quantity (what c-Y m-Y would yank).
R	Region	If a region has been established, that region is used without prompting; otherwise, the user is permitted to interactively establish a region to use.

The results are displayed in the typeout window.

You can modify the behavior of the command with numeric arguments:

- 2 means ignore case and style in making the comparison
- 4 means ignore leading whitespace
- 6 means ignore case, style, and whitespace

Source Compare saves the output in a support buffer named *Source-Compare-*. You can read the comparison while checking the file, for example, by going into two window mode with the comparison in one window and the file in the other.

Example

This example shows a comparison between the file bottomley.lisp, as it was read into the buffer, and the buffer horatio.lisp, which contains the contents of the file new *plus* changes that have been made:

```
Source compare made by cheapjack on 1/17/90 14:59:20           -*-Fundamental-*-
of File S:>cheapjack>bottomley.lisp.1
with Buffer horatio.lisp >cheapjack S:
**** File S:>cheapjack>bottomley.lisp.newest, Line #4

  (setq TV:*wholine-clock-delimiters* nil)      ; Kill the "[]" around the clock
**** Buffer horatio.lisp >cheapjack S:, Line #4
  (setq tv:*mouse-exit-target-global-enable* nil)  ; Scroll blobs must die
  (setq TV:*wholine-clock-delimiters* nil)      ; Kill the "[]" around the clock
***************
```

```
Done.
```

When entered with a numeric argument of 6, Source Compare ignores both leading whitespace and case and style.

```
Source compare made by robin-hood on 1/17/90 15:45:09          -*-Fundamental-*-
of Buffer bud-abbott
with Buffer lou-costello
No differences encountered.
Source compare made by robin-hood on 1/17/90 15:51:34          -*-Fundamental-*-
of Buffer lou-costello
with Buffer bud-abbott
No differences encountered.
```

When entered with a numeric argument of 4, Source Compare ignores only leading whitespace.

```
Source compare made by robin-hood on 1/17/90 15:54:20          -*-Fundamental-*-
of Buffer bud-abbott
with Buffer lou-costello
**** Buffer bud-abbott, Line #11
  (si:cp-on si:*cp-dispatch-mode* 'si:arrow-prompt)
**** Buffer lou-costello, Line #11
  (SI:CP-ON SI:*CP-DISPATCH-MODE* 'SI:ARROW-PROMPT)
***************

Done.
```

When entered with a numeric argument of 2, Source Compare ignores only case and style.

```
Source compare made by robin-hood on 1/17/90 15:53:38          -*-Fundamental-*-
of Buffer bud-abbott
with Buffer lou-costello
**** Buffer bud-abbott, Line #5
 (setf si:*kbd-auto-repeat-enabled-p* t)

**** Buffer lou-costello, Line #5
             (setf si:*kbd-auto-repeat-enabled-p* t)

***************

Done.
```

Source Compare Merge

Source Compare Merge (M-X)

Compares two text quantities, prompting for type and name, and produces a new version that reconciles the differences between the two. When comparing the text

quantities, you see both versions and can choose which version which version (if any) to accept. You can also manually edit one or both versions at any point.

Possible quantity types, and their associated keys are:

Key	*Quantity*	*Notes*
B	Buffer	Prompts for a buffer name.
D	Definition	Uses the definition at the cursor.
F	File	Prompts for a file name.
K or c-Y	Last Kill	Does not prompt. Uses most recently killed quantity (what c-Y would yank).
P or m-Y	Previous Kill	Does not prompt. Uses second most recently killed quantity (what c-Y m-Y would yank).
R	Region	If a region has been established, that region is used without prompting; otherwise, the user is permitted to interactively establish a region to use.

At each place where the sources differ, the command prompts you twice. The first time you specify what to do to resolve the difference (prompts: `Specify which version to keep:`). (For example, you can keep one or the other version, both of them, or neither.) Respond to the prompt using these subcommands:

Option	*Action*
1	Leaves the first alternative in the text, redisplays the contents, and asks for confirmation of change.
2	Leaves the second alternative in the text, redisplays the contents, and asks for confirmation of change.
*	Leaves both alternatives in the text, redisplays the contents, and asks for confirmation of change.
I	Leaves both alternatives in the text, along with the message lines from the source compare (`*** MERGE LOSSAGE ***`), but does not ask for confirmation.
SPACE	Leaves both alternatives in the text, but does not redisplay the contents or ask for confirmation.
!	Disposes of this and all remaining differences the same way, without confirmation. It asks: `What to do with remaining differences (1, 2, *, I, or RUBOUT?)` It uses whichever option you choose for the rest of the differences.
c-R	Exits from the prompt and allows you to edit. Press `END` to return to this question.

RUBOUT	Leaves nothing in the new buffer and does not redisplay the contents or ask for confirmation.

The second time you confirm or reject the change that was made. The screen now shows the change that was made as a result of your choice and prompts: `Please confirm the change that has been made: (SPACE, RUBOUT, or c-R)`. Confirming it keeps that change and moves on to the next difference. Rejecting it returns to the prior appearance so that you can make a different choice:

Option	*Action*
SPACE	Yes, that's right.
RUBOUT	No, take that back.
c-R	Exits from the prompt and allows you to edit. Press END to return to this question.

When you finish confirming your decisions, Zmacs incorporates all changes into the new version in the specified buffer and the minibuffer displays: `Done. Resectionizing the buffer.`

Source Compare Merge also has a mouse interface. You can answer the first question by clicking Left on the text you want to keep or on the dividing line between them to keep both. You can answer the second question by clicking Left for "yes" (changes confirmed) or Middle for "no" (changes rejected).

Compare/Merge Commands for Definitions

The compare/merge commands operate on definitions by comparing, or comparing and merging, the current version with the newest version, newest version on disk, or installed version.

Comparing/Merging Current/Newest Versions

Source Compare Newest Definition (m-X)

Compares the current definition with the newest version in the normal source file for this definition, regardless of patch files. This command never looks in patch files; it only looks in original source files. If the definition was added by a patch (so that no original source file was recorded), the command cannot find the name of the source file. However, if you read the source file into the editor, it finds the definition in the editor buffer. You can use this command for comparing patch files and source files.

Source Compare Merge Newest Definition (m-X)

Compares and merges the current definition with the newest version in the normal source file. This command never looks in patch files; it only looks in original source files. If the definition was added by a patch (so that no original source file was recorded), the command cannot find the name of the source file. However, if

you read the source file into the editor, it finds the definition in the editor buffer. You can use this command for comparing patch files and source files.

Comparing/Merging Current/Saved Versions

Source Compare Saved Definition (m-X)

Compares the current definition with the source for the newest version on disk.

Source Compare Merge Saved Definition (m-X)

Compares and merges the current definition with the source for the newest version on disk.

Comparing/Merging Current/Installed Versions

Source Compare Installed Definition (m-X)

Compares the current definition with the source for the installed version.

Source Compare Merge Installed Definition (m-X)

Compares the current definition with the source for the installed version, merging the results.

Window Commands

Using Two Windows, Select Bottom

c-X 2 Two Windows

Shows two windows, selecting the bottom one. It splits the frame into two editor windows, selects the bottom one, and displays the next buffer from the global history in it. With a numeric argument, it displays that same buffer in the second window.

Using Two Windows, Select Top

c-X 3 View Two Windows

Shows two windows, selecting the top one. It splits the frame into two editor windows, selects the top one, and displays the next buffer from the global history in it. With a numeric argument, it displays that same buffer in the second window.

Creating Two Windows, Specifying Other Contents

c-X 4 Modified Two Windows

Selects a buffer, file, or definition in the other window. c-X 4 combines the functions of splitting the frame and selecting contents for the second window. It prompts for the type of contents you want for the second window: Select what in other window? (B, F, D, or J), for buffer, file, definition, or jump to register. Then it reads the name of the file, buffer, definition, or register that you want to select for that window.

Creating Two Windows with the Region in Top

c-X 8 Two Windows Showing Region

Makes two windows on the same buffer, with the top one displaying the current region.

Changing Window Size

c-X ^ Grow Window

Changes the size of the current window by some number of lines. With a positive numeric argument, it expands the window; with a negative numeric argument, it shrinks the window.

Choosing The Other Window

c-X O Other Window

Moves the cursor to the other window.

Returning to One Window

c-X 1 One Window

Returns the editor frame to displaying only one window. It expands the current window to use the whole frame. With a numeric argument, it expands the other window to use the whole frame.

Scrolling The Other Window

c-m-V Scroll Other Window

Scrolls the other window up several lines. By default, it scrolls the same way as c-V. With no argument, it scrolls a full screen. With just a minus sign as an argument (c-m- -V), it scrolls a full screen backward. A numeric argument tells it how many lines to scroll — a positive number scrolls forward, a negative number scrolls backward.

Splitting The Screen

Split Screen (m-X)

Pops up a menu that offers to create a new buffer or find a file; makes several windows split among the buffers as specified.

File Manipulation Commands

The commands described in this section are unlike most other Zmacs commands. Their main business is not manipulating buffers and their contents, but rather files out in a file system. First we discuss some commands for dealing with files, then we describe buffer and file attributes, and finally we explain *Dired Mode*, a special Zmacs mode for directory editing.

Creating a Directory

Create Directory (m-X)

Creates a new directory. It prompts for a directory name, using the standard conventions for defaults. For consistency between hierarchical and nonhierarchical file systems, you specify the directory to be created as the directory component of a pathname. That is, you must end the directory name with whatever delimiter or separator is appropriate for the host.

Example

Host	*Directory string*	*Result*
TOPS-20	<A.B.C>	Creates directory C
Multics	>udd>Sun>Luna>z>	Creates directory z
Genera	>sun>luna>b>	Creates directory b
UNIX	/usr/jek/new/	Creates directory new

Currently, the file servers for VAX/VMS and TOPS-20 can fail to create directories, due to missing options.

Listing Files in a Directory

List Files (m-X)

Prompts for the name of a directory and displays the names of all the files in that directory.

The file names are mouse sensitive. Pointing at a file name and clicking left is the same as doing a c-X c-F (Find File) on that file. Clicking right pops up a menu with three items:

Load — Loads the file into the Lisp world. The file must be either a Lisp source file or a compiled Lisp (*bin* or *ibin*) file.

Find — Reads the file into an editor buffer.

Compare — Compares the file with its most recent version and prints the differences.

Displaying the Contents of a Directory

c-X c-D — Display Directory

Displays the directory of the file in the current Zmacs buffer. c-X c-D does not ask for a directory but lists files with the same host, device, directory, and name as the file in the current buffer. It lists files with any type and version. With a numeric argument, it prompts for a directory to list and lists that directory.

The heading of the directory listing is mouse sensitive; clicking left on it selects a Dired buffer containing that directory listing.

c-U c-X c-D does the same thing as List Files, except that it gives more details about each file.

Show Directory

Show Directory (m-X)

Prompts for the name of a directory and and displays the directory contents for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful displaying a --MORE-- message at the bottom.

SPACE — Displays the next screenful.

BACKSPACE — Displays the previous screenful.

RUBOUT — Exits.

Anything else exits and is executed as a command.

Show Login Directory

Show Login Directory (m-X)

Displays the directory contents of the user's home directory for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful displaying a --MORE-- message at the bottom.

SPACE — Displays the next screenful.

BACKSPACE — Displays the previous screenful.

RUBOUT — Exits.

Anything else exits and is executed as a command.

Showing a File

Use Show File to look at a file without editing it.

Show File (m-X)

Prompts for the name of a file and displays the file contents for viewing only in the typeout window. If there is more than one screenful, it pauses between screensful, displaying a --MORE-- message at the bottom.

SPACE, c-V, SCROLL	Displays the next screenful.
BACKSPACE, m-V	Displays the previous screenful.
RUBOUT	Exits.

Anything else exits and is executed as a command.

Showing the Properties of a File

Show File Properties (m-X)

Prompts for the name of a file and displays all the properties of the file that are maintained by the file system on which it resides. These are the properties such as creation date and time, author, time of last access, and length. For files on a Lisp Machine file system, it displays user-defined properties as well.

It prompts for a file specification, which it merges with the current default to form the pathname. Wildcards are not accepted; this must correspond to a unique file or directory name.

Setting the Properties of a File

Set File Properties (m-X)

Sets the properties of a file. This is a synonym for Change File Properties (m-X).

Properties are the qualities of the file that are maintained by the file system on which it resides, such as creation date and time, author, time of last access, and length. For files on a Symbolics File System, this means user-defined properties as well. It prompts for the name of a file and pops up a choose-variable-values window, allowing you to alter various properties of the file. The exact properties that can be altered depend on the file system, but they might include:

- Generation (version) retention count
- Author
- Creation, modification, and reference dates
- Protection flags

- Other file-associated information

Hardcopying a File

Hardcopy File (m-X)

Enables you to specify the name of a file for printing on a local hardcopy device.

For full information on Genera hardcopying, see the section "How to Get Output to a Printer".

Renaming a File

Rename File (m-X)

Renames one or more files. It prompts for the name of a file and then asks for a new name for that file. It renames the specified file with that new name.

If the source file specification is wild, the target file specification must also be wild.

Copying a File Into Another

(m-X) Copy File

Copies any type of file to another specified file.

Prompts from the minibuffer for the names of two files and copies the contents of the first into the second. In file systems supporting multiple versions, this creates a new version of the second file whose contents are identical to those of the first.

Copy File determines whether the source file is a character file or a binary file and copies the file appropriately. Different file systems sometimes use different character sets, and if the file is a character file, character translations have to be done (for example, on some hosts Return characters have to be converted into a carriage return and a line feed).

The numeric argument controls copying of attributes and properties. With no numeric argument, it copies creation date and author and determines the mode (binary or character) of copy by the file being copied. To force mode, or suppress author or creation date copying, supply a numeric argument created by adding the values corresponding to the descriptions below:

1 Force copy in 16-bit binary mode.

2 Force copy in character (text) mode.

4 Suppress copy of author.

8 Suppress copy of creation date.

Examples

For example, to suppress author and creation date for copying:

```
c-12 (m-X) Copy File
```

Use wildcard pathnames to specify groups of files for copying. For example, to copy all files in the subdirectory `mine`:

```
F:>program>mine>*.*
```

If the source file specification is wild, the target file specification must also be wild.

```
                                                                    you type:
m-X Copy File
                                                                       Zmacs:
Copy File from:
                                                                    you type:
scrc:<lmfs>*.l*sp;0

(Copies all the newest .LISP and .LSPs)
                                                                       Zmacs:
to:
                                                                    you type:
ff:>sys-hold>scrc-sources>old-*.*.*
                                                                       Zmacs:
SCRC:<LMFS>TEST.LSP.3 is copied into

ff:>sys-hold>scrc-sources>old-test.lisp.3


SCRC:<LMFS>FILES.LISP.147 is copied into

ff:>sys-hold>scrc-sources>old-files.lisp.147
```

Note that .LSP gets mapped into .lisp because Copy File uses canonical types when the type of the target pattern is **:wild**.

This command can copy file authors and creation dates, when the target operating system supports setting these attributes. This action is not the default.

Creating Links to Files

Create Link (m-X)

Creates a link to a file. It prompts in the minibuffer for the names of two files as arguments; first the name of the link, then the name of the target pointed to by the link.

Deleting Files

Delete File (m-X)

Deletes a file. It prompts in the minibuffer for a file name, which can be wild. With a wild name as an argument, deletes multiple files. It lists the files that would be deleted and requires that you confirm the list. It deletes the files, showing any errors that occur but continuing rather than halting. Displays a message in the minibuffer if the specified file does not exist.

Deleting Multiple Versions

Clean Directory (m-X)

Deletes excess versions or temporary file types in the specified directory. The default for excess versions is more than two. It prompts for confirmation of files being deleted. With a numeric argument *n*, it deletes excess versions greater than *n*.

Excess is defined by the value of the Zmacs variable File Versions Kept or by the numeric argument. The temporary file types are defined by the Zmacs variable Temp File Type List. It accepts wildcards in the file name specification. (For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".)

Clean File (m-X)

This command works in file systems supporting multiple versions. It prompts for the name of a file (not including version number) and deletes excess or temporary versions of the specified file, keeping the most recent *n* files. Any numeric argument specifies the number of versions to keep. With no numeric argument, the default keeps two versions and deletes any excess. It prompts for confirmation of files being deleted.

Note:

- To specify file types to be automatically marked for deletion, change the value of the variable **zwei:*temp-file-type-list***, which contains a list of these files. This variable also accepts the value **:anything**, which can be any file type.

- To alter the default number (2) of versions to be kept, change the value of the variable **zwei:*file-versions-kept*** to any **:fixnum**.

Buffer and File Attributes

Attributes

Each buffer and generic pathname has *attributes*, such as Package and Base, which can also be displayed in the text of the buffer or file as an attribute list. An attribute list must be the first nonblank line of a file, and it must set off the

listing of attributes on each side with the characters -*-. If this line appears in a file, the attributes it specifies are bound to the values in the attribute list when you read or load the file.

How Attributes Work

Suppose you want your new program to be part of a package named **graphics** that contains graphics programs. In this case, you want to set the Package attribute to **graphics** in three places: the generic pathname's property list; the buffer data structure; and the buffer text. Here are two ways to make the change:

- If the package already exists in your Lisp environment, use Set Package (m-X) to set the package for the buffer. The command asks you whether or not to set the package for the file and attribute list as well. You can use this command to create a new package.

- Use Update Attribute List (m-X) to transfer the current buffer attributes to the file and create a text attribute list. Edit the attribute list, changing the package. Use Reparse Attribute List (m-X) to transfer the attributes in the attribute list to the file and the buffer data structure. If the package you specify by editing the attribute list does not exist in your Lisp environment, Reparse Attribute List asks you whether or not to create it with default characteristics.

Attribute-Manipulating Commands

Update Attribute List (m-X)

Updates the attribute list (-*- line) of the buffer. It creates or updates the attribute list of the file, using the current set of parameters. A new attribute list inherits the default base (10) and the default syntax (Common-Lisp) plus the Package, Mode, Backspace, and Fonts attributes of the current buffer. It includes the Backspace and Fonts attributes in the line only if they have values other than the defaults. It does not change other attributes in an existing mode line.

Reparse Attribute List (m-X)

Reparses the attribute list (-*- line) of the buffer. It finds the attribute list for the buffer and processes it to set up the environment that the line specifies. It changes the major mode, package, base, and so on, as necessary. When you edit the attribute list, you should then use this command to make the changes take effect in Zmacs. The changes take effect both for the editor buffer and for the file that the buffer is editing.

Example

Suppose the package for the current buffer is **user** and the base is 8. You want to create a package called **graphics** for the buffer and associated file. You also want to set the base to 10. If no attribute list exists, use Update Attribute List (m-X) to

create one using the attributes of the current buffer. An attribute list appears as the first line of the buffer:

```
;;; -*- Mode: LISP; Package: USER; Base: 8 -*-
```

Now edit the buffer attribute list to change the package name from USER to GRAPHICS and to change the base from 8 to 10. Use Reparse Attribute List (m-X). The command queries:

```
The file belongs in package GRAPHICS, which does not exist.
Create it with default characteristics,
 Try again, or Use another package? (C, T, or U)
```

Answer C to create the new package. The package becomes **graphics** and the base 10 for the buffer and the file.

File Attribute Checking

Zmacs notes errors in file attribute lists and warns you when it finds an unknown attribute. It goes ahead and ignores the unknown attribute in the list. The purpose of the warning is simply to help you detect misspellings.

Setting the Package

Set Package (m-X)

Changes the package associated with the buffer. It prompts for a package name. Forms that are read from the buffer are read in that package. (The default value for this attribute is **user**.)

Note that package names in an attribute list are always taken relative to the current syntax. Thus, **user** means something different in Zetalisp and Common-Lisp syntaxes.

The Set Package (m-X) command expects the name of a package as its argument. The package is expected to already exist. If the package does not exist, you are queried as to whether you wish to create it with default characteristics. In many cases, you should find the package definition or define the package explicitly instead of creating it with default characteristics. See below for more information about this.

Information about the package attribute exists in four places. Set Package offers to set the package for the generic pathname attribute list and updates the attribute line in the buffer when you answer Yes to:

```
Set it for the file and attribute list too?
```

Your answer affects the various versions of the package attribute as follows:

Location	*"Y"*	*"N"*
Generic pathname	changes	same
Buffer property	changes	changes
Buffer text	changes	same

```
Current package      changes      changes
```

The system is informed that the file belongs to the specified package. If you are not sure what to answer, say Yes. The global variable **zwei:*set-attribute-updates-list*** controls this query. Its default value is **:ask**. Setting the variable to **t** means Yes; **nil** means No.

Although the default value for the package attribute is **user**, you can have any package as the default package by specifying it as the value of the Zmacs variable Default Package. (For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".) You can set the variable in your init file by using the internal form of its name. (See the section "Creating an Init File".)

For example, in your init file:

```
(login-forms
        (setf zwei:*default-package* (pkg-find-package "tv")))
```

If you set the variable to **nil**, it sets the default to the package from the previous buffer.

You can edit a file attribute line to specify characteristics of the package. The package can appear in any of the following formats, which affects what happens if the file is read into a buffer and the package does not exist:

package-name — This package is expected to exist, and if it does not, the user is queried as to whether the package should be created with default characteristics. In many cases, it is not the right thing to create the package with default characteristics; instead, the user should find and evaluate the package definition before reading in the file.

(*package-name*) — If the package does not exist, it is automatically created with default characteristics. This differs from the above case in the sense that the developer of the program has anticipated that the package might not be defined in the user's environment, and has made an explicit decision that the package should be created automatically if it does not exist.

(*package-name key1 val1 key2 val2 ...*) — If the package does not exist, it is created with the characteristics given in the keyword/value pairs, which are package-creation options. This way of creating a package is usually only appropriate for quick and informal uses; normally, it is appropriate to define the package explicitly.

Base and Syntax Defaults

The mode line of Lisp source files (the line marked by -*-) contains the Base and Syntax attributes. The base can be either 8 or 10 (default). The syntax of a pro-

gram can be either Zetalisp or Common-Lisp. The defaults for these attributes are as follows:

- If there is a Base attribute, but no Syntax attribute, the syntax defaults to Common-Lisp.
- If there is a Syntax attribute of Common-Lisp, and no Base attribute, the base is assumed to be 10.
- If there is neither a Base nor a Syntax attribute, Base is assumed to be the default base (10) and the syntax is assumed to be Common-Lisp. Furthermore, a warning is issued to the effect that there is neither a Syntax nor a Base attribute. You should edit your program accordingly. With most programs, the Zmacs command Update Attribute List (m-X) will add the appropriate attributes to the mode line, following the above defaults.

Setting Lisp Syntax

The default syntax for Lisp buffers is Common-Lisp. If you are using Zetalisp, you must explicitly set the syntax in the file attribute line. For more information about Symbolics Common Lisp and Zetalisp, see the section "Lisp Dialects Available in Genera".

The file attribute line of a Common Lisp file should be used to tell the editor, the compiler, and other programs that the file contains a Common Lisp program. The following file attributes are relevant:

Syntax	The value of this attribute can be Common-Lisp or Zetalisp. It controls the binding of the Zetalisp variable **readtable**, which is known as ***readtable*** in Common Lisp. The default syntax is Common-Lisp.
Package	**user** is the package most commonly used for Common Lisp programs. You can also create your own package. Note that the Package file attribute accepts relative package names, which means that you can specify **user** rather than **cl-user**.

The following example shows the attributes that should be in an SCL file's attribute line:

```
;;; -*- Mode:Lisp; Syntax:Common-Lisp; Package:USER -*-
```

Set Lisp Syntax (m-X)

Set Lisp Syntax (m-X)

Changes the buffer into Common-Lisp syntax or Zetalisp syntax. It asks whether to update the attribute list (-*- line) of the buffer. If you answer yes, it creates or updates the attribute list of the file, using the current set of parameters, if any. It does not change other attributes in an existing mode line.

Other Set commands for File and Buffer Attributes

Each of the file attributes has a Set command associated with it. You have two choices when you want to change an attribute for a file:

- Edit the text of the buffer and then use Reparse Attribute List.
- Use the relevant Set command and answer Y to its query. The meanings for Y and N are the same as for the Set Package command (except that only the Set Package command affects the current package).

Update Attribute List Query

The Set commands use the value of the global variable **zwei:*set-attribute-updates-list*** to determine whether to query you about updating the file attribute list. The default value for the variable is **:ask**; set to **nil** to suppress the query.

Value	*Meaning*
:ask	Always asks whether to update the attribute list.
nil	Never updates the attribute list.
t	Always updates the attribute list.

Set *attribute* (M-X)

where *attribute* is one of the following: Backspace, Base, Fonts, Key, Lowercase, Nofill, Package, Patch File, Syntax, Tab Width, Variable, or Vsp. It sets *attribute* for the current buffer. It queries whether or not to set *attribute* for the file and in the text attribute list.

Attribute Descriptions

The following table describes some of the attributes, their associated Set commands, and the default value for the attribute.

Backspace	The Set Backspace command (default value **nil**) controls whether a backspace character in a file displays as the word "back-space" with a lozenge around it or performs the backspace. The default is the lozenge form.

Base

The Set Base command (default value 10) specifies the value of **zl:ibase** that the Lisp reader uses when reading forms from the file. Thus, Base controls the **zl:ibase** used when you evaluate or compile parts of the buffer, *and* controls the value of **zl:base** for printing during evaluating all or part of the buffer. This value does not affect the values of either **zl:base** or **zl:ibase** in the Lisp Listener you get by using SUSPEND.

Fonts

The Set Fonts command (default value **nil**) changes the set of fonts to use. It reads a sequence of font names separated by spaces, commas, or both from the minibuffer.

Lowercase

The Set Lowercase command (default value **nil**) means that the file being edited is intended to contain lowercase code or text. When the Lowercase attribute is **nil** (that is, not present), whatever case handling you specify prevails. To automatically uppercase code, use the following in your init file:

```
((login-forms
  (setf zwei:lisp-mode-hook
   'zwei:electric-shift-lock-if-appropriate))
```

(See the section "Creating an Init File".) When the Lowercase attribute is anything but **nil** (you answer Y to its query), the Electric Shift Lock Mode is never turned on automatically.

Nofill

The Set Nofill command has a default value of **nil**, which means that whatever autofilling behavior you specify prevails. When Nofill is anything else (you answer Y to its query), it means that autofilling is not appropriate for people who specify the mode of "autofilling if appropriate".

Use Nofill sparingly. Setting it means that everyone who edits the file has to be satisfied with Auto Fill Mode being off by default. In most cases, it is more reasonable to let an individual user's preferences prevail. It is useful for files that are not plain text, such as mailing lists, where you need to avoid spurious line breaks.

To have autofilling turned on by default, use the following in your init file (see the section "Creating an Init File"):

```
(login-forms
  (setf zwei:text-mode-hook
   'zwei:auto-fill-if-appropriate))
```

People who do not want it never get it by default.

Patch-File	The Set Patch File command has a default value of **nil**, which means that the file does not contain patches. When a file is classified as containing patches (you answer `Y` to its query), **fdefine** does not warn about functions being redefined during loading. Classifying something as a patch file also affects Edit Definition (which prefers files that are not patches) and **defvar** (which becomes **zl:setf**).
Tab-Width	The Set Tab Width command (default 8 characters) specifies how many spaces the editor uses between "tab stops".
Vsp	The Set Vsp command (default 2 pixels) specifies the vertical spacing (in pixels) between the text lines of an editor window. It specifies the distance between the descenders of one line and the ascenders of the next.

Dired Mode

There is a special Zmacs mode, called *Dired*, just for doing housekeeping in a directory. In this mode, you see the names of all the files in a directory at once, and can manipulate these files in various ways.

Entering Dired

The following commands specify a directory to manipulate and enter Dired mode.

Dired (`m-X`)

Edit Directory (`m-X`)

Prompts for a wildcard file specification for files contained in the specified directory. The default edits all files in the current directory by specifying wild name, type, and version. You must type the pathname in the form acceptable to your host system.

`c-X D` Dired

Edits the files in the directory that contains the current file.

With a numeric argument of 1, shows files with the same host, device, directory, and name as the file in the current buffer. It lists files with any type and version.

With a `c-U` argument, it prompts for a wildcard file specification showing the name of a directory to edit.

The Dired Display

When you go into Dired mode, Zmacs creates a special buffer that contains the names of the files that are under consideration, as well as some auxiliary information pertaining to those files. In a typical Dired buffer, each line describes a single file and lists the following information, from left to right:

- An indicator (D) that shows if the file has been marked for deletion or is already deleted
- The physical volume of the file (on some hosts)
- The name of the file
- The length of the file in blocks (where the length of a block is system-dependent)
- The length of the file in bytes, followed by the byte length in bits, enclosed in parentheses
- ! if the file has not been backed up to tape
- $ if the file has been marked against reaping
- @ if the file has been marked against deletion
- The file's creation date
- The file's creation time
- The date the file was last referenced, enclosed in parentheses
- The author of the file
- Optionally, the name of the last user to read the file

If there are too many files to be displayed in one screenful, the Zmacs window looks only at one section of the directory at a time (although the buffer does contain the names of all the files).

The files are arranged in alphabetical order by name.

Updating the Display

Use the Revert Buffer (m-X) command to update a Dired display. (See the section "Re-reading a File Into the Buffer".) After using Dired commands (or native host commands) to perform operations on files in your directory, invoke Revert Buffer, which reexecutes Dired with the default directory name and re-reads the updated directory into the buffer.

Dired Commands

Dired mode has its own command table (comtab) for manipulating the files whose names are displayed. These commands are described in this section. All invocations

given in this section are with respect to the Dired comtab and do not apply to regular Zmacs.

You use Dired by moving the cursor around to various lines and then specifying operations to be performed on the file listed on that line (the *current file*, while in Dired Mode).

Most Dired commands schedule some action for the future rather than performing it instantly. For example, when you want to delete a file using Dired, you move the cursor to the line describing that file and type D. Rather than deleting the file immediately, Dired *marks the file for deletion*. The deletion actually happens when you leave Dired mode and confirm your request. (See the section "Getting Out of Dired".)

Some of the commands in Dired mode take numeric arguments. You type numeric arguments in exactly the same way as you do in Zmacs proper, except that you do not have to hold a modifier key down while typing the argument — just typing the number suffices.

Command Summary

The following table summarizes the Dired commands:

Character	*Action*
RUBOUT	Undeletes file above the cursor.
SPACE	Moves to the next file.
!	Moves to the next file that is not backed up.
$	Complements the Don't Reap ($) flag.
,	Describes the attribute list of this file. In text files, this is the -*- line of the file. In compiled Lisp files, it includes information about the compilation as well.
.	Changes properties of current file.
@	Complements the Don't Delete (@) flag.
=	Compares this file with the newest version (Source Compare).
A	Queues this file for function application.
C	Copies this file to someplace else.
D	Marks the file for deletion (K, c-D, c-K are synonyms).
E	Edits the file in a buffer, or runs Dired if the line is a subdirectory name.
F	Marks the file for formatting and printing on the standard hardcopy device.
G	Sets and enforces the generation retention count.

*n*H Marks excess versions of the file for deletion (argument means whole directory).

I Displays a table explaining the Dired commands.

L Loads the file into Lisp.

*n*N Moves to the next file with more than *n* versions (see the Zmacs variable File Versions Kept). (For descriptions of Zmacs variables: See the section "How to Specify Zmacs Variable Settings".)

P Prints the file on the standard hardcopy device.

Q Exits. It shows the files marked for deletion and prompts for confirmation. The exit display marks files that have special status, using the following marks:

: a link
> most recent version
$ file marked for not reaping
! file not backed up

R Renames this file to something else.

U Undeletes either the file on the current line or the file on the line above.

V Views the file without creating a buffer (using View File conventions).

X Executes an extended command (same as m-X).

Default Pathnames in Dired

When the current buffer is a Dired buffer, and you execute an editor command that accepts a file name as an argument, the default file name is the file name that appears on the line of the Dired buffer that point is on.

This makes it easier to do things to the file that you are currently operating on in Dired. For example, you can move point to some line, do Compile File (m-X), and the command defaults to that file name.

Getting Out of Dired

Q Dired Exit
END

Leaves Dired mode. It prints the names of files marked for various actions and gets your final confirmation that these actions are really to be performed.

At this point the available options are:

Y Delete but do not expunge, also doing any other marked actions.

N Go back to Dired.

Q or X Abort out of Dired.

E Delete files and expunge directory. This is meaningful for file systems in which there is undeletion, such as TOPS-20, TENEX, and the Lisp Machine file system. This command is useful if you use Dired to free up disk space, since the disk space is not deallocated until the directory is expunged.

I Displays a table explaining the Dired commands.

Dired Exit performs those actions and returns to the previous buffer.

ABORT Dired Abort

Leaves Dired mode at once, without performing any actions on marked files. You can also just switch to another buffer.

Online Documentation for Dired

If you do not have a manual and cannot remember what the commands do, just press HELP.

? Dired Help

HELP

Displays a short table explaining the Dired commands.

Dired Menu

Click Right in Dired to display the Dired menu, which offers to perform the following actions on the listing:

```
Sort by reference date (up)
Sort by reference date (down)
Sort by creation date (up)
Sort by creation date (down)
Sort by file name (up)
Sort by file name (down)
Sort by file size (up)
Sort by file size (down)
Dired Automatic
Dired Automatic All
Dired Change File Properties
Dired Describe Attribute List
```

See the section "Deleting Multiple File Versions in Dired". See the section "Changing File Properties in Dired". See the section "Viewing File Attributes in Dired".

Loading a File in Dired

L Load File

Loads the current file. It displays a message `Loading the file...` in a typeout window and finishes with the message `Loading done`.

Moving Around in Dired

SPACE Down Real Line
c-N

Moves point to the next line (same as in regular Zmacs). With a numeric argument of *n*, it moves point forward *n* lines.

c-P Up Real Line

Moves point to the previous line (same as in regular Zmacs). With a numeric argument of *n*, it moves point backward *n* lines.

Viewing File Attributes in Dired

, Dired Describe Attribute List

This command is also available on the pop-up menu that you get when you click right in Dired. It prints out the contents of the attribute list of the current file (the one where point is). It works for character files and compiled files.

Changing File Properties in Dired

. Dired Change File Properties

This command is also available on the pop-up menu that you get when you click right in Dired. It edits the properties of the current file. These properties are the qualities of the file that are maintained by the file system on which it resides, such as creation date and time, author, time of last access, and length. For files on a Lisp Machine file system, this means user-defined properties as well. It pops up a choose-variable-values window, allowing you to alter various properties of the file. The exact properties that can be varied depend on the file system, but they might include:

- Generation (version) retention count
- Author
- Creation, modification, and reference dates
- Protection flags

- Other file-associated information

Viewing and Editing File Contents in Dired

You might want to look at the contents of a file before deciding what to do with it. You might also want to read the file into a buffer and edit it.

V — Dired Show File

Displays the contents of the current file on the typeout window.

Use this command when you just want to skim the contents of the file, not edit it. You can move forward while viewing with SPACE, c-V, or SCROLL and move backward with BACKSPACE or m-V.

E — Dired Edit File

Reads the current file into a Zmacs buffer and selects that buffer. You are then back in normal Zmacs and can edit the file normally. When you want to return to Dired mode, just use the c-m-L command to reselect the Dired buffer.

Comparing Recent Versions of Files in Dired

Often before deciding whether or not to delete a file, you want to find out exactly how extensive the differences are between the file and its most current version.

= — Dired Srccom

Compares the current file with its most recent version and displays the differences on the typeout window. With an argument of c-U, it asks what version to compare it to.

Copying and Renaming Files

C — Dired Copy File

Copies the current file. It prompts for the new pathname, displaying the default pathname.

R — Dired Rename File

Renames the current file. It prompts for the new pathname, displaying the default pathname.

Marking Files for Deletion

D — Dired Delete
K
c-D
c-K

Marks the current file for deletion. Dired puts a D in the first column to show that the file has been so marked.

With a numeric argument of *n*, it marks the next *n* files for deletion.

Sometimes you mark a file for deletion by mistake. Here is how you recover from this error:

U Dired Undelete

U takes one of two actions:

1. If the current file is marked for deletion, printing, or a function application (with a D, P, or A), reprieves it.

2. In file systems with soft deletion, U marks a deleted file for undeletion.

In either case, U removes the D, P, or A next to the file. If the current file is not marked with D, P, or A, U reprieves the file on the immediately preceding line, positioning point on that line.

With a numeric argument of *n*, it reprieves the files on the next *n* lines including the current line.

RUBOUT Dired Reverse Undelete

Reprieves the file on the preceding line.

With a numeric argument of *n*, it reprieves the files on the previous *n* lines including the current line.

Deleting Multiple Versions

If you are using Dired for housekeeping purposes, the following commands are useful:

N Dired Next Hog

Moves point to the next file with superfluous versions. Superfluous is defined by the value of the Zmacs variable File Versions Kept (whose default is 2) or by a numeric argument. (For descriptions of Zmacs variables, see the section "How to Specify Zmacs Variable Settings".)

H Dired Automatic

This command is also available on the pop-up menu that you get when you click right in Dired. It marks all the superfluous versions of the current file for deletion. With an argument of c-U, it marks superfluous versions of all files in the Dired buffer.

Setting Generation Retention Count

G Dired Set Generation Retention Count

Sets and enforces the generation retention count on this group of files, which specifies how many versions to save (that is, deletes multiple versions).

With a numeric argument *n*, sets it to *n* versions. With no numeric argument, prompts for a number in the minibuffer. An argument of zero means save all versions. *Enforce* means mark for deletion or undeletion.

Protecting Files From Being Reaped

In addition to keeping other users aware of protected files, protection features can also inform the system itself. Some file systems have automatic reaping facilities that go into action when storage becomes scarce. Most such systems have a *don't reap* bit associated with each file; use it to protect only your most vital files.

`$` Dired Complement No Reap Flag

Complements the Don't Reap flag associated with the current file; Dired displays the flag as $ between the length and date on that line. With a numeric argument of *n*, it complements the flag on the next *n* files, including the current one.

Protecting Files From Being Deleted

`@` Dired Complement Dont Delete Flag

Complements the Don't Delete flag associated with the current file; Dired displays the flag as @ between the length and date on that line.

With a numeric argument of *n*, it complements the flag on the next *n* files, including the current one.

Finding Files That Have Not Been Backed Up

Many file systems have tape backup facilities so that files can be copied onto tape against the possibility of a file system disaster. These systems almost always associate a bit with each file that is set when the file is created or modified and cleared when it is backed up to tape.

`!` Dired Next Undumped

Moves point forward to the next file that has not yet been backed up; Dired displays the flag as ! between the length and date on that line.

Marking Files to be Hardcopied

You might want to obtain a hardcopy of a group of related files. Dired allows you to mark files to be hardcopied as well as to be deleted.

`P` Dired Hardcopy File

Marks the current file for printing. Dired puts a P in the first column to show that the file is marked.

You can specify a numeric argument *n*, marking the next *n* files for printing.

Applying Arbitrary Functions to Files

Very occasionally, you want to perform some operation on selected files in your directory for which there is no Dired command provided. When this occurs, you can write up the operation that you want to perform as a Lisp function, whose single argument is the pathname of the file. The following command is relevant:

`A` Dired Apply Function

Marks the current file for having an arbitrary function applied to it. Dired puts a A in the first column to show that the file has been so marked. With a numeric argument of *n*, it marks the next *n* files, including the current one.

Zwei Undo Facility

The Zwei Undo facility remembers all the changes that you have made in an editor buffer and allows you to selectively undo any or all of the changes you have made. The Undo facility is available from Zmacs, Converse, the Zmail draft editor, and other editor buffers based on the Zwei substrate. (It is not available from the Input Editor or in the minibuffer.)

The simplest operation of the Undo facility is to undo the most recent change to the editor buffer. Go to a buffer, type something in, delete it, and then press `c-sh-U`. The deletion is undone. Region marking shows what was undone. Now press `c-sh-R`. You're back where you started. It is always safe to undo, because you can always redo, and vice versa.

The Undo (`m-X`) and Redo (`m-X`) commands are similar to `c-sh-U` and `c-sh-R` with the added feature that a display in the minibuffer shows you what will be undone or redone before any action is taken. `HELP U` also displays the change before undoing it.

Keep pressing `c-sh-U`. Previous changes to the buffer are undone. You can keep doing this until the buffer is returned to its original state. When you reach this point, if the buffer contains a file, it's no longer marked as needing to be saved. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing to be compiled.

Repeated pressing of `c-sh-R` will successively restore the buffer until all the undo commands have been cancelled out.

If you read in a file with no intention of changing it and accidentally type some characters into it, use `c-sh-U` rather than `RUBOUT` to get rid of them. That way, the buffer is no longer considered to be modified.

Undo commands operate only on the current buffer. Each buffer has an undo history, and a separate redo history. The undo history can be displayed with `c-0 c-sh-U`. Likewise, the redo history can be displayed with `c-0 c-sh-R`. Items in the history are mouse-sensitive. You can undo or redo all changes *up through a given*

change or you can undo or redo *any single change* in the history. By default, both histories are discarded when you save the buffer. See the section "Discard Change History".

Of course, subsequent changes may depend on the single change that you are undoing or redoing, so no guarantee can be made that undoing change number 13 in a 29-change history will have no effect on changes 14 through 29. (On the other hand, you can always back out of any undo or redo.)

This sounds more complicated in writing than it is when you are doing it. A few minutes experimentation in an editor buffer will make you a competent and confident user of the most important and common undoing and redoing operations.

After an undo or redo, the text that was modified is highlighted the same as if you had marked a region, but in this case there is no region, and the highlighting disappears when you type the next command. The history also shows you what constitutes each change. See the section "What is a Change to the Undo Facility?".

What is a Change to the Undo Facility?

To the Undo facility, a change is a *recorded* change to the textual contents of an individual buffer. Each buffer has its own undo and redo histories of recorded changes.

Most common changes are recorded. You can undo:

- Insertions into a buffer, including Execute Into Buffer and Evaluate [and Replace] Into Buffer (m-X).
- Deletions from a buffer, including those made by RUBOUT, m-RUBOUT, c-D, m-D and c-W.
- Text modifications, such as done by m-Q (fill paragraph) or c-TAB (indent differently).

In general, the way changes are recorded is to record the previous contents of the part of the buffer that was changed. Longer changes are compressed to minimize the amount of storage space.

Certain changes are not recorded by the Undo facility. You cannot undo:

- Renaming a file.
- Compiling a function.
- Reading a file into a buffer.
- Moving around in the buffer.

- Patch system insertions into patch buffers (and other changes to buffers made by the system).
- Revert Buffer (m-X).
- Anything in support buffers such as *Callers-17* or *Definitions-1*.
- Source Compare Merge (m-X) family of commands.

The Undo (m-X) and Redo (m-X) commands and HELP U all display the change before making it. You can also see information about changes in the undo and redo histories displayed by c-0 c-sh-U and c-0 c-sh-R.

You can get a good idea of what a change is by looking at the region marking that you see after an undo. Not all undos result in a clearly marked region — an undo of an insertion does not, for instance — but a line in the minibuffer reports the nature of each undo or redo.

The Undo facility uses simple rules to define changes. Changes are separated by blank lines, paragraphs, if you will. When you type or delete a single character in a new paragraph, that single character is a change. If you then type another character, the two characters together are a change, and so on until you reach the end of the paragraph or expression, that is, another blank line.

Likewise, successive deletions are merged, as are multiple simple commands operating on a single area of text.

There are variables to precisely define all aspects of a change. See the section "Customizing the Undo Facility".

Undo and Redo Commands

The undo and redo commands come in three styles:

- Undo and Redo (m-X) commands. These commands display the change that will be made before making it.
- Quick Undo and Redo commands. These are bound to c-sh-U and c-sh-R, respectively.
- Quick Undo and Redo in Region commands. These are bound to m-sh-U and m-Sh-R, respectively.

Each of these commands takes numeric arguments that allow selective undoing and redoing and also display the undo and redo histories. Use the undo and redo histories to make wholesale changes.

In addition, you can undo with the the HELP U key.

Undo Zwei Command

Undo (m-X)

Undoes a change to the buffer. The change is displayed first.

The first time you issue the command, it undoes the last change to the buffer. The next time you issue it, it undoes the previous change to the buffer. You can continue this until all changes to the buffer are undone and the buffer is considered unmodified. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing compiling.

For an otherwise equivalent accelerated version of this command, see the section "Quick Undo Command".

If you undo something you didn't intend to, Redo (m-X) redoes any undo. See the section "Redo Zwei Command".

With no numeric argument, Undo undoes the most recent change that has not already been undone. With a positive numeric argument, Undo undoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as issuing the command *n* times.

With a negative argument, Undo undoes *only* the *n*th most recent change that has not already been undone. This allows undoing changes out of order, but can sometimes surprise you when the undone change overlaps with a later change that was not undone. Use at your own risk.

With an argument of 0, Undo displays the undo history. Each change is numbered and described in abbreviated form. The undo history is mouse-sensitive. Clicking Left undoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle undoes *only* the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Undo Command

c-sh-U Quick Undo

Undoes a change to the buffer. The first time you press it, it undoes the last change to the buffer. The next time you press it, it undoes the previous change to the buffer. You can continue this until all changes to the buffer are undone and the buffer is considered unmodified. And, if you undo all the changes to a section since it was compiled, it is no longer marked as needing compiling.

For an otherwise equivalent prompting version of this command, see the section "Undo Zwei Command".

c-sh-R redoes any undo. See the section "Quick Redo Command".

With no numeric argument, c-sh-U undoes the most recent change that has not already been undone. With a positive numeric argument, c-Sh-U undoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, c-sh-U undoes *only* the *n*th most recent change that has not already been undone. This allows undoing changes out of order, but can sometimes surprise you when the undone change overlaps with a later change that was not undone. Use at your own risk.

With an argument of 0, c-sh-U displays the undo history. Each change is numbered and described in abbreviated form. The undo history is mouse-sensitive. Clicking Left undoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle undoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Undo in Region

m-sh-U Quick Undo in Region

m-sh-U works just like c-sh-U except that it is limited to changes in the current region rather than in the current buffer. You are warned if an undo extends past the region. See the section "Quick Undo Command". There is no prompting (m-X) version of this command.

With no numeric argument, m-sh-U undoes the most recent change that has not already been undone. With a positive numeric argument, m-sh-U undoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, m-sh-U undoes *only* the *n*th most recent change that has not already been undone. This allows undoing changes out of order, but can sometimes surprise you when the undone change overlaps with a later change that was not undone. Use at your own risk.

With an argument of 0, m-sh-U displays the undo history of the region. Each change is numbered and described in abbreviated form. The undo history is mouse-sensitive. Clicking Left undoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle undoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the region affected by the change.

Redo Zwei Command

Redo (m-X)

Redoes a change to the buffer. The change is displayed first.

The first time you issue the command, it redoes the last undo in the buffer. The next time you issue it, it redoes the previous undo in the buffer. You can continue this until all changes to the buffer are redone and the buffer is back where you started from before you did the first undo.

For an otherwise equivalent accelerated version of this command, see the section "Quick Redo Command".

If you redo something you didn't intend to, Undo (m-X) undoes any redo. See the section "Undo Zwei Command".

With no numeric argument, Redo redoes the most recent change that has not already been redone. With a positive numeric argument, Redo redoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as issuing the command *n* times.

With a negative argument, Redo redoes *only* the *n*th most recent change that has not already been redone. This allows redoing changes out of order, but can sometimes surprise you when the redone change overlaps with a later change that was not redone. Use at your own risk.

With an argument of 0, Redo displays the redo history. Each change is numbered and described in abbreviated form. The redo history is mouse-sensitive. Clicking Left redoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle redoes *only* the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Redo Command

c-sh-R Quick Redo

Redoes a change to the buffer. The first time you press it, it redoes the last undo in the buffer. The next time you press it, it redoes the previous undo in the buffer. You can continue this until all changes to the buffer are redone and the buffer is back where you started from before you did the first undo.

For an otherwise equivalent prompting version of this command, see the section "Redo Zwei Command".

c-sh-U undoes any redo. See the section "Quick Undo Command".

With no numeric argument, c-sh-R redoes the most recent change that has not already been redone. With a positive numeric argument, c-sh-R redoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, c-sh-R redoes *only* the *n*th most recent change that has not already been redone. This allows redoing changes out of order, but can sometimes surprise you when the redone change overlaps with a later change that was not redone. Use at your own risk.

With an argument of 0, c-sh-R displays the redo history. Each change is numbered and described in abbreviated form. The redo history is mouse-sensitive. Clicking Left redoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle redoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Quick Redo in Region

m-sh-R Quick Redo in Region

m-sh-R works just like c-sh-R except that it is limited to changes in the current region rather than in the current buffer. You are warned if a redo extends past the region. See the section "Quick Redo Command". There is no prompting (m-X) version of this command.

With no numeric argument, m-sh-R redoes the most recent change that has not already been redone. With a positive numeric argument, m-sh-R redoes the *n* most recent changes. An argument of 1 is the same as no argument and an argument of *n* is the same as pressing the key *n* times.

With a negative argument, m-sh-R redoes *only* the *n*th most recent change that has not already been redone. This allows redoing changes out of order, but can sometimes surprise you when the redone change overlaps with a later change that was not redone. Use at your own risk.

With an argument of 0, m-sh-R displays the redo history of the region. Each change is numbered and described in abbreviated form. The redo history is mouse-sensitive. Clicking Left redoes *back to* the previous change, just as in using a positive numeric argument. Clicking Middle redoes just the highlighted change, just as in using a negative numeric argument. Clicking m-Left shows you the context of a change by highlighting the section of the region affected by the change.

The Undo and Redo Histories

The Undo facility keeps an undo history and a redo history for each Zwei buffer. By default, the history is discarded when you save the buffer, but you can set the variable **zwei:*discard-change-record-after-saving*** to **nil** if you wish to override that behavior. There is also a Discard Change History (m-X) command.

You can display the Undo history by giving an argument of 0 to the Undo (m-X) command or the c-sh-U commands. A 0 argument to m-sh-U displays the undo history of the current region.

An argument of 0 to the Redo (m-X) or c-sh-R displays the redo history of the buffer. A 0 argument to m-sh-R displays the redo history of the current region.

Histories are mouse-sensitive. Clicking Left on a history entry undoes or redoes *back to* the previous change. Clicking Middle undoes or redoes *only* the highlighted change. Clicking m-Left shows you the context of a change by highlighting the section of the buffer affected by the change.

Discard Change History

Discard Change History (m-X)

Throws away both the Undo and Redo histories of the current buffer. The buffer is still considered modified after this command is executed, but the histories are gone.

See the variable **zwei:*discard-change-record-after-saving***.

Customizing the Undo Facility

You can customize the Undo facility by setting one or more variables. Most of these variables affect the way the Undo facility records which changes it can undo or redo, but there are also variables for turning the Undo facility off, for turning off the region marking, and for changing the way the Undo facility handles yanking and multiple replaces.

You can set any of these variables in Lisp programs, including your `lispm-init` file. You must use **setf**, not **setq**, to set these variables.

A number of these variables are also defined as Zwei variables and can be set from inside an editor using the Set Variable (m-X) command. Using the Set Variable command, you can set each variable per buffer, per mode, or globally.

zwei:*enable-change-recording* *Variable*

Set to **nil** to turn off the Undo facility. The default is **t**.

zwei:*undo-sets-region* *Variable*

If **t**, undo and redo operations use region marking to highlight where the change was done. The marking disappears with the next keystroke. If **nil**, there is no marking.

The Zwei variable is called Undo Sets Region. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*yank-is-separately-undoable* *Variable*

If **t**, a yank is separately undoable from any edits before or after it. If **:multi-line**, which is the default, this is true only when the yanked text is more than one line. If **nil**, a yank can be merged with edits before or after it, approximately the same as if the yanked text had been typed in one character at a time.

The Zwei variable is called Yank is Separately Undoable. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*discard-change-record-after-saving* *Variable*

Set to **nil** to cause the Undo facility to *keep* the change histories when a buffer is saved. The **nil** setting conses a lot of garbage and may cause results contrary to user expectations. The default is **t**, meaning that change histories are cleared when a buffer is saved.

The Zwei variable is called Discard Change Record After Saving. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*simple-change-contiguity-range* *Variable*

Default = 3. This sets the maximum numbers of unchanged characters between sections of a simple change *within a line*. If more unchanged characters than this intervene between two changes, they will be considered two undoable changes. Fewer, and it's all one change.

zwei:*simple-change-size* *Variable*

Default = 50. This is the *maximum* number of characters in a change. Once a change is this number of characters, a new change is begun. Changes can be smaller than this number of characters. Edits at the end of such a change are considered part of the same change, but edits in the middle start a new change.

The Zwei variable is called Simple Change Size. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*insertion-amendment-size* *Variable*

Default = 10. This many or fewer characters are considered an amendment to the previous insertion and not a new change, and therefore are cannot be undone separately.

The Zwei variable is called Insertion Amendment Size. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*insertion-breakup-lines* *Variable*

Default = 4. Two insertions of at least this many lines with a blank line between them are considered two undoable changes. In other words, this is the size of an undoable paragraph insertion.

The Zwei variable is called Insertion Breakup Lines. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*undo-each-replace-separately* *Variable*

If **nil**, which is the default, all the changes made by a Replace String (c-Z) or Query Replace (m-Z) or other member of the Query Replace (m-X) family are undone as a unit. If **t**, the changes can be undone separately. Setting this variable is a matter of taste and style.

The Zwei variable is called Undo Each Replace Separately. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

zwei:*record-small-changes* *Variable*

Set to **nil** to cause the Undo facility to record only major changes, but to ignore small changes, such as RUBOUT. The default is **t**, and is strongly recommended.

The Zwei variable is called Record Small Changes. Using the Set Variable (m-X) command, you can set it per buffer, per mode, or globally.

Zmacs Speller

The Zmacs Speller is a small, simple set of tools to help you spell words right. The Speller can examine a single word, a Zmacs region or buffer, a file, or a group of files, for spelling errors. When it encounters a word not listed in one of its dictionary files, it alerts you with a message and a menu of possible responses. A large dictionary of common words is included as part of the Zmacs editor. You can add other dictionaries of special spellings, as used by individuals or groups of users on your system, to the list.

The quickest way to see what the Speller can do is to go to a Zmacs text buffer, type an incorrectly spelled or nonexistent word, and then press m-$. Say you type "gorax". When you press m-$, you see in the minibuffer at the bottom of the screen:

```
"gorax" is unknown and possibly misspelled.
```

A menu appears offering you a choice of **Prompt** and **Accept** and the word "borax", which is the only word in the basic dictionary close to the spelling of "gorax". The menu might also contain the names of one or more dictionaries.

If you meant to type "borax", click on the word on the menu and the spelling is changed.

If you want to accept the spelling "gorax", click on **Accept** or move the mouse cursor off the menu.

If you want some entirely different word, click on **Prompt** and you are prompted to type another word. If that word is spelled correctly, the message

```
Correcting "gorax" to "monkey".
```

appears in the minibuffer. If the word is spelled incorrectly, the change is made anyway, and the following messages

```
Warning: "shimbax" is not in the dictionary.
Correcting "gorax" to "shimbax".
"shimbax" is unknown and possibly misspelled.
```

appear in the minibuffer. Then the menu appears again to give you a chance to change "shimbax" to something else again.

If you click on a dictionary name, the spelling is added to that dictionary in memory for the rest of the session. You can also add the new words permanently to a dictionary by saving the dictionary to a file. See the section "Save Spell Dictionary".

See the section "Save All Spell Dictionaries".

Once you've tested the m-$ command, read a file into the buffer. Enter the command m-X Spell Buffer and watch what happens. Chances are there is a misspelled

word in the buffer, or at least a word that is not in the dictionary. The menu produced by this command has an additional entry, **Accept once**. Click on **Accept once** if you want the Speller to flag the word the next time it appears. Click on **Accept** if you want the Speller to ignore the word for the duration of the spelling check.

NOTE

To the Speller, any word not in the dictionaries is misspelled by definition. Not all words or alternate spellings are in the basic dictionary.

Also, for purposes of the Speller, a word in running text is defined to be a sequence of characters, each of which must be a letter or an apostrophe. The apostrophe is allowed so that contractions and possessives are recognized as words. This definition of a "word" is not exactly the same as that used by the Zwei "word" commands (`m-F` and so forth).

The Zmacs Speller Menu

When the Speller encounters a word not in a current dictionary in the course of a `m-$`, Spell Buffer, Spell Region, or Spell Word command, a warning appears in the minibuffer

```
"rugnue" is unknown and possibly misspelled.
```

and a menu gives you a series of actions to choose from.

Prompt

Prompts in the minibuffer for a replacement word. The misspelled word is replaced by the new word. The replacement word is also checked against the dictionaries and you are warned if it isn't found. Any uppercase letters you type are included when the word is replaced. If the questioned word had an initial capital letter, so will the replacement. Any other capital letters in the original word are ignored.

While the menu is showing, pressing `c-P` is the same as clicking on **Prompt**.

Accept once

Accepts this spelling just this once. That is, the Speller ignores the spelling this time, but if the same misspelling is encountered again, it challenges the spelling again. Moving the mouse cursor off the menu is the same as clicking on **Accept once**.

While the menu is showing, pressing `c-O` is the same as clicking on **Accept once**.

Accept

Accepts this spelling and considers it a correct spelling for the duration of the command. There is no way to save words permanently using **Accept**. To save

words permanently, you must accept them by clicking on a dictionary name.

While the menu is showing, pressing c-A is the same as clicking on **Accept**.

DICTIONARY-NAME — Accepts the spelling and adds it to the dictionary for the remainder of the boot session. You can also add the spelling to the dictionary file permanently, see the section "Save Spell Dictionary", and see the section "Save All Spell Dictionaries". To add a dictionary to the list, see the section "Read Spell Dictionary".

suggestion — Whenever possible, the Speller suggests words that are close in spelling to the questioned word. One or more suggestions may appear on the menu. If one of the suggestions is what you intended, click on it and the spelling in the text is changed to the suggested spelling.

Speller Commands for Spelling

The Zmacs Speller allows you to check spelling by word, region, buffer, file, or groups of files (including tag tables).

Spell This Word

m-$ Spell This Word

Checks the spelling of the current word. If point is just to the right of a word, that word is checked. This means you can type a word and then press m-$ to check the spelling. If point is on a word, that word is checked.

Use this command to check the spelling of individual words in your current buffer. If a region exists, m-$ checks the spelling of the region, just like Spell Region (m-X).

When checking the spelling of a single word, m-$ informs you if the spelling of the word has been checked against a site-specific or user-specific dictionary. If a word has been checked against the basic Speller dictionary, the following message appears in the minibuffer:

```
"tatterdemalion" is spelled correctly.
```

If a word has been checked against a site-specific or user-specific dictionary, the following message appears in the minibuffer:

"Wollongong" is spelled correctly *(according to TRADEMARKS).*

For more information about speller dictionaries: See the section "Speller Dictionaries".

Use Spell Word (m-X) to check the spelling of a word before typing it into a buffer. See the section "Spell Word".

Spell Word

Spell Word (m-X)

Prompts for a word and runs a spelling check on it. If the word is spelled according to the dictionaries, you are informed in the minibuffer. If the word is not spelled according to the dictionaries, you are informed. If the dictionaries contain any words similar to your misspelling, they are listed also.

Use this command to check the spelling of words before you type them in your buffer.

Use m-$ (Spell This Word) to check the spelling of a word already typed into the buffer. See the section "Spell This Word".

Spell Region

Spell Region (m-X)

Runs a spelling check on the current region.

Use this command to check spelling in a region you have marked.

Spell Buffer

Spell Buffer (m-X)

Runs a spelling check on the entire current buffer. It does not matter where point is when you issue the command. The entire buffer is checked.

Spell File

Spell File (m-X)

Runs a batch-mode spelling check over a file. With wildcards, you can also specify a group of files.

This command prompts for a pathname and also for the name of a buffer where the words not in the dictionaries are to be written. The questioned words are written to a buffer in alphabetical order, but without identification as to the file they came from. You can save this buffer if you wish.

See the section "Tags Spell".

Tags Spell

Tags Spell (m-X)

Runs a batch-mode spelling check on all the buffers of the current tags table and finds all of the words that aren't in any of the dictionaries and writes them (in alphabetical order) into a buffer you specify. You can save this buffer if you wish.

See the section "Spell File".

See the section "Tag Tables and Search Domains in Zmacs".

Speller Dictionaries

The Speller considers a word to be spelled correctly if it is in a current dictionary and to be misspelled if it is not in a current dictionary. The Speller always checks against the basic dictionary provided as part of Zmacs. In addition, the Speller checks against one or more optional site-specific dictionaries and one or more user-specific dictionaries.

When used to check the spelling of an individual word, m-$ informs you in the minibuffer if the spelling has been checked against a site-specific or user-specific dictionary.

Here are some basic definitions of dictionary terms:

- A *dictionary* or *current dictionary* is an object in the Lisp environment containing a set of words. It has an associated pathname, and a **modified-p** flag. The pathname refers to a dictionary file. The **modified-p** flag indicates whether words have been added to the dictionary since it was read in. If the **modified-p** flag is on, the Save Spell Dictionary and Save All Spell Dictionaries commands saves the dictionary. When a dictionary is saved, the **modified-p** flag is turned off.

  Most Speller commands operate only on the current dictionaries that are present in memory. You must save any changes to these dictionaries if you want the changes to be permanent.

  See the section "Save Spell Dictionary".

  See the section "Save All Spell Dictionaries".

- A *dictionary file* is a file that holds a set of words. It can be a binary (compiled) dictionary or a character (text) dictionary. These files are read into memory.

  ° A binary dictionary is compiled and has the canonical file type "dict" ("dc" on UNIX, "dct" on VMS). The binary dictionary is fast to load or dump.

  ° A character dictionary can be any file with text in it. There is no naming restriction, but it is clearer to use the file type "text". The words in the file are the words of the dictionary set, minus duplicates. This format can be easily created, examined and modified by editing.

- The Speller commands operate on a *list of dictionaries* currently being used. Order in the list doesn't matter. Some dictionaries on the list appear in the correction menu, and some do not. Whether or not to appear on the menu is a property of a dictionary.

To make changes in a dictionary permanent, the dictionary must be saved.

See the section "Save Spell Dictionary".

See the section "Save All Spell Dictionaries".

For an introduction to creating your own Speller dictionaries: See the section "Quick and Dirty Guide to Adding and Maintaining a Spell Dictionary".

Quick and Dirty Guide to Adding and Maintaining a Spell Dictionary

Here's how to add your own spell dictionary and keep it up to date:

1. Go into Zmacs and use c-X c-F to create a new file called spell.dict in your top-level directory.

2. Type in any words you know you want to have in your dictionary, or none at all, if you prefer.

3. Write the file out using c-X c-S.

4. If the file is quite long, use Compile Spell Dictionary (m-X) to make a binary dictionary, which loads faster.

5. Use Read Spell Dictionary (m-X) to read the new dictionary in this time. Henceforth, it will be read in when you log in.

6. As you use the Speller, you can add words to your dictionary from the Speller Menu by clicking on the dictionary name, or with Add Word to Spell Dictionary (m-X).

7. To save those new words, use Save Spell Dictionary (m-X).

If you begin to suspect that you have misspelled words in your spell dictionary, here's what to do:

1. Issue the Zmacs command Execute Command Into Buffer (m-X) in an empty Zmacs buffer.

2. Issue Show Contents of Spell Dictionary (m-X). The dictionary is written out into the buffer.

3. Find the misspelled words.

4. Issue Delete Word From Spell Dictionary (m-X) for each misspelled word.

5. Issue Save Spell Dictionary (m-X) to save the corrected dictionary.

This is deliberately simplified information on adding a dictionary and keeping it up to date. For complete information on speller dictionaries, see the section "Speller Dictionaries".

Speller Dictionary Management

Every system includes a basic dictionary, which is in the file SYS:SPELL;BASIC.DICT. This is a dictionary of about 30,000 common English words. Not all words or alternate spellings are in the basic dictionary. The dictionary does not include the names of Lisp language forms.

You can add dictionaries for yourself as an individual user, or for your site. Any text file can be used as a dictionary. Dictionaries can be in either binary (compiled) format, or character (text) format.

Adding User-specific Speller Dictionaries

The Zmacs Speller checks by default for a Speller dictionary in your home directory. This is a standard dictionary and is always included on the list of dictionaries if it exists.

The user-specific dictionary is sought in your directory, on your file server, under your name, with the file name "spell" and the file type "dict" or "text".

If you wish to add words to a dictionary as you go along, using the Speller, you must read the dictionary in ahead of time. Then, any time you click on the dictionary name, the word is added to the dictionary in memory. If you wish to add the words to the dictionary file on disk, you can save the dictionary.

See the section "Read Spell Dictionary".

See the section "Save Spell Dictionary".

See the section "Save All Spell Dictionaries".

A dictionary file is just a file of text. Put all the words you want in your dictionary into a file in your top-level directory and you have a user-specific dictionary.

To speed loading, you can compile the file. It is good practice to use the "text" file type for character (text) dictionary files and the "dict" file type for binary (compiled) dictionary files. When loading standard dictionaries, the Speller looks first in your directory for the file named spell.dict. If that file is not found, the Speller looks for spell.text. See the section "Compile Spell Dictionary".

If you wish to use other dictionaries besides the basic dictionary, you can add them to the dictionary list interactively with the m-X Read Spell Dictionary command. See the section "Read Spell Dictionary". You can also include other dictionaries on your dictionary list using **zwei:read-spell-dictionary**. Speller dictionary functions are most effectively used in a lispm-init file to do automatic dictionary operations when you log in. See the function **zwei:read-spell-dictionary**.

For example, evaluating the following form:

```
(zwei:read-spell-dictionary "f:>skimpy>my-old-list.text")
```

reads that file into memory as a dictionary and adds *MY-OLD-LIST* to the dictionaries named on the menu. In most cases, the menu lists just the name of the file and not the rest of the pathname, but if two dictionary files have the same name, then the menu lists the pathname for both (name-first, like Zmacs buffer names).

Conversely, evaluating this form:

```
(zwei:read-spell-dictionary "f:>skimpy>my-old-list.text" nil)
```

reads the file into memory as a dictionary, but does not add it to the dictionaries named on the menu. See the function **zwei:read-spell-dictionary**.

Note: If you use **zwei:read-spell-dictionary** in your lispm-init file, you must also use **zwei:read-standard-spell-dictionaries** if you want to use the standard dictionaries. **zwei:read-spell-dictionary** overrides the auto-loading of the standard dictionaries.

See the section "Adding Site-specific Speller Dictionaries".

Adding Site-specific Speller Dictionaries

The Zmacs Speller loads the basic dictionary in SYS:DATA;BASIC.DICT when you first use the speller. It is not loaded until you request it or need it.

You can also have other dictionaries designated as site-specific. Site-specific dictionaries are always included on the dictionary list for each user except for those users who override the auto-loading of standard dictionaries by evaluating **zwei:read-spell-dictionary** without also evaluating **zwei:read-standard-spell-dictionaries**. See the section "Adding User-specific Speller Dictionaries".

You can make a dictionary site-specific by adding a User-Property attribute to the Site object in the namespace with the global-name Spell-Dictionary and a value that will be parsed as a pathname. For more information, see the section "Attributes for Objects in the Namespace Database". Also see the section "Using the Namespace Editor".

There is a command processor command for creating a Speller dictionary including all the user names for your site.

Create Spell Dictionary From Namespace command

Create Spell Dictionary From Namespace *namespace pathname*

Creates a Speller dictionary with all the user-names, first names, and last names from the list of User objects in the namespace. (For speed, the command works by accessing the files that hold the namespace database rather than by accessing the actual namespace servers.)

namespace The namespace you wish the dictionary to represent.

pathname The pathname of the binary dictionary file to be created.

Once you have created the file, use the namespace editor to make it a site-specific dictionary. See the section "Adding Site-specific Speller Dictionaries".

Speller Dictionary Commands

Here are the commands for manipulating Speller Dictionaries.

Compile Spell Dictionary

Compile Spell Dictionary (m-X)

Compiles a text (character) dictionary into binary format for quicker loading. The command prompts for two pathnames. The first is the name of the character dictionary, which can be any text file. The second is the pathname of the binary dictionary. Binary dictionary files usually have a file type of "dict".

Read Spell Dictionary

Read Spell Dictionary (m-X)

Reads a dictionary from a file. By default, the dictionary is added to the menu. When a dictionary is on the menu you can add words to it as they are checked. With a numeric argument, the dictionary is not added to the menu. You are prompted for the pathname of the dictionary, which can be either character (text) or binary (compiled).

Show Spell Dictionaries

Show Spell Dictionaries (m-X)

Displays the list of dictionaries currently being used by the Speller.

Save Spell Dictionary

Save Spell Dictionary (m-X)

Writes an an updated version of a dictionary file to include all words added in the current session. You can save any dictionary from the list of dictionaries, but it is only meaningful to save a dictionary from the displayed list, since those are the only dictionaries being modified. Pathnames are unchanged. Saving an unmodified dictionary does nothing.

For the command to display the list of dictionaries, see the section "Show Spell Dictionaries".

See the section "Save All Spell Dictionaries".

Save All Spell Dictionaries

Save All Spell Dictionaries (m-X)

Saves updated copies of all modified dictionaries on the list of dictionaries. Dictionary pathnames are unchanged. Writes an an updated version of each dictionary file to include all words to that file added in the current session. The command saves only dictionaries from the displayed list, since those are the only dictionaries being modified. Pathnames are unchanged. Saving an unmodified dictionary does nothing.

For the command to display the list of dictionaries, see the section "Show Spell Dictionaries".

See the section "Save All Spell Dictionaries".

Kill Spell Dictionary

Kill Spell Dictionary (m-X)

Removes a dictionary from the list of dictionaries used by the Speller. This command has no effect on the dictionary file. Type in the pathname of the dictionary in name-first order (like Zmacs buffer names). Completion of dictionary names is available.

Add Word to Spell Dictionary

Add Word To Spell Dictionary (m-X)

Adds a word to a dictionary. The command prompts for both the word and the name of the dictionary. The word remains in the dictionary as long as the dictionary is read in.

You can also add a word at the time the Speller challenges it, by clicking on the dictionary name.

You must save the Spell dictionary to add the word to the dictionary file. See the section "Save Spell Dictionary", and

see the section "Save All Spell Dictionaries".

To add a word to a spell dictionary under program control, see the function **zwei:add-words-to-spell-dictionary**.

Delete Word From Spell Dictionary

Delete Word From Spell Dictionary (m-X)

Deletes a word from the on-line dictionary. This command prompts for both the word and the dictionary. If you wish to have the word permanently removed from the dictionary file, you must save the dictionary after executing this command. Otherwise, it will reappear the next time the dictionary is loaded. See the section "Save Spell Dictionary".

To delete a word from a dictionary under program control,

see the function **zwei:delete-words-from-spell-dictionary**.

Show Contents of Spell Dictionary

Show Contents Of Spell Dictionary (M-X)

Shows all the words in a dictionary. Completion of dictionary names is available.

With a numeric argument, the command prompts for a pathname and writes the words to that file. In this way, you can get a text file of a binary dictionary. You can also do this with the Execute Command Into Buffer (M-X) command, followed by Show Contents of Spell Dictionary.

To see the list of speller dictionaries, see the section "Show Spell Dictionaries", and see the section "Execute Command Into Buffer".

Speller Dictionary Functions

zwei:read-spell-dictionary *pathname* &optional (*menu-p* **t**) *Function*

Intended primarily for use in lispm-init files. It reads a dictionary from a file into virtual memory and adds it to the list of dictionaries used by the Zmacs Speller.

If the optional *menu-p* argument is **nil**, the dictionary is not added to the dictionaries shown on the menu. The default is to show the dictionary on the menu.

zwei:read-spell-dictionary overrides the auto-loading of standard dictionaries. If you use this function in your lispm-init file, you must also use **zwei:read-standard-spell-dictionaries** if you want the standard dictionaries loaded. See the function **zwei:read-standard-spell-dictionaries**.

See the section "Speller Dictionary Management".

zwei:read-standard-spell-dictionaries &key *for-general-use* *Function*

Reads the basic dictionary, any site-specific dictionaries, and, optionally, a user-specific dictionary into memory and makes them a part of the list of dictionaries used by the Zmacs Speller.

If the optional keyword argument *for-general-use* is **nil**, only the basic dictionary and any site-specific dictionaries are read in. The default is to read in those dictionaries plus the user's own user-specific dictionary. Only the user-specific dictionary appears on the menu.

You do not need to evaluate this function in your lispm-init file unless you are using **zwei:read-spell-dictionary**, which overrides the auto-loading of standard dictionaries, or unless you wish to set *for-general-use* to **nil**.

See the function **zwei:read-spell-dictionary**.

See the section "Speller Dictionary Management".

zwei:add-words-to-spell-dictionary *pathname list-of-words* &optional *ok-if-dictionary-not-found* *Function*

Adds words to a dictionary in virtual memory. Use it to patch a dictionary in a world load that already has the dictionary loaded. If the optional argument *okay-if-dictionary-not-found* is **t** and no dictionary is found, nothing happens. If the argument is *nil*, which is the default, the function signals an error.

For example,

```
(zwei:add-words-to-spell-dictionary "shoebox:>fred>arfnarf.dict"
       '("roadhog" "birdbrain") t)
```

This function is also available from a command, see the section "Add Word to Spell Dictionary".

See the function **zwei:delete-words-from-spell-dictionary**.

The dictionary file on disk is not affected. To add a word to a dictionary file, you must save the dictionary.

See the section "Save Spell Dictionary".

See the section "Save All Spell Dictionaries".

zwei:delete-words-from-spell-dictionary *pathname list-of-words* &optional *ok-if-dictionary-not-found* *Function*

Deletes words from a dictionary in virtual memory. Use it to patch a dictionary in a world that already has the dictionary loaded. If the optional argument *okay-if-dictionary-not-found* is **t** and no dictionary is found, nothing happens. If the argument is **nil**, which is the default, the function signals an error.

For example,

```
(zwei:delete-words-from-spell-dictionary "bogue:>lorna>snage.dict"
       '("wonderissimo" "megastupidity") t)
```

This function is also available from a command, see the section "Delete Word From Spell Dictionary".

See the function **zwei:add-words-to-spell-dictionary**.

The dictionary file on disk is not affected. To delete a word from a dictionary file, you must save the dictionary.

See the section "Save Spell Dictionary".

See the section "Save All Spell Dictionaries".

Editing Lisp Programs

Genera programmers develop programs in repeated cycles, each a sequence of editing, compiling, testing, and debugging. These cycles are often nested. Zmacs allows you to edit and test large programs dynamically, without frequent file system operations.

As a programmer using Genera you typically read a file containing Lisp code into an editor buffer, make modifications, test the results, make more changes, and so on, until satisfied with the behavior of the program. Only then do you need to write the buffer back out to the file system. The debugging loop is much tighter and more responsive than in traditional programming environments. You can even evaluate Lisp forms directly from inside the editor, without returning to a Lisp Listener. Alternatively, you can divide the screen into a Lisp Listener window and a Zmacs window, so that you can direct your attention to either without changing the display.

Zmacs provides extensive features for locating source code of specified functions. If an error occurs, the Debugger can cause Zmacs to read in the source of the function that got the error. You can then debug and recompile the function.

When you edit a file with a Lisp type, Zmacs puts that buffer into Lisp mode. A command exists for explicitly placing a buffer in Lisp mode:

Lisp Mode (m-X) Lisp Mode

Places the current buffer into Lisp mode.

Base and Syntax Default Settings for Lisp

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Common-Lisp. See the section "Buffer and File Attributes in Zmacs".

Commenting Lisp Code

Zmacs differentiates between the different comment indicators for different major modes. Comments in Lisp begin with a semicolon. The Lisp reader ignores everything between a (significant) semicolon and the next newline. By convention, there are three kinds of comments, beginning with one, two, and three semicolons:

- Comments beginning with a single semicolon are placed to the right of a line of code, start in a preset column (the *comment column*), and describe what is going on in that line.

- A comment with two semicolons is a long comment about code within a Lisp expression and has the same indentation as the code to which it refers. It describes the function of a group of lines.

- A comment headed by three semicolons is normally placed against the left margin, and describes a large piece of code, such as a function or group of functions.

This section outlines Lisp commenting conventions and explains Zmacs commands for manipulating comments.

Indenting for Comment

`c-;` Indent For Comment

`m-;`

If the current line has no comment, moves point out to the comment column (inserting spaces to get there, if necessary) and starts a comment by inserting a semicolon there. If the current line already has a comment, it indents it correctly and leaves point at the beginning of it. Zmacs positions the various kinds of comments appropriately. If a comment begins at the left margin, it leaves it there.

With a numeric argument *n*, it realigns any comments on the next *n* lines, including the current line, but does not create any new comments.

If a comment cannot be positioned at the comment column because the associated line of code is too long, comments are moved to the right until they are clearly separated from the code.

Killing a Comment

`c-m-;` Kill Comment

Kills any comments in the region. If no region exists, kills comments on the current line if no region exists. This command can be reversed with Undo if no other undoable command has intervened.

Moving Down to Comment on Next Line

`m-N` Down Comment Line

Moves point to the beginning of the comment on the next line. If there is no comment on the next line, it creates one. If the comment on the current line is empty, it deletes it before going to the next line.

With a numeric argument *n*, it moves point to the beginning of the comment on the *n*th line after the current one.

Moving Up to Comment on Previous Line

`m-P` Up Comment Line

Moves point to the beginning of the comment on the previous line. If there is no comment on the previous line, it creates one. If the comment on the current line is empty, it deletes it before going on to the previous line.

With a numeric argument *n*, it moves point to the beginning of the comment on the *n*th line before the current one.

Setting the Comment Column

`c-X ;` Set Comment Column

Sets the comment column to be the current horizontal position of the cursor.

With a numeric argument, it finds the nearest comment above the current line, sets the comment column to line up with that comment, and actually puts a comment on the current line at that column.

Creating a New Indented Comment Line

`m-LINE` Indent New Comment Line

Makes a new blank line after the current line and starts a new comment there, indented properly. If there was already a comment on the current line, the comment on the new line is of the same kind. (That is, it has the same number of semicolons and is indented the same.) If there was no comment on the starting line, `m-LINE` starts a new line, indenting the new line as appropriate for the major mode.

Inserting and Removing Lisp Comments from Regions

`c-X c-;` Comment Out Region

Comments out each line in the region. When the region ends at the beginning of a line, it does not comment out that line. If any part of the line is part of the region, then it does comment out that line.

With a numeric argument (`c-U c-X c-;`) the command restores lines in the region that have been commented out. When any part of the line is part of the region, comments are removed from around that line.

The removal works the same as the commenting out. That is, a single semi-colon (;) in column 1 is removed. Removal of comments stops at a line without a semi-colon in column 1, even if more lines that have been commented out remain in the region. The rest of the region does remain in this case, so that you can resume.

Evaluating and Compiling Lisp Programs

The commands in this section form a link between the Zmacs editor and the Lisp language. They allow the evaluation and compilation of code from Zmacs buffers. These commands are an important part of the debugging loop.

When a Lisp form is being compiled or evaluated, the editor displays a message that classifies what is being compiled.

It classifies macros as functions (because these go in the function cell of a symbol). For example:

```
Compiling Function SUN
Evaluating Variable MARS
Compiling Flavor STAR
```

Evaluating Lisp Programs

m-ESCAPE — Evaluate Minibuffer

Evaluates expressions from the minibuffer. You enter Lisp expressions in the minibuffer, which are evaluated when you press END. The value of the expression itself appears in the echo area. If the expression displays any output, that appears as a typeout window.

Evaluate Into Buffer (m-X)

Evaluates an expression read from the minibuffer and inserts the result into the buffer. You enter a Lisp expression in the minibuffer, which is evaluated when you press END. The result of evaluating the expression appears in the buffer before point. With a numeric argument, it also inserts any typeout that occurs during the evaluation into the buffer.

Evaluate Buffer (m-X)

Evaluates the entire buffer. The result of evaluating the buffer appears in the minibuffer. With a numeric argument, it evaluates from point to the end of the buffer.

Evaluate Region (m-X)
c-sh-E

Evaluates the region. When no region has been defined, it evaluates the current definition. It shows the results in the echo area.

c-m-sh-E — Evaluate Region Verbose

Evaluates the region. When no region has been defined, it evaluates the current definition. It shows the results in a typeout window.

Evaluate Region Hack (m-X)

Evaluates the region, ensuring that any Lisp variables appearing in a **defvar** have their values set. When no region has been defined, it evaluates the current definition. It shows the results in the echo area.

Evaluate Changed Definitions (m-X)

Evaluates any definitions that have changed in any of the current buffers. With a numeric argument, it prompts individually about whether to evaluate particular changed definitions (the default evaluates all changed definitions).

Evaluate Changed Definitions of Buffer (m-X)
m-sh-E

Evaluates any definitions that have changed in the current buffer. With a numeric argument, it prompts individually about whether to evaluate particular changed definitions (the default evaluates all changed definitions).

Evaluate And Replace Into Buffer (m-X)

Evaluates the Lisp object following point in the buffer and replaces it with its result.

c-m-Z Evaluate And Exit

Evaluates the buffer and exits Zmacs. It selects the window from which the last **ed** function or the last Debugger c-E command was executed.

Compiling Lisp Programs

Compile Buffer (m-X)

Compiles the entire buffer. With a numeric argument, it compiles from point to the end of the buffer. (This is useful for resuming compilation after a prior Compile Buffer has failed.)

Compile Changed Definitions (m-X)

Compiles any definitions that have changed in any of the current buffers. With a numeric argument, it prompts individually about whether to compile particular changed definitions (the default compiles all changed definitions).

Compile Changed Definitions of Buffer (m-X)
m-sh-C

Compiles any definitions that have changed in the current buffer. With a numeric argument, it prompts individually about whether to compile particular changed definitions. The default is to compile all changed definitions.

Compile Changed Definitions of Tag Table

Compile Changed Definitions of Tag Table (m-X)

Compiles any definitions that have changed in any of the buffers in the current tag table. With a numeric argument, the command prompts individually about whether to compile particular changed definitions. The default is to compile all changed definitions.

Compile File (m-X)

Compiles a file, offering to save it first (if it has an associated buffer that has been modified). It prompts for a file name in the minibuffer, using the file associated with the current buffer as the default. It does not load the file.

Load File (m-X)

Loads a file, possibly saving and compiling it first. It prompts for a file name, taking the default from the current buffer. It checks to see if the file you are compiling corresponds to a buffer and offers to save that buffer if it is modified. If the

.bin file is older than the .lisp file, it offers to compile the file first. If the typeout window displays any compiler warnings, Load File asks if you really want to load the file despite the compiler warnings.

m-Z Compile And Exit

Compiles the buffer and exits Zmacs. It selects the window from which the last **ed** function or the last debugger c-E command was executed.

Lisp Compiler Warnings

Compiler warnings are kept in an internal database that you can inspect and manipulate in various ways with several editor commands.

Compiler Warnings (m-X)

Creates the compiler warnings buffer (called *Compiler-Warnings-1*) if it does not exist, puts all outstanding compiler warnings in that buffer, and switches to that buffer. You can view the compiler warnings by scrolling around and doing text searches through them using Edit Compiler Warnings (m-X).

Edit Compiler Warnings (m-X)

Prompts you with the name of each file mentioned in the database, allowing you to edit the warnings for that file. It then splits the Zmacs frame into two windows: the upper window displays a warning message and the lower one displays the source code whose compilation caused the warning. After you have finished editing each function, c-. gets you to the next warning: the top window scrolls to show the next warning and the bottom window displays the function associated with this warning. Successive uses of c-. take you through all of the warning messages for all of the files you specified. When you are done, the last c-. puts the frame back into its previous configuration.

Edit File Warnings (m-X)

Asks you for the name of the file whose warnings you want to edit. You can give either the source file or the compiled file. Only warnings for this file are edited. If the database does not have any entries for the file you specify, the command prompts you for the name of a file that contains the warnings, in case you know that the warnings are stored in another file.

Load Compiler Warnings (m-X)

Loads a file containing compiler warning messages into the warnings database. It prompts for the name of a file that contains the printed representation of compiler warnings. It always replaces any warnings already in the database.

Parenthesizing Lisp Expressions

m-(Make ()

Inserts matching parentheses, leaving point between them. With a numeric argument *n*, it encloses the next *n* Lisp expressions in parentheses. When the number

of expressions requested cannot be satisfied, it beeps and does nothing. With point on the open parenthesis of a **defun**, an argument of 1 encloses the whole **defun** within a new set of parentheses. Any argument larger than 1 would have no effect. In Text Mode, a word or a phrase within parentheses is treated as a Lisp form.

See also the description of the command m-) (see the section "Motion Among Top-Level Expressions".) Matching parentheses in Zmacs Lisp buffers flash in both directions.

The matching open parenthesis flashes when the cursor is sitting just past a close parenthesis and the matching close parenthesis flashes when the cursor is sitting on an open parenthesis.

Expanding Lisp Expressions

Two editor commands allow you to expand macros: Macro Expand Expression and Macro Expand Expression All.

c-sh-M — Macro Expand Expression

Reads the Lisp expression following point and expands the form itself but not any of the subforms within it. It displays the result in the typeout window. With a numeric argument, it pretty-prints the result back into the buffer immediately after the expression.

m-sh-M — Macro Expand Expression All

Reads the Lisp expression following point, and expands all macros within it at all levels. It displays the result in the typeout window. With a numeric argument, it pretty-prints the result back into the buffer immediately after the expression. It assumes that every list in the expression is a form, so if car of a list is a symbol with a macro definition, the purported macro invocation is expanded.

Locating Source Code to Edit

The functions that make up a program or system can depend on each other in complicated ways. When you are editing one function, you sometimes have to go off and look at another function, and possibly modify that one too.

This section describes the Edit Definition command and other commands that list and/or edit various sets of definitions. In addition, two pairs of List and Edit commands help identify changed code by finding or editing *changed* definitions in buffers. By default, the *changed* commands find changes made since the file was read; use numeric arguments to find definitions that have changed since they were last compiled or saved.

The Zmacs Edit Definition Commands

Edit Definition (m-.) is a powerful command to find and edit function definitions, macro definitions, global variable definitions, and flavor definitions. In general,

Zmacs treats as a definition any top-level expression having in functional position a symbol whose name begins **def**.

It is particularly valuable for finding source code, including system code, that is stored in a file other than that associated with the current buffer. It finds multiple definitions when, for example, a symbol is defined as a function, a variable, and another type of object. It maintains a list of these definitions in a support buffer.

Zmacs Command: m-.

This command is one of the most valuable tools of the system. When you are developing or debugging programs, you can use m-. to find the definition of an ordinary function, generic function, flavor, method, variable, package, or other type of definition. Completion is supported on the definition, if it is already in an editor buffer.

m-. prompts for a definition to find. You can enter a large variety of representations, and m-. figures out what definition you are seeking. For example, you can enter symbols with or without package prefixes.

You can provide any of the following responses to the m-. prompt:

symbol — Finds the definition of *symbol*, which can be an ordinary function or generic function. For generic functions, the **defgeneric** form is found if one exists; all existing methods are also found. *symbol* can also be one of: variable, package, **defstruct** structure, flavor, or other types of definitions.

(*generic-function flavor*) — Finds the definitions of one method that implements *generic-function* on instances of *flavor* and asks if you mean that method. If not, it proceeds to find other methods, including special-purpose methods such as **:before**, **:after**, **:default**, and so on.

(*symbol property*) — Finds the function named by function spec (*:property symbol property*). This is a handy abbreviation.

function-spec — Finds the definitions of *function-spec*. For example, you could enter **(flavor:method change-status cell)** to find the method of that function spec. Often it is more convenient to enter the list **(change-status cell)** instead.

When the requested Lisp object has multiple definitions, one of them is displayed. You can then use c-U m-. to cycle through the other definitions. Also, a list of all definitions and the files they are located in is stored in a buffer called **Definitions-n**. The position of the cursor in that buffer controls where c-U m-. will go next.

You can also point at forms with the mouse, in a buffer or in other windows, and click m-Left to edit the definition.

Example of the m-. Command

Suppose you are modifying a function called **sun**, which was written by someone else. **sun** calls the unfamiliar **luna**, and you need to find out what **luna** does before proceeding. Use m-. to peek at the definition of **luna**.

When you type m-., Zmacs prompts you for the name of a definition. If point is in the expression where **luna** is called, the default name is **luna**, and you need only press END. If point is somewhere else and the default is wrong, you can point at the word **luna** with the mouse or you can type it in. To let you know that you can define a name with the mouse, the mouse cursor changes to an arrow pointing straight up. All the symbols that are names of definitions you could specify become mouse sensitive.

Edit Installed Definition (m-X)

Edits the installed version of the file that contains the definition of a specified Lisp object. It prompts for the name of the definition; if one of your buffers already contains the installed version of that definition, it selects that buffer. Otherwise, it reads in the source file that contains the definition. It always positions the cursor in front of the definition. When the object has more than one definition, use a numeric argument to edit another definition of the same object. You can repeat this until there are no more definitions of that object.

Edit Changed Definitions (m-X)

Determines which definitions in any Lisp Mode buffer have changed and selects the first one. It makes an internal list of all the definitions that have changed since the buffer was last saved and selects the first one on the list. Use c-. (Next Possibility) to move to subsequent definitions. See the section "Displaying the Next Possibility".

Edit Changed Definitions accepts a numeric argument to control the time point for determining what has changed:

Value	*Meaning*
1	For each buffer, since the file was last saved (the default).
2	For each buffer, since the buffer was last read.
3	For each definition in each buffer, since the definition was last compiled.

Edit Changed Definitions of Buffer (m-X)

Determines which definitions in the current buffer have changed and selects the first one. It makes an internal list of all the definitions that have changed since the buffer was last saved and selects the first one on the list. Use c-. (Next Possibility) to move to subsequent definitions. See the section "Displaying the Next Possibility".

Edit Changed Definitions of Buffer accepts a numeric argument to control the time point for determining what has changed:

Value	*Meaning*

1 Since the file was last saved (the default).

2 Since the buffer was last read.

3 Since the definition was last compiled.

Edit Cp Command

Edit CP Command (m-X)

Reads the source of a CP command into an editor buffer. With a numeric argument, prompts for a comtab. Otherwise it looks for the command in the global comtab.

Edit System Files

Edit System Files (m-X)

Reads all of the files of a system into buffers. With a numeric argument, the files of the component system are also read into buffers.

The List Definition Commands

List Definitions (m-X)

Displays the definitions in a specified buffer. It reads the buffer name from the minibuffer, using the current buffer as the default. It displays the list as a typeout window. The individual definition names are mouse sensitive.

List Duplicate Definitions (m-X)

Displays the duplicate definitions in the current buffer, if any. This is especially useful for checking patch files or files made by merging several programs together. c-. (Next Possibility) moves point to duplicate definitions that occur earlier in the file, beginning with the earliest duplicate and not including the latest duplicate. See the section "Displaying the Next Possibility".

List Changed Definitions (m-X)

Displays a list of any definitions that have been edited in any buffer. Use c-. (Next Possibility) to start editing the definitions in the list. See the section "Displaying the Next Possibility".

List Changed Definitions accepts a numeric argument to control the time point for determining what has changed:

Value *Meaning*

1 For each buffer, since the file was last read (the default).

2 For each buffer, since the buffer was last saved.

3 For each definition in each buffer, since the definition was last compiled.

List Changed Definitions of Buffer (m-X)

Displays the names of definitions in the buffer that have changed. It makes an internal list of the definitions changed since the buffer was read in and offers to let you edit them. Use c-. (Next Possibility) to move to subsequent definitions. See the section "Displaying the Next Possibility".

List Changed Definitions of Buffer accepts a numeric argument to control the time point for determining what has changed:

Value	*Meaning*
1	Since the file was last read (the default).
2	Since the buffer was last saved.
3	Since the definition was last compiled.

The Edit Callers Commands

When you are modifying a large system, you often have to make sure that changing a function does not render unusable other functions that call the modified one. Zmacs provides facilities for editing the sources of all the functions defined in the current world that call a given one. This removes some of the unpleasantness of making incompatible changes to large programs and is a good example of how Zmacs interacts with the Lisp environment to make programming easier.

Edit Callers (m-X)

Prepares for editing all functions that call the specified one. The prompt is the same kind that Edit Definition gives you. It reads a function name via the mouse or from the minibuffer with completion.

It selects the first caller; use c-. (Next Possibility) to move to a subsequent definition. See the section "Displaying the Next Possibility".

Multiple Edit Callers (m-X)

Prompts for the names of a group of functions and edits those functions in the current package that call *any* of the specified ones. It reads a function name from the minibuffer, with completion, initially offering a default function name. It continues prompting for more function names until you end the list with RETURN.

Use m-X Multiple Edit Callers Intersection when you want to edit those functions that call *all* of the specified functions. See the section "Multiple Edit Callers Intersection".

Multiple Edit Callers Intersection

Multiple Edit Callers Intersection (m-X)

Prepares for editing all the functions that call *all* of the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for a function name until you end it with just a carriage return.

Use m-X Multiple Edit Callers if you want to edit all the functions that call *any* of the specified functions. See the section "Multiple Edit Callers ".

List Callers (m-X)

Prompts for the name of a function exactly the way Edit Callers does, but instead of editing the callers in the current package of the specified function, it simply displays their names. The names are mouse-sensitive. If you point at one and click left, you can edit the source of that caller. If you click right, a menu pops up that offers to give the argument list of the selected caller, to disassemble it, to edit it, or to see its documentation string. In addition, c-. ("Next Possibility") works in this context, offering the first caller to be edited, and queuing up the other callers to be edited in sequence.

Multiple List Callers (m-X)

Lists all the functions that call *any* of the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for more function names until you end the list with RETURN.

The list of function names is mouse-sensitive: see List Callers (m-X). c-. (Next Possibility) edits the callers. See the section "Displaying the Next Possibility".

Use m-X Multiple List Callers Intersection when you want only callers that call *all* the specified functions. See the section "Multiple List Callers Intersection".

Multiple List Callers Intersection

Multiple List Callers Intersection (m-X)

Lists all the functions that call *all* of the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for a function name until you end it with just a carriage return.

Use m-X Multiple List Callers if you want to list functions that call *any* of the the specified functions. See the section "The Zmacs Edit Callers Commands".

Patching

For complete information about patching, see the section "Patch Facility".

During a typical maintenance session you might make several changes to existing definitions or write new ones. Rather than recompiling the entire system every time you change a source file, you can copy only the new or revised code into a *patch file* and write the file ("finish" the patch). Whenever you finish a patch, the patch facility automatically compiles the file and records the event in a "patch registry" for the system, noting the number of the patch, the system being patch, and a brief user-supplied description. As soon as a user loads the patch file (after the system is loaded), the state of the given system in the user's machine is presumably the same as in the developer's machine when the patch was finished.

The patch facility allows you to have several patches in progress at once. Thus you can patch several different systems or several different minor versions of the same

system during one work session. The patch facility manages this potentially dangerous situation in the following way. Every time you start a patch, a number and a place in the patch registry is reserved for the patch in production. The patch is marked *in-progress*. When the patch is finished, the entry is completed and the in-progress mark removed. If you decide to abort the patch, the registry entry is automatically deleted.

The ability to have more than one patch in-progress to more than one system makes it imperative that you keep track of the state of your various patches. If a patch is left unfinished (unwritten), the **load-patches** function will load neither the in-progress patch nor any subsequent finished patches.

The patch facility considers patches to be active or inactive and in one of the following states: initial, in-progress, aborted, or finished. Show Patches (m-X) displays the state of all patches started in this work session. If more than one patch is in progress, one of them is known as the *current patch*. The commands that add patches, like Add Patch (m-X), add only to the patch considered by the patch facility to be the current patch. The command Select Patch (m-X) displays a menu of active patches and allows you to make another patch the current one.

In general you should adhere to the following steps in making a patch. It is assumed that your system is patchable; that is, the **:patchable** option appears in the system declaration.

1. You must load (via **load-system**) the major version of the system that you want to patch.
2. Read in the source files you want to edit into a Zmacs buffer. Make all changes and test them thoroughly. Write the source file.
3. Use the appropriate Zmacs commands to make your patch. Begin the patch, using Start Patch (m-X).
4. Add the changed code to the patch buffer by using Add Patch (m-X), Add Patch Changed Definitions of Buffer (m-X), or Add Patch Changed Definitions (m-X).
5. Finish the patch, using Finish Patch (m-X), or abort the patch, using Abort Patch (m-X).

Commands provided for initiating a patch are Start Patch (m-X), Start Private Patch (m-X), and Add Patch (m-X).

Start Patch (m-X)

Starts a new patch, prompting you for the name of the system to be patched; it must be a system currently loaded. It assigns a new minor version number for that particular system by writing a new version of the patch directory file with an entry for that minor version number. The patch is marked as in-progress. It starts constructing the patch file in an editor buffer, but does not select the buffer.

While you are making your patch file, the minor version number that has been allocated for you is reserved so that nobody else can use it. Thus, if two people are patching the same system at the same time, they cannot be assigned the same minor version number.

The command does not actually move any definitions into the patch file. You must explicitly do so with Add Patch Changed Definitions of Buffer (m-X), Add Patch Changed Definitions (m-X), or Add Patch (m-X).

The patch facility permits you to start another patch before finishing the current one. However, if your new patch is to the same system, the patch facility warns you that you already have a patch in progress and allows you to take one of four actions:

- Abort the in-progress patch and start a new patch.
- Finish the in-progress patch and start a new patch.
- Proceed with the second patch (initial patch) for this system and leave the in-progress patch intact.
- Use the existing buffer and do not start a new patch.

Start Private Patch (m-X)

Although similar to Start Patch (m-X), Start Private Patch (m-X) does not have any relationship to systems, major and minor version numbers, and official patch directories. Rather it allows you to make a private patch file that you can load, test, and share with other users before you install a numbered patch that is automatically available to all users.

Instead of prompting for a system name, the command prompts for a file name. It also prompts for a patch note to be saved with the patch. The default for this private patch note is the same as the name component of the private patch pathname, except that spaces are converted to hyphens. This patch note is also offered as the subject line of a mail message if you select **yes** for *Send mail about this patch* in the Finish Patch menu.

Start Private Patch does not actually move any definitions into the patch file. Use Add Patch Changed Definitions of Buffer (m-X), Add Patch Changed Definitions (m-X), or Add Patch (m-X) to insert the code. Finishing the patch (using Finish Patch (m-X)) writes it out to the specified file.

Note: Use the Load File command or Load File (m-X) to load a private patch; the Load Patches command and the **load-patches** function do not load private patches.

Add Patch (m-X)

Starts a new patch if none is underway, prompts you for a system name, and inserts the region or current definition into the patch buffer. If a patch was in progress, Add Patch (m-X) just adds the region or current definition to the current patch file.

Warns you if your editor buffer conflicts with the system being patched. If you mistakenly use Add Patch on code that does not work, select the buffer containing the patch file and delete it. Then later you can use Add Patch (m-X) on the corrected version. For each patch you add, it queries for a patch comment, which it then inserts in the patch file. Just pressing END means "no comment".

Add Patch (m-X), Add Patch Changed Definitions (m-X), or Add Patch Changed Definitions of Buffer (m-X) insert code into the patch file. If the patch is being made to the system the current buffer's file came from, the commands proceed.

If there is a current patch, and it is not appropriate for the system that the buffer's file came from, you see a menu showing all of the current patches, plus an option to create a new patch appropriate for the buffer, plus an option to abort.

Add Patch Changed Definitions of Buffer (m-X)

Add Patch Changed Definitions of Buffer (m-X) selects each definition that was changed in the buffer and asks you whether or not you want the definition patched.

For each definition, you can respond as follows:

Response	*Action*
Y	Patches the definition.
N	Skips the definition.
P	Patches the definition and any additional modified definitions in the same buffer without asking any more questions.

A definition needs to be patched if it has been changed since it was last patched or if it has not been patched since the file was read into the buffer.

For each patch you add, it queries for a patch comment, which it then inserts in the patch file. Just pressing END means "no comment".

Add Patch Changed Definitions (m-X)

Add Patch Changed Definitions (m-X) selects a buffer in which definitions were changed and asks whether or not you want to patch the changed definitions. Answering N skips the buffer and proceeds to the next buffer of the same mode, if any. Answering Y selects each definition that has changed in that buffer and asks you whether or not you want the definition patched. For each definition, you can respond as follows:

Response	*Action*
Y	Patches the definition.
N	Skips the definition.

P Patches the definition and any additional modified definitions in the same buffer without asking any more questions; when done, it proceeds to the next buffer.

If there are more buffers containing definitions to be patched, it asks questions again when it gets to the next buffer.

A definition needs to be patched if it has been changed since it was last patched and if it has not been patched since the file was read into the buffer.

For each patch you add, it queries for a patch comment, which it then inserts in the patch file. Just pressing END means "no comment".

Add Patch Changed Definitions selects buffers based on the mode of the buffer from which the command is issued. Thus if you are in a Lisp mode buffer, any Lisp mode buffers with definitions to be patched are offered, and if you are in another mode buffer, only buffers of that mode are offered.

When making multiple patches during one work session use the Select Patch and Show Patches commands to keep track of patches.

Select Patch (m-X)

When you are making more than one patch during a work session, Select Patch (m-X) allows you to choose a different patch as the current patch from a menu of active patches. The patching commands (like Add Patch and Add Patch Changed Definitions of Buffer) insert definitions into the patch file that you have selected as the current patch. To insert patch definitions into another buffer, use Select Patch to choose that buffer as the current patch.

Show Patches (m-X)

Show Patches (m-X) displays the state of all patches started in this session. Patches are either active or inactive and can be in one of the following states: initial, in-progress, aborted, or finished. *Inactive patches* are in an aborted or finished state. *Active patches* are in an initial or in-progress state. *Initial* means that the patch buffer has been initialized but as yet no definitions have been added to the buffer. *In-progress* means that the patch buffer has been initialized and definitions have been added to the buffer.

Show Patches groups the active and inactive patches and identifies the current patch.

After making and testing all of your patches, use the Finish Patch command to install the patch in the system.

Finish Patch (m-X)

Finish Patch (m-X) installs the patch file so that other users can load it. This command saves and compiles the patch file (patches are always compiled). If the compilation produces compiler warnings, the command asks whether or not you want to finish the patch anyway. If you do, or if no warnings are produced, a new version of the patch directory file is written. The in-progress mark is removed from the entry in the patch registry.

The command allows you to edit the patch comments, which are written to the patch directory file. (**load-patches** and **zl:print-system-modifications** print these comments.) It then asks you whether you want to send mail about the patch. If you say "yes", it opens a mail buffer and inserts initial contents, including the name of the patch file and your patch comment.

Note: By default the Finish Patch command queries you about sending mail. You can alter this behavior by changing the value of the variable **zwei:*send-mail-about-patch***. Its valid values are **:ask**, the default value, which queries the user; **t**, which opens a Zmacs mail buffer without querying; and **nil**, which takes no action regarding the sending of patch mail.

The Finish Patch menu lists the modified source files for the patch and offers to save them as part of the Finish Patch process. If you do not save your files as part of the Finish Patch process, Finish Patch displays a reminder to save your files when it finishes writing the patch directory. You can set the variable **zwei:*finish-patch-save-sources-default*** (default **nil**) to **t** in your init file to have Finish Patch save your files automatically.

Sometimes you start making a patch file and for a variety of reasons do not finish it — for example, you decide to abort the patch, you need to end your work session at this machine, or your machine crashes. In each of these situations it is of the utmost importance that you leave the patch directory file in a clean state; that is, either go back and finish the patch (as soon as possible!) or deallocate the patch number reserved to you. Failure to do so has unfortunate consequences: users at your site will not be able to load patches.

If your machine has crashed, use Resume Patch (m-X) to reclaim access to the patch number previously assigned to you. You can continue with the patch (assuming you saved the source files just prior to the crash) or use Abort Patch (m-X) to deallocate the patch number. Begin the patch again if you wish. If you simply decide to abandon the patch file, then just use Abort Patch. If you must boot your machine before finishing the patch, then save the patch buffer and as soon as possible use Resume Patch to read in the relevant patch file; finish the patch or abort it, as you wish.

Abort Patch (m-X)

Abort Patch (m-X) deallocates the minor version number that was assigned by the Start Patch or Add Patch commands. It tells Zmacs that you are no longer interested in making the current patch and offers to kill the patch buffer. The next time you do Add Patch (m-X), Zmacs starts a new patch instead of appending to the one in progress.

Resume Patch (m-X)

Resume Patch (m-X) allows you to return to a patch that you were not able to finish in the same boot session in which you started it; for example, your machine might have crashed or you had to boot your machine suddenly. It reads in the relevant patch file if it was previously saved; otherwise it just reclaims your access to the minor version number allocated to you when you started the patch. Abort or finish the patch.

Under certain circumstances you might find it necessary to recompile and reload a patch file.

Recompile Patch (m-X)

Recompile Patch (m-X) recompiles an existing patch file. This command is useful when, for example, an existing patch needs to be edited or a compiled patch file becomes damaged in some way. Never recompile a patch manually or in any way other than using the Recompile Patch command. This command ensures that source and object files are stored where the patch system can find them.

Use Recompile Patch with caution! Recompiling a patch that has already been loaded by other users can cause divergent world loads.

Reload Patch (m-X)

Reload Patch (m-X) reloads an existing patch file. This command makes it easy to reload a patch file without having to know its pathname.

You might want to have your herald announce private patches that you make. **note-private-patch** adds a private patch to the database in your world and includes the name of the patch in the herald.

note-private-patch *string* *Function*

Adds a private patch to the database in your world. **note-private-patch** takes a *string* argument. For example, the following adds the private patch called patch.lisp:

```
(note-private-patch "s:>maria>patch.lisp")
```

Subsequent displays of your herald show the inclusion of that patch in your world.

You create private patches using the Start Private Patch (m-X) command and then the standard patch commands for adding to and finishing the patch. Use the Load File command or Load File (m-X) to load a private patch; the Load Patches command and the **load-patches** function do not load private patches.

sct:require-patch-level-for-patch &rest *system-major-minor-specs* *Function*

Enforces a patch's dependency on some particular patch level in another system or systems. It is used at the head of any patch file that requires a certain patch level in some other system to load or operate correctly. For example:

```
(sct:require-patch-level-for-patch '(system 357. 510.) '(tape 69. 10.))
```

If the patch level requirements are not all met, loading the patch (and subsequent patches for that system) is skipped. After patches are loaded for all systems, Load Patches is called again to see if the patch levels of the other systems are now high enough to permit loading of additional patches. You can specify the patch level requirements from the Finish Patch menu.

zwei:*send-mail-about-patch* *Variable*

Controls whether the Finish Patch menu question "Send mail about this patch?" comes up set to **yes**. The possible values are **t**, **nil**, or **:ask**. The default is **:ask**. If the value is **t**, the question always appears with **yes** set.

zwei:*finish-patch-save-sources-default* *Variable*

Controls whether the Finish Patch menu question "Save sources for this patch?" comes up set to **yes** or **no**. The possible values are **t** or **nil**. The default is **nil**. If the value is **t**, the question always appears with **yes** set.

Customizing the Zmacs Environment

Now that you are familiar with the basic Zmacs concepts and techniques, you can set up a large set of minor modes, Zmacs and Lisp variables, and parameters to change the way the editor works. Zmacs's flexibility allows you to change which keys are connected to which commands, write your own commands, and install them in lieu of the standard system commands. A few users make extremely radical changes to the point where almost every key has a new meaning.

This section describes:

- Zmacs minor and major modes, and how they provide a degree of customization
- Creating new commands with keyboard macros
- Saving keyboard macros
- Setting key bindings
- Specifying Zmacs variable settings
- Sample init file forms for automatically reloading your customized environment

Built-in Customization Using Zmacs Minor Modes

Definition of Zmacs Minor Modes

A *minor mode*:

- Is an option.
- Is independent of other minor modes and of the selected major mode.

Zmacs has an extended command for each minor mode (m-X) that turns the mode on or off. With no argument, the command turns the mode on if it was off and off if it was on. This is known as *toggling*. A positive argument always turns the mode on, and a zero argument or a negative argument always turns it off.

All the minor mode commands are suitable for connecting to single- or double-character commands if you want to enter and exit a minor mode frequently. See the section "Zmacs Key Bindings".

For information about setting minor modes permanently, see the section "Setting Mode Hooks in Init Files".

Example

Auto Fill Mode (m-X)

Turns on *Auto Fill Mode*, a minor mode that inserts Return characters automatically to break lines as you type. You can turn Auto Fill Mode on regardless of your major mode. If the mode line displays Fill, Auto Fill Mode is on. If Auto Fill Mode is already turned on, this command turns it off.

This mode is useful when you are typing large amounts of text. It makes it unnecessary to look at the screen or to worry about line length: you just type in the text without newlines and Zmacs inserts them whenever they are needed.

Auto Fill Mode works by establishing a hook that runs after you press one of the activation characters (SPACE, RETURN, ., ?, !, or]) that activate filling in this mode. When you press one of these characters in Auto Fill Mode, Zmacs does more than simply insert it. First it checks to see whether the line exceeds the maximum allowable line length or *fill column* (see Set Fill Column below). If the line is too long, Zmacs finds the last word on the current line that fits inside the fill column. Zmacs then inserts a newline right after that word. Extra spaces (if any) are deleted from the beginning of the newly formed line.

Because of the way Auto Fill Mode works, you will often find yourself typing a word out beyond the fill column. The word will be moved to the next line as soon as you press one of the activation characters.

The fill column is used by Auto Fill Mode (and by the paragraph adjusting commands) to decide where to break lines. It is measured in pixels, not in characters, so that Auto Fill Mode works even if characters of different widths appear in a buffer. (A *pixel* is a tiny rectangular area on the screen that is either all white or

all black. Pixels are the smallest addressable region of the display. If you look closely, you can see the separate rectangular pixels that make up everything on the display.)

You can change the fill column with the following command:

`c-X F` Set Fill Column

Changes the fill column to match up with the current position of the cursor. That means that if point is at the end of a line, filled lines will not be longer than the current one from now on. You are given a choice of whether to set the fill column for the buffer, the major Zmacs mode, or globally.

With a positive numeric argument *n* less than 200, the fill column is set to be *n* character-widths, and if *n* is 200 or greater, the fill column is set to be *n* pixels.

In effect, this command sets the right margin. Auto Fill mode and the `m-Q` use the fill column setting.

Summary of Minor Modes

Atom Word Mode (`m-X`)

Makes word commands, such as `m-D` and `m-F`, work on Lisp objects, not words, when in Lisp mode. For example, in Atom Word Mode, `m-F` would move all the way across `:show-machine-name-in-wholine`, while with Atom Word Mode off (the default), `m-F` would move only from `show` to `machine`. Atom Word Mode ignores lists and **nil**.

This command does not display anything in the mode line. Turn off Atom Word Mode by issuing the command again.

Auto Fill Lisp Comments Mode (`c-m-X`)

Turns on auto filling of comments, but not code. This command displays `Fill-Comments` in the mode line.

Auto Fill Mode (`m-X`)

Turns on auto filling. Auto Fill mode allows you to type text endlessly without worrying about the width of your screen. Return characters are inserted where needed to prevent lines from becoming too long. This command displays `Fill` in the mode line.

Electric Character Style Lock Mode (`m-X`)

Puts comments in the character style that is the value of **zwei:*comments-character-style-index***, which is set to the value of `(si:style-index (si:parse-character-style '(nil :italic nil)))`, 4, corresponding to NIL.ITALIC.NIL. This command displays`Electric Character-Style-lock` in the mode line.

Electric Shift Lock Mode (`m-X`)

Facilitates typing in programs that are in uppercase. Whenever you type a character that is part of a Lisp symbol, such as the name of a function, variable, or special form, Zmacs inserts it in uppercase, but when you type a character that is

part of a character string or a comment or after a slash, Zmacs inserts it normally. This command displays Electric Shift-lock in the mode line.

EMACS Mode (m-X)

Provides commands for EMACS users. It puts bit-prefix commands on ESCAPE, c-^, and c-C, and Universal argument on c-U. It also makes c-I a synonym for TAB, c-H a synonym for BACKSPACE, and c-] a synonym for ABORT. This command displays EMACS in the mode line.

Overwrite Mode (m-X)

Turns on overwrite mode. In overwrite mode, ordinary printing characters replace existing text, instead of inserting themselves next to it. It is good for editing pictures. This command displays Overwrite in the mode line.

Word Abbrev Mode (m-X)

Turns on Word Abbrev Mode. If Word Abbrev Mode is already on, this command turns it off. Word Abbrev Mode allows you to define word abbreviations that expand as you type them. This command displays Abbrev in the mode line.

Major Modes

Whenever you are editing some text, some set of modes is in effect. The buffer is always associated with one major mode that tells the editor what kind of document is being edited. A major mode has the following characteristics:

- It has its own distinct set of key bindings.
- It affects groups of related language-specific items, such as delimiter characters and indentation rules.

The major modes are listed below. You can establish the mode:

- By turning it on using the prefix m-X followed by the name of the mode. For example, to invoke Lisp Mode, type: m-X Lisp Mode.
- By setting it in the attribute list. See the section "Buffer and File Attributes in Zmacs".
- By having Zmacs do it for you when you specify a file with c-X c-F or the Edit File command. It recognizes the type component of the pathname of the file (for example, folon.lisp) and puts the buffer in the corresponding mode.

Fundamental Mode

Fundamental Mode enters Zwei's fundamental mode (the default mode).

Lisp Mode

Lisp Mode sets things up for editing Lisp code. It puts Indent-For-Lisp on TAB.

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Common-Lisp. See the section "Buffer and File Attributes in Zmacs".

Text Mode

Sets things up for editing English text. It puts Tab-To-Tab-Stop on TAB.

Zmacs FEP-Command Mode

Sets up the buffer for editing FEP command files. This mode is automatically entered when you read in a FEP .boot file. Use c-m-X FEP Command Mode to enter this mode manually. Comment delimiters are "(" and ")". TAB is Indent Relative. All tabbing is done by inserting spaces, not tabs.

Note

Zmacs supports Fortran Mode as a part of FORTRAN 77, the separately priced software product. For more information, see the User's Guide to Symbolics Fortran.

Macsyma Mode

Macsyma Mode enters a mode for editing MACSYMA code. It modifies the delimiter dispatch tables appropriately for MACSYMA syntax, makes comment delimiters /* and */. It puts Indent-Relative on TAB.

Midas Mode

Midas Mode sets things up for editing PDP-10 assembly language code.

Bolio Mode

Bolio Mode sets things up for editing Bolio source files. It is like Text Mode, but also makes c-m-*N*, c-m-:, and c-m-* insert font characters, and makes word-abbrevs for znil and zt.

Teco Mode

Teco Mode sets things up for editing TECO. It makes comment delimiters be !* and *!. It puts Indent-Nested on TAB, Forward-Teco-Conditional on m-', and Backward-Teco-Conditional on m-].

User-Defined Major Modes

In Zmacs, you can define your own major modes (see **zwei:defmajor** in the code).

File Types and Major Modes

You can control the default major mode associated with a particular file type. For example, Zmacs sets the major mode to Lisp for files with type lisp. The repository for this information is a list called **fs:*file-type-mode-alist***.

For example, suppose you wanted to associate the file type tex with text mode:

```
(push '("tex" . :text) fs:*file-type-mode-alist*)
```

The **car** of an element should be either a canonical type symbol or a string when the type is not one of the known canonical types.

In addition, suppose you have files that would require Scribe mode, if Zmacs had such a thing. You can define a correspondence between two major modes, using a global variable called **zwei:*major-mode-translations***. It is an alist of major mode names, expressed as keyword symbols.

Example:

```
(push '(:scribe . :text) zwei:*major-mode-translations*)
```

Creating New Commands with Keyboard Macros

A *keyboard macro* is a command that you define to abbreviate a sequence of other commands. If you discover that you are about to type `c-N c-D` 40 times, you can define a keyboard macro to do `c-N c-D` and call it with a repeat count of 40.

The keyboard macros described here are temporary keyboard macros; that is, macros that you use for the duration of your Zmacs session. For information on writing permanent keyboard macros that you can save and initialize in your init file, see the section "Writing and Saving Keyboard Macros".

You define a keyboard macro by telling Zmacs that you are about to write a macro and then typing the commands that are the definition. That is, as you are defining a keyboard macro, the definition is being executed for the first time. When you are finished, the keyboard macro is defined and also has been, in effect, executed once. You can then do the whole thing over again by invoking the macro.

Procedure

1. To start defining a keyboard macro, type `c-X (` (Start Kbd Macro). From then on, your commands continue to be executed, but also become part of the definition of the macro. `Macro-level: 1` appears in the mode line.

2. If you want to perform an operation on each line, do one of the following:

- Start by positioning point on the line above the first one to be processed and then begin the macro definition with a c-N
- Start on the proper line and end with a c-N.

Either way, repeating the macro operates on successive lines.

3. After defining the body of the macro, you can terminate it in several ways.
   - c-X) (End Kbd Macro) terminates the definition.
   - An argument of zero to c-X) automatically repeats the macro (upon termination of the definition) until it gets an error or reaches the end of the buffer.
   - c-X) can be given a repeat count as a numeric argument, in which case it repeats the macro that many times right after defining it, but defining the macro counts as the first repetition (since it is executed as you define it). (Subsequent invocations ignore the numeric argument contained in the macro.)

Inserting an argument of 5 before ending the macro (...c-5 c-X)) executes the macro immediately four additional times.

Starting a Keyboard Macro

c-X (Start Kbd Macro

Begins defining a keyboard macro. A numeric argument means append to the previous keyboard macro.

Ending a Keyboard Macro

c-X) End Kbd Macro

Terminates the definition of a keyboard macro.

Showing a Keyboard Macro

To see the keyboard macro, use Show Keyboard Macro (m-X), which prints the macro at the top of your screen.

Show Keyboard Macro (m-X)

Displays the specified keyboard macro. The name of the macro is read from the minibuffer; just RETURN means the last one defined, which can also be temporary.

Calling the Last Keyboard Macro

The macro thus defined can be invoked again with `c-X E` (Call Last Kbd Macro), which can be given a repeat count as a numeric argument to execute the macro many times.

`c-X E` Call Last Kbd Macro

Repeats the last keyboard macro.

Example

The example below defines a keyboard macro that goes to the beginning of a line, inserts a semicolon, and goes to the next line. It also executes the macro four times, including once as it is being defined.

```
c-X (
c-A
;
c-N
c-4 c-X )
```

For information about setting key bindings permanently, see the section "Zmacs Key Bindings".

Writing and Saving Keyboard Macros

Writing and saving keyboard macros entails:

- Defining the macro with **zwei:define-keyboard-macro**.
- Installing the macro on a keystroke with **zwei:make-macro-command**.
- Storing the macro into a comtab with **zwei:command-store**.

zwei:define-keyboard-macro takes as its arguments the name of the macro and the keystrokes specifying what you want it to do.

Optionally, you can install the macro on a keystroke with **zwei:make-macro-command**, giving the name of the macro, which returns a Lisp function.

zwei:command-store takes that Lisp function and stores it into a comtab, similar to what **zwei:set-comtab** does. **zwei:command-store**, given the key you want to install the macro on and the comtab in which to put it, stores the command in the slot of the comtab that you specify. The combination of **zwei:make-macro-**

command and **zwei:command-store** does the same thing as the Install Macro (m-X) command.

Using variations of the following forms you can save the macros on disk and, if you wish, edit them.

Example 1

Suppose you want to have a command that exchanges the first two words on a line. Put this form in your init file:

```
(ZWEI:DEFINE-KEYBOARD-MACRO EXCH-FIRST-TWO-WORDS (NIL)
  #\C-A #\M-F #\M-T)
```

The macro cannot be more than 255 keystrokes long. If your macro gets this long you should be writing in Lisp, since keyboard macros are not intended to be a programming language. If necessary, you can get around this restriction by breaking your macro into parts and having them call each other.

Suppose you want to install the EXCH-FIRST-TWO-WORDS macro on the keystroke s-Q. Put this form in your init file:

```
(ZWEI:COMMAND-STORE (ZWEI:MAKE-MACRO-COMMAND ':EXCH-FIRST-TWO-WORDS)
                    #\S-Q ZWEI:*ZMACS-COMTAB*)
```

Example 2

The following form defines a keyboard macro called replace-test, which replaces the string dog with the string river:

```
(ZWEI:DEFINE-KEYBOARD-MACRO REPLACE-TEST (NIL)
  #\C-S "dog" #\ESCAPE #\M-RUBOUT "river")
```

To save the keyboard macro replace-test on the keystroke h-S:

```
(ZWEI:COMMAND-STORE (ZWEI:MAKE-MACRO-COMMAND ':REPLACE-TEST)
                    #\H-S ZWEI:*ZMACS-COMTAB*)
```

The h-S command takes a numeric argument as a repeat count.

Defining an Interactive Keyboard Macro

Within the keyboard macro definition, you can specify steps at which you want the macro to query. To define an interactive keyboard macro, use the Kbd Macro Query command after beginning the macro definition (with Start Kbd Macro). Invoke Kbd Macro Query at each spot in the macro where you want the macro to query. Then close the definition with End Kbd Macro.

c-X Q — Kbd Macro Query

Allows user interaction on each iteration of macro, similar to Query Replace (m-X). While defining a keyboard macro, press c-X Q at each step where you want a pause to occur. Upon execution of the macro, it stops and waits at each of those steps for one of the following characters:

SPACE	Continues execution of the macro.
RUBOUT	Skips rest of keyboard macro (use nested c-X (and c-X) for grouping to control range of skip).
? or HELP	Displays HELP information.
.	Continues but does not iterate anymore.
!	Continues, iterates, but does not ask anymore.
c-R	Enters editing mode; c-m-FUNCTION R resumes the keyboard macro.

Naming a Keyboard Macro

Having defined a keyboard macro, you can name it with Name Last Kbd Macro (m-X). A prompt (Name for macro:) appears in the minibuffer.

Name Last Kbd Macro (m-X)

Assigns a name to the most recent temporary keyboard macro, making it permanent. The new name for the macro is read from the minibuffer.

Using Keyboard Macros to Sort

You can use a keyboard macro to set up a sorting mechanism and run it on any region of text.

For information about how to sort using keyboard macros, see the description of Sort Via Keyboard Macros (m-X), see the section "Sorting in Zmacs".

Installing a Macro on a Key

To bind the macro to the key of your choice, use Install Macro (m-X). You are asked to identify the macro and specify the key(s) to which you want it bound.

Install Macro (m-X)

Installs a specified user macro on a specified key. The name of the macro is read from the minibuffer, and the keystroke on which to install it is read in the echo area. If the key is currently holding a command prefix (such as c-X), it asks you for another character, so that you can redefine c-X commands. However, with a numeric argument, it assumes you want to redefine c-X itself, and does not ask for another character.

Installing a Mouse Macro

You can bind the macro to a mouse click instead of a key using Install Mouse Macro (m-X). This command works similarly to Install Macro.

Install Mouse Macro (m-X)

Installs a specified user macro on a specified mouse click. The name of the macro is read from the minibuffer, and the mouse click on which to install it is read in the echo area. When the mouse is clicked to invoke this macro, the macro is invoked from the current location of the mouse cursor.

Deinstalling a Macro

To remove the macro from that key, use Deinstall Macro (m-X). The key is rebound to the standard system usage, if any.

Deinstall Macro (m-X)

Deinstalls a keyboard macro.

Example

This example shows how to install a macro and deinstall the same macro:

```
you type:
m-X Install Macro
minibuffer:
Name of macro to install (CR for last macro defined):
you type:
macro-name or CR
minibuffer:
Key to get it:
you type:
h-T

A menu appears and asks you in which comtab to install the macro:
```

- Just this editor
- Zmacs
- Zwei

```
Click on your choice.

minibuffer:
Command #<DTP-CLOSURE 34465726> installed on Hyper-T.

you type:
m-X Deinstall Macro
minibuffer:
Key to deinstall:
you type:
h-T

The menu appears and asks you to specify in which of the three
comtabs to deinstall the macro.  Click on your choice.

minibuffer:
Command NIL installed on Hyper-T.
```

For information about saving keyboard macros permanently, see the section "Zmacs Key Bindings".

Making Tables Using Keyboard Macros

The keyboard macro facility implemented with the c-m-FUNCTION key provides more features, such as an easy way to make tables.

c-m-FUNCTION

Reads a keyboard macro command, consisting of an optional numeric argument made up of any number of digits (0-9) followed by a non-numeric character, usually a letter. Each keyboard macro command must be preceded by the c-m-FUNCTION prefix. After typing the prefix, you can press HELP for a list of available keyboard macro commands.

Keyboard Macro Commands for c-m-FUNCTION

0-9 Optional numeric argument.

C Calls a macro by name. Prompts in the minibuffer for the name of the macro.

P Begins a macro definition (same as c-X — see the section "Starting a Keyboard Macro".)

R Ends a macro definition (same as c-X — see the section "Ending a Keyboard Macro".)

M Defines a named macro. Prompts for the name of the macro to define and then enters macro definition mode.

S Stops (aborts) macro definition (also c-G).

D Defines a named macro but does not execute it while reading its characters.

SPACE Inserts pauses for user interaction in the macro (same as c-X Q — see the section "Defining an Interactive Keyboard Macro".)

A Steps though characters on successive iterations (for example, letters and numbers). Asks for starting character, amount to increase (or decrease if negative) on each iteration.

U Allows typein terminated by c-m-FUNCTION R. This allows you to stop while in the middle of defining the macro, do other things in the editor, and then go back and finish defining the macro.

T Allows typein every iteration.

The difference between c-m-FUNCTION U and c-m-FUNCTION T is that c-m-FUNCTION U allows typein while defining a macro. This typein does not get stored in the macro, and therefore does not get executed on subsequent iteration, nor when the macro is called again.

c-m-FUNCTION T allows typein on every iteration. As with c-m-FUNCTION U, the typein while defining the macro does not get stored in the macro. But on each subsequent iteration, new typein will be requested.

Example 1

The following example shows how to create a macro that constructs a table using c-m-FUNCTION A.

you type:	c-X (
Minibuffer:	Macro-level: 1 *
you type:	c-m-FUNCTION A
Minibuffer:	Initial character (type a one-character string):
you type:	a RETURN
Minibuffer:	Amount by which to increase it (type a decimal number):
you type:	1 RETURN

(Zmacs inserts the a into the buffer.)

you type:	c-2 c-6 c-X)

As you close the macro, Zmacs inserts into the buffer:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

by executing the macro 26 times, increasing the letter once each time.

Example 2

The following example shows how to create a macro that constructs a table using c-m-FUNCTION A, and this time, c-m-FUNCTION T, which allows typein during every iteration of the macro:

you type:	c-X (
Minibuffer:	Macro-level: 1 *
you type:	Item SPACE
you type:	c-m-FUNCTION A
Minibuffer:	Initial character (type a one-character string):
you type:	1
Minibuffer:	Amount by which to increase it (type a decimal number):
you type:	1
you type:	TAB
you type:	c-m-FUNCTION T
Minibuffer:	Macro-level: 2 *
you type:	Rosemary
you type:	c-m-FUNCTION R
Minibuffer:	Macro-level: 1 *
you type:	RETURN
you type:	c-5 c-X)
you type:	Sage
you type:	c-m-FUNCTION R
you type:	Thyme
you type:	c-m-FUNCTION R
you type:	Parsley
you type:	c-m-FUNCTION R
you type:	Pepper
you type:	c-m-FUNCTION R

The table looks like this:

```
Item 1  Rosemary
Item 2  Sage
Item 3  Thyme
Item 4  Parsley
Item 5  Pepper
```

Key Bindings

A *key binding* is the set of specific keystrokes that invoke a specific command.

How Key Bindings Work: The Comtab

A *command table*, or *comtab*, assigns a command to each possible keystroke. While Zmacs is running, there is always a unique *selected comtab*, in which Zmacs finds the command that corresponds to each user keystroke.

When you type a keystroke, Zmacs looks up the keystroke in the currently selected comtab, finds the appropriate command, and runs it. Usually the command's side effects all occur within the buffer: Point might be moved and text might be deleted, inserted, or rearranged. Sometimes a command has more extensive side effects. A command can alter or replace the selected comtab itself, in which case Zmacs looks up the next keystroke in the new command table.

Zmacs's *basic state* consists of the standard editor key bindings, which reside in one special command table, the *standard comtab* (*Zwei comtab*). The standard comtab interacts with the Zmacs comtab and the various mode-dependent comtabs. The typical selected comtab when in Zmacs is "unnamed" for mode-specific key bindings, which indirects to "Zmacs", which indirects to "Zwei". Although the standard comtab can be temporarily replaced, it is always reselected eventually, often after only one "nonstandard" keystroke.

A keystroke that functions as a prefix actually runs a command that replaces the standard comtab for one keystroke. This is the mechanism by which multikeystroke commands are implemented. For example, there are many two-stroke commands whose first keystroke is c-X. This keystroke runs a command that brings in its own comtab before interpreting the next stroke.

Setting the Key

If you want to put a command on the keystroke of your choice, use Set Key. This command works for any of the already defined commands.

Set Key (m-X)

Installs a specified command on a specified key. If the key is currently holding a command prefix (such as c-X), it asks you for another character so that you can redefine c-X commands. However, with a numeric argument, it assumes you want to redefine c-X itself and does not ask for another character.

It assigns key bindings in the editor that are active in all buffers, and takes two arguments: the name of a command, and a keystroke to invoke it. It reads the name of the command in the minibuffer, completing any command name in any comtab.

Install Command

If you want to put a function on the keystroke of your choice, use Install Command. It takes a function, regards it as a command, and puts it on a key.

Install Command (m-X)

Installs a specified function as a command in the comtab, on a specified key. It takes two arguments: the name of the function (the current definition, that is, top-level expression), and a keystroke to invoke it. (Zmacs treats as a definition any top-level expression having in functional position a symbol whose name begins **"def"**.) If the key is currently holding a command prefix (such as c-X), it asks you for another character so that you can redefine c-X commands. However, with a numeric argument, it assumes you want to redefine c-X itself and does not ask for another character.

How to Specify Zmacs Variable Settings

A *variable* is a name that is associated with a value, for example, a number or a string. Zmacs has editor variables that you can set for customization. (Variables can also be set automatically by major modes.)

You can distinguish the names of Zmacs variables from other Lisp variables by their names — the first letters are capitalized and the names contain spaces rather than hyphens.

Finding Out About Zmacs Variables

To examine the value of a single Zmacs variable, use Describe Variable (m-X). To print a complete list of all variables, use List Variables (m-X).

Some commands refer to variables that do not exist in the initial environment. Such commands always use a default value if the variable does not exist. In these cases you must create the variable yourself if you wish to use it to alter the behavior of the command.

Describing Zmacs Variables

Describe Variable (m-X)

Displays the documentation and current value for a single Zmacs variable. It reads the variable name from the minibuffer, using completion.

Listing Zmacs Variables

List Variables (m-X)

Lists *all* Zmacs variables and their values. With a numeric argument, this command also displays the documentation line for the variable.

Listing Variables by Matching a String

`HELP V` Variable Apropos
`c-HELP V`
`c-m-? V`

Displays the names of all possible Zmacs variables containing a specific substring. With a numeric argument, this command also displays the documentation lines for the variables.

Example

One example of such a Zmacs variable is the Fill Column variable, which specifies the width, in pixels, used in filling text.

For example, `c-1 HELP V` prompts in the minibuffer `Variable Apropos (substring):` and you type `fill col`. It does pattern matching on the variable name and thus matches Fill column, which displays: `Fill column: 576. Width in pixels used in filling text.`

Setting Variables

Settable Zmacs Variables

You can view all settable Zmacs variables with the List Variables (`m-X`) command.

The following are some examples of variables that can be set with Set Variable (`m-X`). In addition, they can be set in init files by using the internal form of their names. For example, Region Marking Mode is **zwei:*region-marking-mode*** internally.

Region Marking Mode
: Value: **:reverse-video** for setting the region to reverse video. The default is **:underline**.

Region Right Margin Mode
: Value: **t**. Causes whatever marks the region (reverse video or underlining) to extend across unfilled space to the right margin. The default is **nil**.

One Window Default

Controls which window remains selected after a One Window (c-X 1) command when you were using more than one window. Possible values:

:current
:other
:top
:bottom

This feature operates best when the current layout has no more than two windows. The value **:current** is the only one that is always well defined with more than two windows on the screen.

Check Unbalanced Parentheses When Saving

Controls whether Zmacs checks a file for unbalanced parentheses when you are saving the file. The check is on (**t**) by default. When it checks a file that you are saving and finds unbalanced parentheses, it queries you about whether to go ahead and save anyway. This applies to all major modes based on Lisp; it is ignored for text modes.

Set Variable

Set Variable (m-X)

Sets any existing Zmacs variable. This command reads the name of a variable (with completion), displays its current value and documentation, and prompts in the minibuffer for a new value. It does some checking to see that the new value has the right type.

Although either uppercase or lowercase works, you are encouraged to capitalize each word of the name for aesthetic reasons, since Zmacs stores the name as you give it.

Customizing Zmacs in Init Files

As you gain sophistication with the more advanced features, you will find the settings of parameters that most please you and put these into a command file (*init file*) that the system executes every time you log in.

Creating an Init File

Create a file named *lispm-init.lisp* (or with the correct Lisp file type suffix for your host operating system) in your home directory on your host system and put your Zmacs customizations there.

This section contains examples of forms that you can place inside a **login-forms** in your init file to customize the editor.

login-forms is a special form for wrapping around a set of forms in your init file. It evaluates the forms and arranges for them to be undone when you log out.

Setting Editor Variables

The forms described show how to set Zmacs variables (the kind that Set Variable (m-X) sets).

To set these variables, which are symbol macros, you must use the **setf** macro. For a description of symbol macros: See the section "Symbol Macros". For a description of the **setf** macro: See the macro **setf**.

Ordering Buffer Lists

```
(SETF ZWEI:*SORT-ZMACS-BUFFER-LIST* NIL)
```

This displays the list of buffers in the order the buffers were created rather than in the order they were most recently visited.

Putting Buffers Into Current Package

```
(SETF ZWEI:*DEFAULT-PACKAGE* NIL)
```

This puts buffers created with c-X B (Select Buffer) into whatever package is current; the default is to put them in the **user** package.

Setting Default Major Mode

```
(SETF ZWEI:*DEFAULT-MAJOR-MODE* :TEXT)
```

This sets the default major mode to Text Mode for buffers with no Mode attribute and no major mode deducible from the file type; the default is Fundamental Mode.

Setting Find File Not To Create New Files

```
(SETF ZWEI:*FIND-FILE-NOT-FOUND-IS-AN-ERROR* T)
```

This beeps and prints an error message when you give c-X c-F (Find File) the name of a nonexistent file. The default prints (New File) and creates an empty buffer, which when saved by c-X c-S (Save File) creates the file that was nonexistent.

Init File Form: Setting Refind File to Not Query for Newer Version of File

```
(SETF ZWEI:*REVERT-UNEDITED-BUFFERS-FOR-NEW-VERSION* :ALWAYS)
```

Controls the prompting behavior of Refind File, Refind All Files, and Revert Buffer if a newer version of the buffer file exists on disk. Its default is **:query**, which means ask you if you would prefer the newer version. It may be set to **:always**,

meaning pick up the newer version without bothering to ask, or **:never**, meaning do not pick up the newer version.

Setting Goal Column for Real Line Commands

```
(SETF ZWEI:*PERMANENT-REAL-LINE-GOAL-XPOS* 0)
```

This moves subsequent c-N and c-P (Down Real Line and Up Real Line) commands to the left margin, like doing c-0 c-X c-N (Set Goal Column to zero).

Fixing White Space For Kill/Yank Commands

```
(SETF ZWEI:*KILL-INTERVAL-SMARTS* T)
```

This tells the killing and yanking commands to optimize white space surrounding the killed or yanked text.

Setting Mode Hooks

Each major mode has a *mode hook*, a variable which, if bound, is a function that is called with no arguments when that major mode is turned on.

Electric Shift Lock in Lisp Mode

```
(SETF ZWEI:LISP-MODE-HOOK 'ZWEI:ELECTRIC-SHIFT-LOCK-IF-APPROPRIATE)
```

This tells Lisp major mode to turn on Electric Shift Lock minor mode unless the buffer has a Lowercase attribute. The effect is that by default Lisp code is written in uppercase.

Auto Fill in Text Mode

```
(SETF ZWEI:TEXT-MODE-HOOK 'ZWEI:AUTO-FILL-IF-APPROPRIATE)
```

This tells Text major mode to turn on Auto Fill minor mode unless the buffer has a Nofill attribute. The effect is that by default lines of text are automatically broken by carriage returns when they get too wide.

Key Bindings

To bind keys, you first define the comtab in which to put the binding. For example, ***standard-comtab*** and ***standard-control-x-comtab*** define features of all Zwei-based editors; ***zmacs-comtab*** and ***zmacs-control-x-comtab*** define features that are Zmacs-specific.

White Space In Lisp Code

```
ZWEI:(SET-COMTAB *STANDARD-CONTROL-X-COMTAB*
                 '(#\SP COM-CANONICALIZE-WHITESPACE))
```

This defines c-X SPACE as a command that makes the horizontal and vertical white space around point (or around mark if given a numeric argument or immediately after a yank command) conform to standard style for Lisp code.

c-m-L on the SQUARE Key

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
                 '(#\SQUARE COM-SELECT-PREVIOUS-BUFFER))
```

This defines the SQUARE key to do the same thing as c-m-L. This key binding is placed in ***zmacs-comtab**** rather than ***standard-comtab**** since buffers are a feature of Zmacs, not of all Zwei-based editors.

Edit Buffers on c-X c-B

```
ZWEI:(SET-COMTAB *ZMACS-CONTROL-X-COMTAB*
                 '(#\c-B COM-EDIT-BUFFERS))
```

This makes c-X c-B invoke Edit Buffers rather than List Buffers. This key binding is placed in ***zmacs-control-x-comtab**** rather than ***standard-control-x-comtab**** since buffers are a feature of Zmacs, not of all Zwei-based editors.

Edit Buffers on m-X

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
                 ()
                 (MAKE-COMMAND-ALIST '(COM-EDIT-BUFFERS)))
```

This makes Edit Buffers available on m-X in Zmacs (by default it is only available on c-m-X).

m-. on m-Left

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
                 '(#\m-MOUSE-L COM-EDIT-DEFINITION))
```

This makes clicking the left mouse button while holding down the META key do what m-. does. Invoking this command from the mouse is convenient when you specify the name of the definition to be edited by pointing at it rather than typing it.

Font Editor

Font Basic Concepts

In Genera, characters can be typed out in any of a number of different typefaces. Some text is printed in characters that are small or large, boldface or italic, or in different styles altogether. Each such typeface is called a *font*. A font is conceptually an array, indexed by character code, of pictures showing how each character should be drawn on the screen. The Font Editor (FED) is a program that allows you to create, modify, and extend fonts.

A font is represented internally as a Lisp object. Each font has a name. The name of a font is a symbol, usually in the **fonts** package, and the symbol is bound to the font. A typical font name is **tr8**. In the initial Lisp environment, the symbol **fonts:tr8** is bound to a font object whose printed representation is something like:

```
#<FONT TR8 234712342>
```

The initial Lisp environment includes many fonts. Usually there are more fonts stored in BFD files in file computers. New fonts can be created, saved in BFD files, and loaded into the Lisp environment; they can also simply be created inside the environment.

If you are loading a font contained in a font file in one of the font directories, the system loads that font the first time you reference it. However, if you are loading a font contained in a file somewhere else in the file system, load that font using the function **fed:read-font-from-bfd-file** *pathname*, where *pathname* is the pathname of the font file.

The **tv** package contains the window system, which includes fonts for screen display (as opposed to fonts for hardcopying).

Attributes of TV Fonts

Fonts, and characters in fonts, have several interesting attributes.

Character Height Font Attribute

One attribute of each font is its *character height*. This is a nonnegative integer used to figure out how tall to make the lines in a window. Each window has a certain *line height*. The line height is computed by examining each font in the font map, and finding the one with the largest character height. This largest character height is added to the vertical spacing (in pixels) between the text lines (*vsp*) specified for the window, and the sum is the line height of the window. The line height, therefore, is recomputed every time the font map is changed or the *vsp* is set. This ensures that any line has enough room to display the largest character of the largest font and still leave the specified vertical spacing between lines. One effect of this is that if you have a window that has two fonts, one large and one small, and you do output in only the small font, the lines are still spaced far enough apart to accommodate characters from the large font. This is because the window system cannot predict when you might, in the middle of a line, suddenly switch to the large font.

Baseline Font Attribute

Another attribute of a font is its *baseline*. The baseline is a nonnegative integer that is the number of raster lines between the top of each character and the base of the character. (The base is usually the lowest point in the character, except for letters that descend below the baseline, such as lowercase p and g.) This number is stored so that when you are using several different fonts side-by-side, they are aligned at their bases rather than at their tops or bottoms. So when you output a character at a certain cursor position, the window system first examines the baseline of the current font, then draws the character in a position adjusted vertically to make the bases of the characters all line up.

Character Width Font Attribute

The *character width* can be an attribute either of the font as a whole, or of each character separately. If there is a character width for the whole font, it is as if each character had that character width separately. The character width is the amount by which the cursor position should be moved to the right when a character is output on the window. This can be different for different characters if the font is a variable-width font, in which a W might be much wider than an i. Note that the character width does not necessarily have anything to do with the actual width of the bits of the character (although it usually does); it is merely defined to be the amount by which the cursor should be moved.

Left Kern Font Attribute

The *left kern* is an attribute of each character separately . Usually it is zero, but it can also be a positive or negative integer. When the window system draws a character at a given cursor position, and the left kern is nonzero, the character is drawn to the left of the cursor position by the amount of the left kern, instead of being drawn exactly at the cursor position. In other words, the cursor position is adjusted to the left by the amount of the left kern of a character when that character is drawn, but only temporarily; the left kern only affects where the single character is drawn and does not have any cumulative effect on the cursor position.

Fixed-width Font Attribute

A font that does not have separate character widths for each character and does not have any nonzero left kerns is called a *fixed-width* font. The characters are all the same width and so they line up in columns, as in typewritten text. Other fonts are called *variable-width* because different characters have different widths and things do not line up in columns. Fixed-width fonts are typically used for programs, where columnar indentation is used, while variable-width fonts are typically used for English text, because they tend to be easier to read and to take less space on the screen.

Blinker Width and *Blinker Height* Font Attributes

The *blinker width* and *blinker height* are two nonnegative integers that tell the window system an attractive width and height to make a rectangular blinker for characters in this font. These attributes are completely independent of all other attributes and are only used for making blinkers. Using a fixed width blinker for a variable-width font causes problems; the editor actually readjusts its blinker width as a function of what character it is on top of, making a wide blinker for wide characters and a narrow blinker for narrow characters. The easiest thing to do is to use the blinker width as the width of the blinker. This works well with a fixed-width font.

Chars-exist-table Font Attribute

The *chars-exist-table* is **nil** if all characters exist in a font, or an **sys:art-boolean** array. This table is not used by the character-drawing software; it is for informational purposes. Characters that do not exist have pictures with no bits "on" in them, just like the Space character. Most fonts implement most of the printing characters in the character set, but some are missing some characters.

Standard TV Fonts

You can use `Show Font HELP` in the Lisp Listener or the List Fonts (`m-X`) command in Zmacs to get a list of all the fonts that are currently loaded into the Lisp environment. The **fonts** package contains the names of all fonts. Here is a list of some of the useful fonts:

fonts:cptfont — This is the default font, used for almost everything.

fonts:jess14 — This is the default font in menus. It is a variable-width rounded font, slightly larger and more attractive than medfnt.

fonts:cptfonti — This is a fixed-width italic font of the same width and shape as **fonts:cptfont**, the default screen font. It is most useful for italicizing running text along with **fonts:cptfont**.

fonts:cptfontcb — This is a fixed-width bold font of the same width and shape as **fonts:cptfont**, the default screen font.

fonts:medfnt — This is a fixed-width font with characters somewhat larger than those of **cptfont**.

fonts:medfnb — This is a bold version of **medfnt**. When you use Split Screen, for example, the [Do It] and [Abort] items are in this font.

fonts:hl12i — This is a variable-width italic font. It is useful for italic items in menus; Zmail uses it for this in several menus.

fonts:tr10i	This is a very small italic font. It is the one used by the Inspector to say *"More above"* and *"More below"*.
fonts:hl10	This is a very small font used for nonselected items in Choose Variable Values windows.
fonts:hl10b	This is a bold version of **hl10**, used for selected items in Choose Variable Values windows.

Entering and Leaving Font Editor

You can enter Font Editor:

- By selecting [Font Editor] in the System menu.
- By typing the Edit Font command at any Lisp Listener.
- By typing `Select Activity Font Editor` at any Lisp Listener.
- By pressing `SELECT {`.

The first time you invoke Font Editor in a session, it takes about 15 seconds to start up; after that, entering Font Editor is very quick. When the startup is complete, you see a *Font Editor Frame*, the window configuration used by Font Editor. You are not editing any particular font: you can experiment with character drawing in this state, but it is best to select a font first.

If you know which font you wish to edit before entering Font Editor, you can save time and steps by typing the *font-name* as an argument to Edit Font:

```
Edit Font font-name
```

font-name can be a string, a BFD object, or any atomic symbol (on any package) whose print name is the name of the font you wish to edit.

You can exit Font Editor either by selecting some other activity (via the System menu, mouse, the `SELECT` key, or `FUNCTION S`), or by using [EXIT] in Font Editor's menu. Whenever you reinvoke Font Editor in the same session, you return to the editing that you were doing when you left Font Editor. Thus, only one Font Editor exists per session, and you do not lose your work by leaving it.

Should Font Editor become unusable because of an error, you can create a second one by pressing `SELECT c-{`. This creates a completely new Font Editor (although not destroying the old one).

Font Editor Basic Concepts

Font Editor, the Subsystem

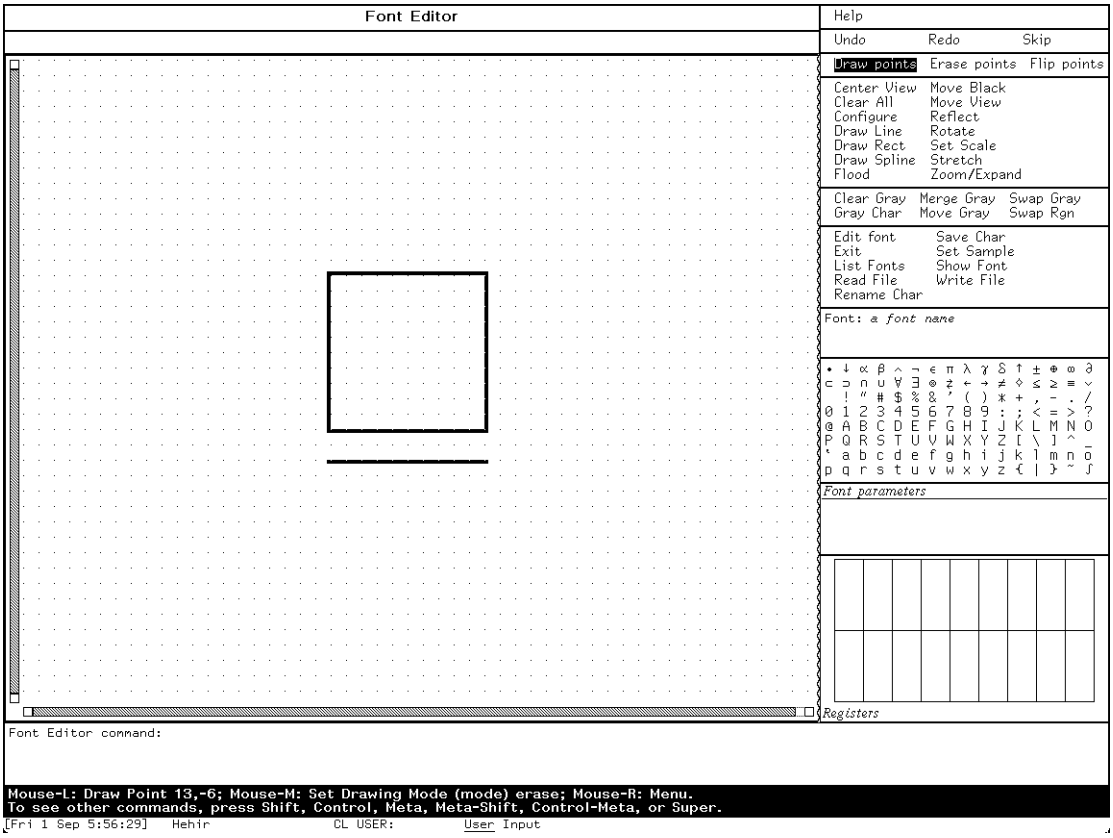


Figure 89. Initial Font Editor Display

Font Editor accepts both menu commands and character (keyboard) commands.

When you enter Font Editor, you see a complex *frame* of many *panes*. The following are descriptions of the panes in the Font Editor frame:

Drawing Pane

The largest pane is the *drawing pane*, which contains a grid of dots forming an array of squares, and a box drawn in the middle. When you edit a character, Font Editor draws the character in this pane, magnified 12 to 1. (You can choose other magnifications with the [Configure] and [Grid Size] Font Editor menu commands.) Each box, delineated by four dots, represents one *pixel* (bit-raster dot) of the character being edited.

The basic technique of editing characters is to draw lines, points, and curves on this pane, using the mouse as a graphic input device, and thus modify the bit-raster definition of the character being edited. Mouse clicks on the drawing pane draw

and clear points. For information on mouse use on the drawing pane: See the section "Drawing in Font Editor".

Character Box

The box drawn in the center of the drawing pane is called the *character box*. It shows the font baseline and character height, as well as the width and kerning of the character being edited. The box itself shows the right and left margins of the character, and the top and baseline of the font. The line under the character box shows the *character height* of the font, which is the height that the window system uses to compute line spacing for windows with the current font in their font map. It represents, in essence, the maximum height of any character in the font, although it is a font parameter, not one computed by inspecting all characters in a font.

You can alter the positioning of the character box, as well the character width it represents. See the section "Viewing and Altering a Character in the Font Editor Character Box".

Sample Pane

The topmost pane of the Font Editor frame is called the *sample pane*. It shows what the character being edited looks like in normal size. That display appears in the leftmost part of the sample pane. About an inch to the right of that, the sample pane shows a life-size *sample string* in the font being edited. (You can set this sample string with the [Set Sample] Font Editor menu command.) The sample string allows you to see what a given word or phrase, drawn in the font being edited, looks like. This allows you to see your changes to a given character in context. Note that the sample pane changes size as you select fonts of differing character height.

Prompt Pane

Between the sample pane and the drawing pane is the *prompt pane*. This is used whenever keyboard typein is required. Occasionally, messages and instructions to you (such as how to use the mouse for curve and line drawing) appear there too.

Menus

To the right of the drawing pane is a set of menus and miscellaneous panes.

Draw Mode Menu

The topmost menu is called the *draw mode menu*; it tells the default interpretation of mouse clicks on the drawing pane. One element of the draw mode menu is always highlighted, and specifies the current interpretation of the mouse on the drawing pane. Selecting (by mouse click) any item on the draw mode pane makes the selected mode be the new default, and highlights that mode. Other ways of changing the draw mode also update the highlighting in this pane. See the section "Drawing in Font Editor".

Under the draw mode menu appear three *command* menus that display a repertoire of commands that you can issue at any time by clicking on their items with the mouse. Many of the items interpret the different mouse buttons differently. See the section "Font Editor Command List". The mouse documentation line at the bottom of the screen displays the interpretation of the mouse buttons when the mouse is positioned over a potential choice.

The three command menus are grouped by related function:

Drawing Pane Menu

The topmost command menu (drawing pane menu) presents a group of commands allowing you to control how you are looking at what you see, and commands to perform automatic transformation and drawing on the character being edited.

Gray Plane Menu

The second command menu contains commands apropos the *gray plane*, which is, in effect, a second pane behind the drawing pane, whose display is shown in gray instead of black. You can use the gray plane to see two characters at once, to see one character as a model while editing another, and so on. The gray plane can be moved around and manipulated in several ways. See the section "The Font Editor Gray Plane".

Outside Font Editor Command Menu

The third command menu contains commands dealing with the world outside Font Editor: reading and writing files, getting help, leaving Font Editor, and selecting and saving characters and fonts.

Status Pane

Under the command menus is the *status pane*, which tells you what font and what character is being edited. The character is displayed in the default Lisp Machine font: this is to be considered an *identification* of the character you are editing. For example, if you display the **greek9** with [Show Font], you see that the omega character in **greek9** occupies the position that corresponds to W in the default Genera font. So the status pane identifies that character as W, but the "real" character (omega) is displayed in the sample pane. The status pane also shows you the width of the character being edited. The width is changed by manipulating the vertical edges of the character box; this action updates the status pane's display.

Character Select Menu

Under the status pane is the *Character Select* menu, which is used to select a character to edit. Simply clicking on an item in this menu (once a font has been selected) draws that character in magnification in the drawing pane, so you can begin editing it. You can also use the Character Select menu to answer any prompt for a character, such as those issued by the Rename Character and Gray Character commands. When a prompt is issued that can be answered by clicking on this menu, it says so in its text.

Font Parameters Menu

Under the Character Select menu is the *Font Parameters* menu. It displays the font-wide parameters, such as blinker height and width, and baseline and character height. This is a Choose Variable Values menu; by clicking on any of the numbers in it, the menu "opens up" and allows you to type in a new value. When you change a font parameter in this way, the change takes effect immediately. The Font Editor frame can even change shape to accommodate the new parameters. All values in this menu are displayed and accepted in decimal, regardless of the setting of **zl:base** and **zl:ibase**.

Register Pane

The final pane of the Font Editor frame is the *register pane*, which is labelled *Registers*. It is divided into as many little boxes (*registers*) as fit; the size of the boxes is computed from the parameters of the current font. Registers can be used to store characters and pieces of characters being edited, and retrieve them, without storing them into any font. See the section "Saving Characters and Pieces of Characters in Font Editor Registers".

Font Editor has an alternative *configuration*, or pane layout, that gives a wide aspect ratio (screen-wide) to the drawing pane, as opposed to the normal tall aspect ratio. The [Configure] menu item in the top command menu can be used to switch configurations. When you click Right on it, it pops up a menu of the two possible configurations, *wide-aspect* and *tall-aspect*. Tall-aspect is the default, shown in Figure 89 . Figure ! shows the wide-aspect configuration.

Many Font Editor commands produce *typeout*, text and/or drawings that are "written over" the whole Font Editor frame display. [Show Font] and [List Fonts] are typical of such commands. When a command produces typeout, the typeout remains until the next command is typed. Pressing SPACE is a command that does nothing; use it to erase typeout and do nothing more.

Selecting a Font

Font Editor edits one font at a time, and one character in that font at a time. You can make new fonts, and add new characters to fonts. Using Font Editor consists

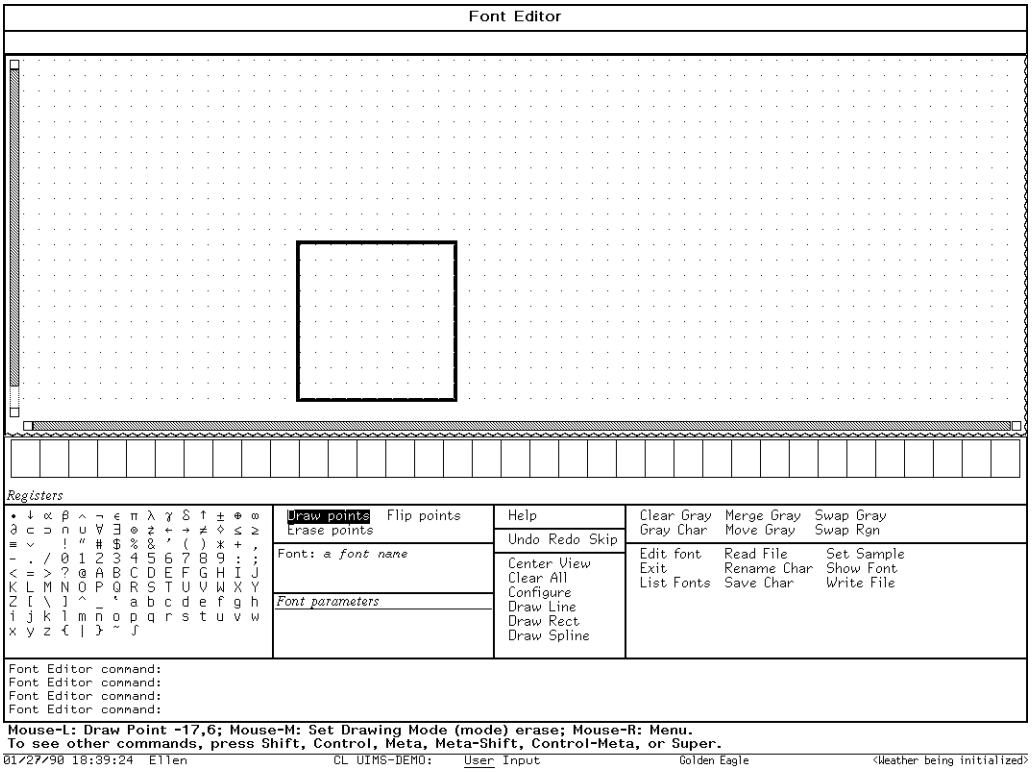


Figure 90. Wide Configuration

of selecting a font, then selecting, successively, several characters in that font, editing each one in turn, and "storing" it back into the font. When this editing is finished, the font in the Lisp environment reflects all of these changes. At that time, you usually want to write the font out to a BFD file, to save your work. See the section "Reading and Writing Font Editor Files".

Font Editor provides several ways to select a font.

- You can name the font to be edited in the command that invoked Font Editor. See the section "Entering and Leaving Font Editor".

- You can select the [Edit Font] menu item, which prompts for the font name in the prompt pane. Use [Edit Font (sh-Left)] to copy an existing font as the first step of making a new font.

- You can list all loaded fonts with the [List Fonts] menu item. The display produced by [List Fonts] is mouse sensitive: moving the mouse over the name of

any font highlights it, and clicking on it begins editing of that font. Using [List Fonts (sh-Left)] lists all fonts on the file computer as well as loaded ones. This usually takes a long time to produce. The keyboard command F can also be used to prompt for the name of a font to edit.

Creating a New Font

If you attempt to edit a font that is not known to the system, Font Editor asks you whether you wish to create that font. This is the way you create new fonts. When you create a new font, the first thing you usually want to do is alter the font parameters (in the font parameters menu) and define the Space character, from which many facilities in the system (including some in Font Editor) determine the "usual width" of characters in this font. As a matter of fact, you might want to reconfigure the Font Editor frame after setting the width of Space, to correctly recalculate the width of registers in the register pane.

Displaying Characters in the Font

When you start editing a font, you are not editing any character. The drawing pane displays a typical character box, and no points. The specifications for the character box reflect the Space character in the font. You must then select a character to edit. Font Editor then displays all of the characters in the font, using the display normally obtainable by [Show Font]. You can erase this display by pressing SPACE or by selecting a character to edit. See the section "Selecting a Character in Font Editor".

Selecting a Character

Once a font has been selected, Font Editor edits one character at a time. You modify the definition of the character by drawing and clearing points on the drawing plane. When you are done editing a character, you store it back in the font by using the [Save Char] menu item. Your changes to the character are not saved until you do this. Furthermore, none of your changes to a font being edited become permanent until you write the font out to a file.

From the Character Select Menu

The usual way to select the character being edited is by using the mouse to select a character in the Character Select menu. When you select a character, it is drawn in magnification in the drawing pane, and the status pane is updated to tell you what character you are editing.

By Creating a New Character

If you attempt to edit a character that is not in the font being edited, Font Editor creates a new character. This is the way new characters are created. The new

character is not actually saved in the font until the [Save Char] command is issued.

From the [Show Font] Display

You can also select a character by displaying all of the characters of the font being edited, via the [Show Font] menu item. The display produced by this command is mouse sensitive: when you move the mouse over the image of a character, it is highlighted, and if you click on it, editing of that character begins. This display is produced automatically when you select a font to be edited.

With the `C` Command

The keyboard `C` command can also be used to select a character. Pressing `C` prompts for a character, which can be supplied from the keyboard or the Character Select menu.

By Renaming Characters

Another way to edit a character is to *rename* the character being edited to some other character. This is one way to move characters around in a font, and make characters into other characters. Selecting the [Rename Char] menu item prompts for a character to call the character being edited. You can answer this prompt either by typing a character from the keyboard, or from the Character Select menu. This changes Font Editor's idea of what character you are editing, and the status pane and sample string (if any) are updated to reflect this fact. Renaming a character does *not* store it back in the font; you must do that by yourself, as usual, with the [Save Char] command when you are done editing it.

Drawing

The most common technique for creating and editing characters is to draw and clear points on the drawing pane using the mouse.

Drawing Characters with the Mouse

Drawing on the drawing pane is in one of three modes at any time, [Set Points], [Clear Points], or [Flip Points]. The highlighted item in the draw mode menu tells which is in effect. When you click left on a box in the drawing pane, that box is made black (set), or white (clear), or complemented (flip), according to the current draw mode. If you hold the left button down (that is, you do not release it after clicking left on a box) and move it around, you set (or clear or complement) all squares over which you pass. In this way, you can draw curves or pictures, fill in areas, clear old mistakes, and so forth. This is the most common operation in Font Editor, and is called *drawing with the mouse*.

You can change the drawing mode either by selecting another draw mode by clicking on an item in the draw mode menu, or by clicking middle on the drawing pane. Clicking middle rotates through the possible draw modes.

When you draw with the mouse, the sample pane is not updated until you release the left button. (You might want to do this every now and then while drawing with the mouse, just to observe what you have in life-size, and then press the left button again, to continue drawing.)

Often, you might want to "temporarily" change the draw mode, either because the draw mode menu is too distant, or the mouse is not in top shape, or because you really want to change the draw mode for just one or two squares. You can do this while drawing by manipulating the CONTROL and META keys on the keyboard. If you hold down CONTROL alone while drawing, the temporary draw mode becomes [Clear Points] for as long as it is held down. Similarly, META alone sets up [Set Points] mode for as long as it is held down. CONTROL and META together temporarily put the mouse in *pass-over* mode, in which it makes no change to any squares it passes over.

Flip mode is useful for final touch-ups, a click at a time, rather than drawing with the mouse button down. Since it changes any square you click on, it is most useful when you fix up single squares in the final stages of editing a character.

Viewing and Altering a Character in the Character Box

The character box is the mechanism by which you can view and alter the boundaries of a character being edited. The following is a description of its edges, and instructions for changing them.

What the Lines Mean

Font Editor displays a *character box* in the drawing pane, to indicate the "boundaries" of the character being edited. These boundaries are not absolute limits outside which the character cannot extend; rather, they are the positions that are to be considered the start and end of this character when it is drawn in use. Characters in italic fonts and foreign scripts often extend into the "territory" of the previous or next character. Such "incursion" is accomplished by a character's containing points outside its limits.

Left and right edges

> The left edge of the character box represents the cursor position at the time the character is drawn in real use. Any points to the left of this are in the "territory" of the previous character. The right edge represents the start of the next character. The distance between the left edge and the right edge is called the *character width*, and specifies the distance by which the window system increments its horizontal cursor position after drawing this character. Points to the right of the right edge of the character box are an incursion into the territory of the next character to the right.

Bottom edge — The bottom edge of the character box (*not* the line under it) represents the *baseline* of the font. The baselines of all characters drawn on a line, in any font, form a continuous line, the normal "bottom" of most characters. Points below the baseline are "descenders".

Top edge — The top edge of the character box represents the top of the character. You cannot put points above the top, but Font Editor lets you draw such points, for you might move them and/or the character box before you save the character. Font Editor warns and asks you what to do if you attempt to save a character that has points above its top edge; this is an error. The distance between the top edge and the baseline is fixed for any given font (although you can use Font Editor to change the value of that number). If you are making a new font, you should carefully consider this parameter (the font's *baseline*) before generating any characters.

Character height — The line below the bottom of the box represents the *character height* of the font, which is the distance between the top edge and this line. This distance, too, is a fixed parameter for any font, although you can use Font Editor to alter it for the whole font. You cannot put points below this line; if you do, they appear in the territory of the *next* line when drawn, and are cleared or overwritten inconsistently. The maximum of the character heights of all fonts in the font map of a window is used to compute the line spacing of a window.

Altering the Character Box

You can move the edges of the character box on the drawing pane by clicking on them (within one-half box on either side) with the right mouse button. Hold the button down and move the line to where you want it to be, and then release the button.

Moving the character box redefines the orientation of the character, as drawn, with respect to the other characters in the same font.

If you attempt to move the bottom edge, top edge, or character height line, you move them all, and thus move the whole character box vertically. You cannot move them individually because the distances between them are fixed parameters for the font. If you alter these parameters by selecting them in the Font Parameters menu, the character box is altered and redrawn appropriately.

Sometimes, you want to move the whole character box without changing its shape. The easiest way to do this is to move the data being displayed with the [Move Black] menu item. See the section "Transformations on Characters in Font Editor".

The Gray Plane

The gray plane is a "shadow" "behind" the drawing pane that allows you to look at another character in addition to the one you are editing. The character (or piece of a character) in the gray plane shows up in light gray in the drawing pane. Where bits are on in both the gray plane and the character being edited (the *black plane*), a dark gray square is shown.

Frequently, the gray plane is used to hold a character that resembles, or has pieces of, the character being edited, to serve as a guide for drawing the new character. At other times, the gray plane is used to hold a piece of a character, to be merged later into the black plane.

The second of the three command menus is a special menu for commands dealing with the gray plane. It is also possible to fetch previously created patterns into the gray plane from the register pane. See the section "Saving Characters and Pieces of Characters in Font Editor Registers".

Getting Things into Gray

The most common ways of putting drawings into the gray plane are to move the black plane into it and to fetch characters into it. The [Swap Gray] and [Gray Char] menu items do this.

With [Swap Gray]

[Swap Gray] exchanges the black and gray planes; what had been black becomes gray, and what had been gray becomes black. After you use [Swap Gray], you are editing in the black plane what had been in the gray plane, and what you had been editing in the black plane (where all editing is done) is now visible in the gray plane. You can clear the black plane with [Erase All]; [Clear Gray] (in the gray plane menu) clears the gray plane.

You can swap the gray and black plane to bring the gray plane up for editing, to move something you have edited into the gray plane, or to do both at once.

With [Gray Char]

You can bring characters directly into the gray plane. Using [Gray Char] prompts you for a character in the current font to be brought into the gray plane. You can then type the character, or select it in the Character Select menu. The keyboard command `G` does this, too. The character is placed at the character box. It does not really matter where the character is placed, though, because before merging it or using it, you can move it to any place in the gray plane by using [Move Gray]. See the section "Merging Characters with the Font Editor Gray Plane".

You can bring characters from other fonts into the gray plane by using [Gray Char (R)]. A Choose Variable Values menu is presented, offering choices not only of character and font, but of scaling as well. Click on values you wish to change; keep in mind that the [Character] item expects a single character when you use it. Scaling allows you to grow or shrink the character being fetched before

bringing it into the gray plane. The numerator and denominator of the scale fraction are displayed and interpreted as decimal numbers. When you are done choosing values for [Gray Char], use [Do It] to bring in the character.

Merging Characters with the Gray Plane

The gray plane is the mechanism for adding pieces of characters into characters being built. You do this in two steps:

1. Put a character or a piece of a character into the gray plane and position it. You use the [Move Gray] command to reposition a drawing in the gray plane. It leaves the black plane and the character box unaffected; it moves bits within the gray plane only. When you use it, you are asked in the prompt pane for two points, which you indicate by clicking left on them in the drawing pane. These points indicate where *from* and where *to* move the data in the gray plane. Font Editor temporarily grays (in a distinguishable gray) the points you select so that you can see them, and then moves all the data in the gray plane so that the first point is moved to the second. Usually, rather than clicking random points, you should click a specific point in the gray drawing and the point in the black drawing with which you wish the gray point to coincide. You might also think of these points as a point in the gray plane and a point in the black plane to which the point in the gray plane is to be made to coincide.

2. Merge it into the black plane. The [Add in Gray] command merges the gray plane *into* the black plane. Normally, you use [Add in Gray]. This turns on (makes black) each point in the black plane that is "over" a turned-on (gray) point in the gray plane, and leaves the gray plane as it was. Thus, the points that were gray now all appear in dark gray, indicating they are on in both planes. Using [Add in Gray (M)] is similar, but clears the gray plane afterwards.

You can also merge the gray plane into the black plane by other logical operations than the default Inclusive Or: using [Add in Gray (R)] pops up a menu of logical combination operators. ANDCA (turn off all black points corresponding to "on" points in the black plane, that is, punch a hole in the black plane as indicated by the gray plane) and XOR (flip all points in the black plane that are on in the gray plane) are offered, as well as the default value, IOR.

Saving Characters and Pieces of Characters in Registers

Font Editor's gray plane allows you to edit one character or piece of a character. You can also save characters and pieces in *registers*. The register pane shows the contents of registers that can hold characters and pieces of characters for reuse.

Saving a Drawing into a Register

You save a drawing (in the black plane, after editing) by clicking left on one of the empty registers (little boxes) in the register pane. Do *not* use the first (upper left-hand) one. Clicking left on an empty register (one that looks blank) saves the current black drawing in that register. Registers are mouse-sensitive, and grow a thick border when you move the mouse over them. Click on an empty register, and the drawing in the black plane appears in that register, in the register pane, and remains there. Font Editor makes every effort to show you a visible piece of that character, so that you know it is there.

Retrieving the Contents of a Register

To retrieve a register, click left on it, and the contents of the register are transferred into the black plane. If you click on a register that has a drawing in it, that drawing goes into the black plane. If it does not have a drawing in it, the black plane goes into it. Thus, clicking left on registers is usually the only dealing you have with them.

Retrieving the Black Plane While Manipulating Registers

You might click on a different register than the one you intended. Or perhaps a register is not really empty, but has a peculiar drawing in it that has a gigantic empty middle. In either of these cases, you might lose the very work in the black plane that you were trying to save. Thus, Font Editor always copies the current black plane into the upper left-hand register when fetching the contents of a register, in case you made a mistake. You can then click on the upper left-hand register to retrieve its contents.

Drawings saved in registers are saved as bits; the orientation and size of the character box are not saved.

It is possible to save the gray plane into a register, or fetch a register into the gray plane. It is also possible to store into a nonempty register from either plane. If you want to do any of these operations, click right on a register, and a menu of possible operations pops up.

Transformations

Although drawing with the mouse is the most common way to create characters and pieces of characters, Font Editor can provide a good deal of automatic drawing help, such as drawing lines and curves and performing transformations on the character being edited. As is true of drawing with the mouse, all of these operations are applicable only to the black plane. If you want to perform them on the gray plane, swap planes, perform them, and swap back.

Clearing the Drawing

The simplest operation on a drawing is getting rid of it; [Erase All] clears the entire (black) drawing. The gray drawing, if any, is left intact. You are queried to

make sure you really want to clear the entire drawing. This function is also accessible via the keyboard command E.

Rotating Drawings

Font Editor can rotate characters any number of degrees. Rotations are performed about the center of the square whose top, right, and left edges are the top, right and left edges of the character box, and thus, whose bottom must be, and is, a distance below the top of the character box equal to the character width.

Rotation	*Mouse command*
90 degrees counter clockwise	[Rotate (L)]
Help	[Rotate (M)]
Choose number of degrees	[Rotate (R)]

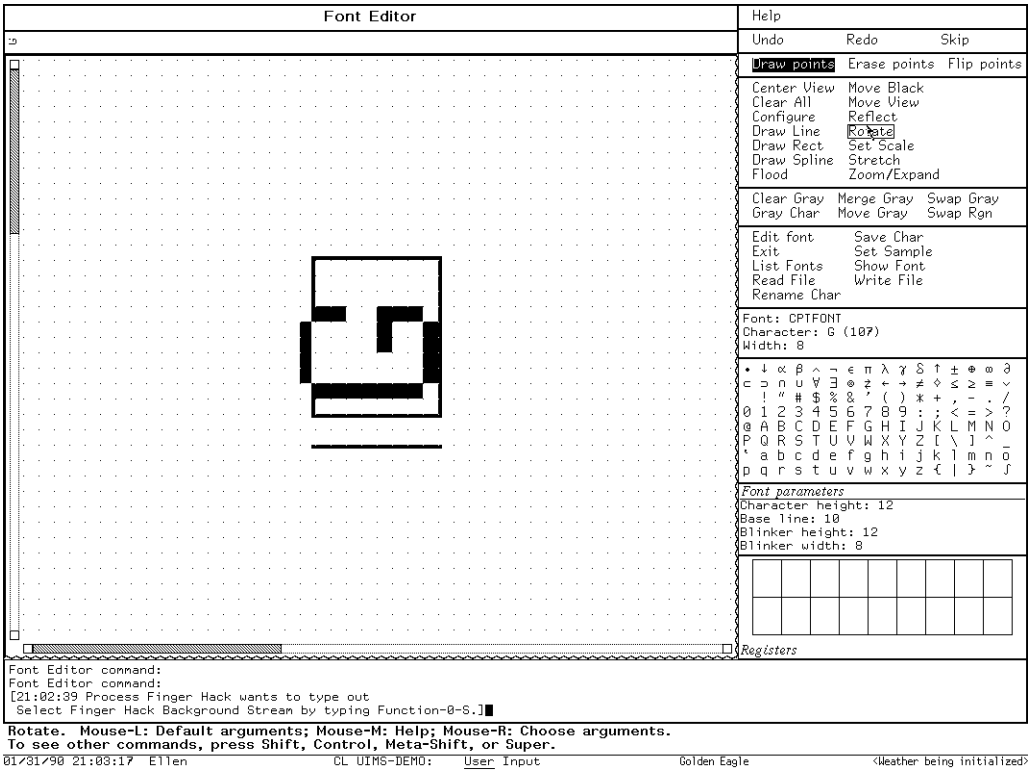


Figure 91. [Rotate (L)]

Reflecting Drawings

Font Editor can reflect drawings about any of four lines. Using [Reflect (R)] pops up a menu of the four lines about which to rotate the drawing. Those lines all pass through the "center" of the character box, the point halfway between the left and right edges and halfway between the top and the *bottom line*, *not* the baseline.

These lines are the horizontal (-), vertical (|), and 45-degree diagonal (/ and \) lines through the center point of the character box.

Reflection is subtle; it is very different than rotation. Imagine the drawing as made of sheet metal, lying on the plane. Rotation moves the character around in the plane, turning it, but never lifting it off the plane. Reflection picks it up, and puts it *back*, *face down* on the plane. The effects of diagonal reflections are subtle. The best way to understand these commands is to edit an asymmetrical but simple character (the one of choice is F) in a straightforward font (for example, HL12B), and try these various reflections upon it, as well as the rotations.

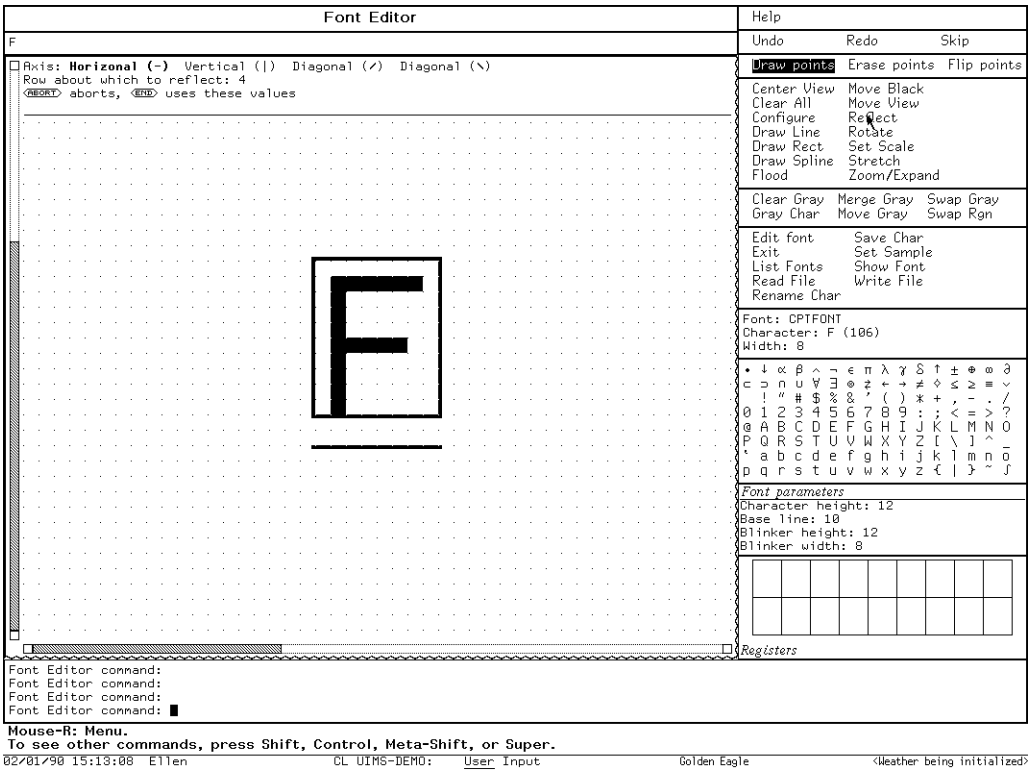


Figure 92. [Reflect-R]

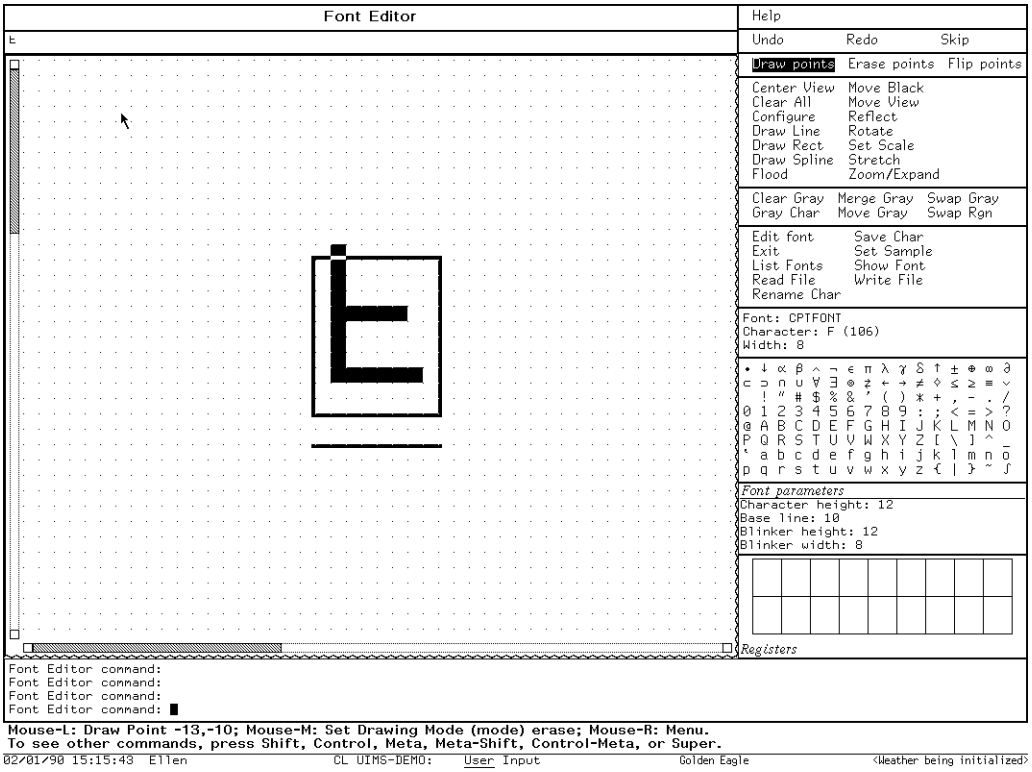


Figure 93. The Horizontal Axis (-)

Moving the Drawing

You can move the drawing around with [Move Black]. [Move Black] moves the drawing with respect to the character box, the drawing pane itself, and the gray plane. [Move Black] prompts for two points, a point in the black plane and a point to which to move it. The whole black drawing moves along with it as well.

Drawing Lines and Curves

Font Editor can draw approximate lines and curves in the drawing. Rather than drawing actual lines and curves on the drawing, Font Editor manipulates squares *along* the line or curve desired. Thus, if you ask to draw a line that is not straight up, down, or across, Font Editor approximates as well as it can.

To draw a line, use [Draw Line], and select two points between which to draw a line. As with all commands in which Font Editor prompts for points, the points are temporarily grayed when you click on them, to verify your choices. The line is

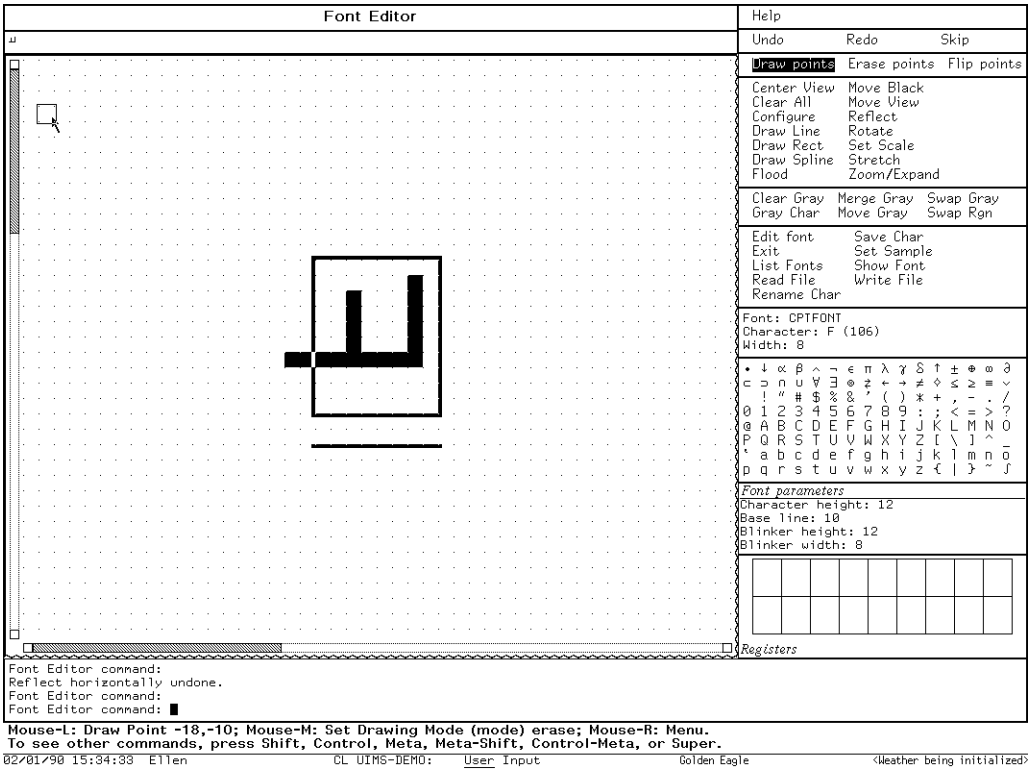


Figure 94. The 45-degree Diagonal (/)

drawn in the current draw mode, which means it clears a line if appropriate, or even flips all the points along one (which is hardly ever appropriate).

To draw a curve, use [Draw Spline]. Then click left on all the points through which the curve is to pass. When you are done, use [Draw Spline (R)]. The spline-drawing package is called to compute the points of an unconstrained cubic spline through these points, and the approximate curve is drawn in the current draw mode. See the section "Drawing Splines on Windows".

Stretching and Contracting

Font Editor can stretch or contract drawings. This is not the same as growing or shrinking them. Stretching means inserting duplicate rows or columns at a given point of the drawing, and contracting means removing rows or columns. Growing and shrinking, in general, mean scaling the whole drawing up or down. The latter is done with the options to [Gray Char]. See the section "Getting Things into the Font Editor Gray Plane".

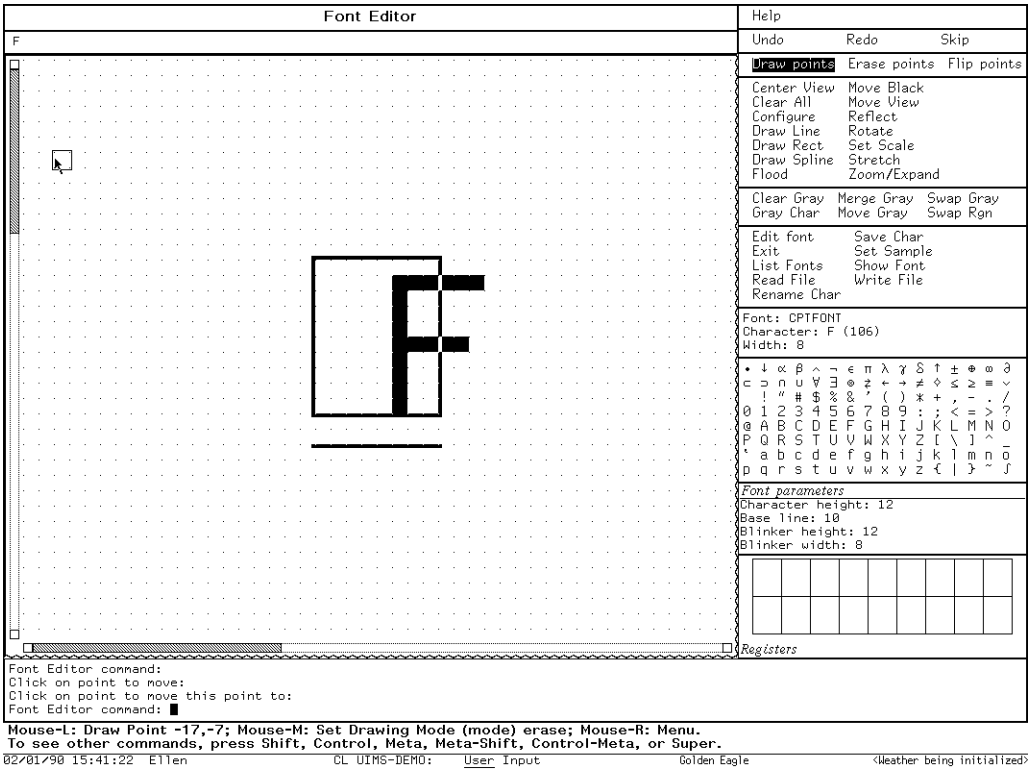


Figure 95. Moving the Drawing with [Move Black]

The relative orientation of the first and second points clicked on specifies whether you want to stretch or to contract.

Stretching a Drawing Horizontally

Stretching a drawing horizontally means making some number of copies of a column of squares to the right of that column. To stretch a character horizontally, use [Stretch], and then click left on any square in the column to be "stretched". Then click left on any square in the column to the right of that to which that column is to be stretched (that is, the last column to be a duplicate of the column being stretched). The entire drawing is stretched, with the required number of copies of the duplicated column inserted.

Contracting a Drawing Horizontally

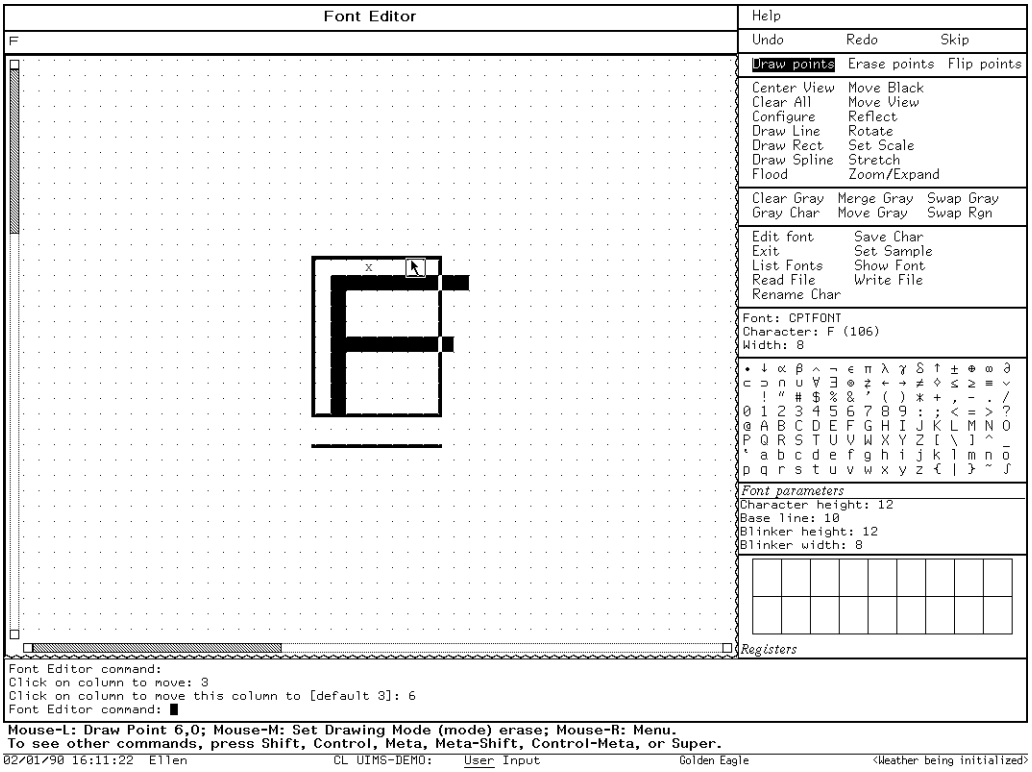


Figure 96. Stretching Horizontally

Contracting a drawing horizontally means eliminating some number of columns of squares. To shrink a character horizontally, use [Stretch]. Then click left on any square in the rightmost column not to be eliminated, at the right edge of the columns to go, and then on the leftmost column to be eliminated. You should think of this as clicking on a column to move, and where to move it to.

Stretching a Drawing Vertically

Stretching a drawing vertically means making some number of copies of a row of squares below that row. To stretch a character vertically, use [Stretch (R)] and select Stretch Row. Then click left on any square in the row to be "stretched". Then click left on any square in the row below that to which that row is to be stretched (that is, the last row to be a duplicate of the row being stretched). The entire drawing is stretched, with the required number of copies of the duplicated row inserted.

Contracting a Drawing Vertically

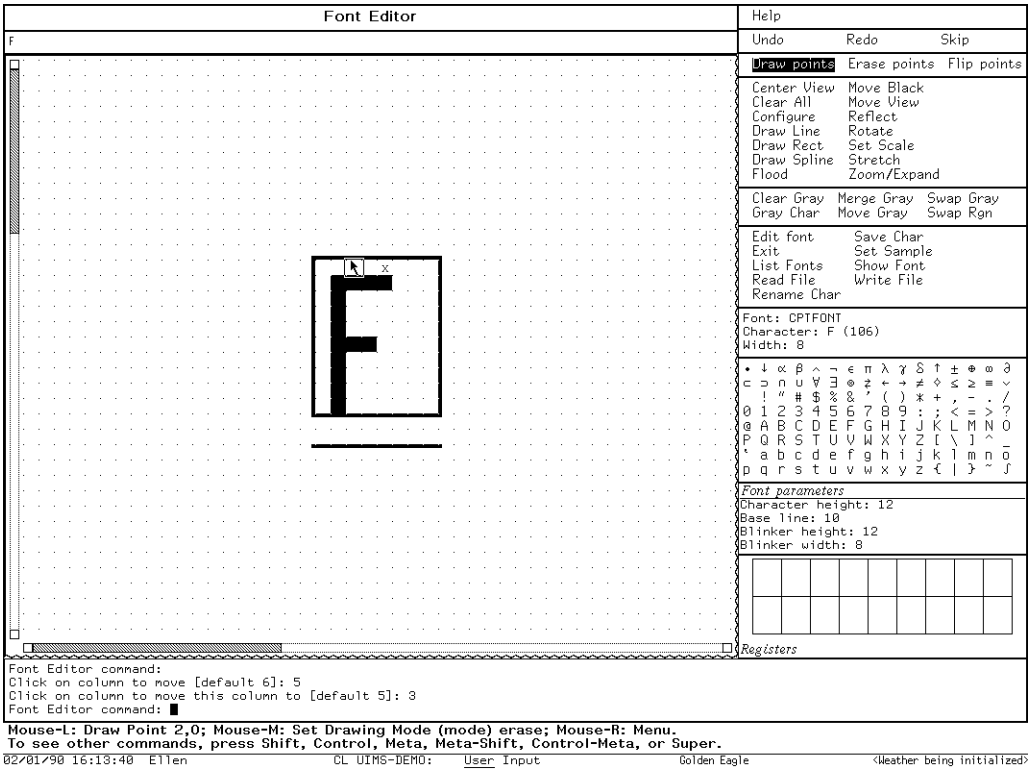


Figure 97. Contracting Horizontally

Contracting a drawing vertically means eliminating some number of rows of squares. To shrink a character vertically, use [Stretch (R)] and choose Stretch Row. Then click left on any square in the topmost row not to be eliminated, at the top edge of the rows to go, and then on the topmost row to be eliminated. You should think of this as clicking on a row to move, and where to move it to.

The Sample String

When you edit a font, it is usually convenient to maintain a *sample string*, displayed in the font, so that you can see how the character you are editing looks in the context of other characters next to which it might appear.

Font Editor allows you to set a sample string. The straightforward method of setting it is to select the [Set Sample] menu item: doing so prompts you for the string, which should be short enough to fit in the sample pane (it is clear if it does not, as you only see the end of it). End the string by pressing RETURN. The string is then displayed in the sample pane.

If the sample string contains the character being edited, occurrences of that character are updated whenever any change is made to the drawing. Thus, the occurrences of the character being edited in the sample string reflect the state of the current drawing, not the state of that character stored in the font.

Two other ways to ask Font Editor to prompt you for the sample string are clicking any button on the sample pane itself, and issuing the `U` command from the keyboard. This last is often the most convenient, because you are then going to type the string itself.

Adjusting the Display

The commands and facilities described here deal with positioning the drawing display and modifying its visible characteristics. They do not actually change the data in the drawing, but rather, the way it is viewed.

Positioning the Drawing

Both the black and gray drawings can be thought of as being drawn on an infinite plane. The character box is in the center of that plane. Although [Move Black] and [Move Gray] exist to move the drawings, and the character box can be moved by clicking on it, sometimes you might want to reposition the entire drawing, character box, black drawing, gray drawing, and all. This can also be viewed as repositioning the *view* of the drawing offered by the drawing pane. Font Editor provides several techniques for repositioning the entire drawing.

[Move View] The simplest is [Move View]. [Move View] works just like [Move Gray] and[Move Black]. When you use [Move View], it prompts you for two points, which you indicate by clicking left on squares on the drawing pane. The first point is a point on the drawing; the second is a point in the pane to which to move it. The whole drawing is moved, perhaps simultaneously vertically and horizontally, so that the first point is where the second point had been.

[Center View] Another common need is to recenter the drawing, that is, put the character box back in the middle. This is the way the drawing pane starts out when you begin editing a character. The [Center View] menu item performs this task. Use [Center View] to recenter the drawing. The keyboard `H` (for Home) command does this too.

Scrolling Another way to reposition the display is to *scroll* it up or down or left or right. In order to scroll the display vertically, a scroll bar is provided at the left of the drawing pane. When you move the cursor to the extreme left edge of the drawing pane and bounce the cursor at that edge, the cursor changes to a double-pointed arrow and the left margin of the drawing pane displays a graph of the vertical portion of the drawing

you are looking at. The status line documentation reflects the possible options at this point.

To scroll the drawing horizontally, a scroll bar is provided at the bottom of the drawing pane. When you move the cursor to the extreme bottom edge of the drawing pane and bounce the cursor at that edge, the cursor changes to a double-pointed arrow and the bottom edge of the drawing pane displays a full-grid length graph of what horizontal portion of the drawing you are looking at. The status line documentation reflects the possible options at this point.

Setting the Box Size in the Drawing Pane

You can set the size of boxes in the drawing pane. Normally, it is 12, meaning each box, corresponding to one pixel of the actual character, is represented by a box 12 pixels wide and high. To set the size of boxes, use [Grid Size]. Font Editor prompts you for the size of a box, in decimal. This size can not be bigger than 64 pixels. If you type a carriage return without typing any number, the default size of 12 pixels is reestablished.

Setting the Height and Width of the Drawing Pane

You can tell the Font Editor frame to show either a wide drawing pane, as wide as the screen, or a tall drawing pane, almost as tall as the screen. These two *configurations* of the frame are chosen from a pop-up menu that is obtained by using [Configure]. This command can also be used to have Font Editor recompute its configuration, for example, to reshape its registers after you have edited the Space character of a font.

Reading and Writing Files

Font Editor can read and write files containing fonts in any of a variety of formats. The most common format is BFD, the standard font format of Genera. If you are making fonts for use by the Genera display and window system or the LGP-3, this is the only format you should ever have to deal with.

Most of the other formats are for compatibility with other systems and earlier releases of Genera software. Notable among these formats is PXL format, which is a standard font format with the *TEX* system on UNIX. BFD format is the default for all file reading and writing operations.

Reading Files

Use [Read File] and type in the file name to read in a font file. The file type defaults from the (canonical) type of the pathname presented as the default. For example, if you type `fix9.bfd`, or just `fix9`, you read a BFD file, whereas if you type

fix9.bin, you read a BIN file. Similarly, fix9.ibin reads an IBIN file. Font Editor complains if you supply a file type that is not a valid font file type for the machine you are using. Pressing R is equivalent to using [Read File].

When you read in a font via [Read File], it is actually loaded. It becomes part of the Lisp environment, and appears in listings of loaded fonts produced by [List Fonts] as well as by the Show Font command and by Zmacs. After Font Editor loads the file and looks for the font you specified, you are editing that font.

It is sometimes necessary to read in font files of exotic types, whose file types (as expressed in the name of the file) are not indicative of the format of the font. For instance, you might have renamed a BFD or other file to myfont.temp, and now you want to read it in. Since Font Editor cannot determine the font format from this file type, you must specify the font format explicitly. This is done by using [Read File (R)]: Font Editor offers a menu specifying file types. Click on the file type involved: Font Editor then prompts for a pathname and reads the file. Font Editor interprets the file, however, according to the format specified by the menu, not by the file type.

Writing Files

Font Editor can also write out font files. Files are written from the description of a font residing in the Lisp environment, not from any temporary Font Editor image of the font. Since Font Editor maintains no temporary image of the font, but actually stores edited characters back in the font when you use [Save Char], this is not a problem unless you forget to save your characters.

Use [Write File] to write the font file out. The file type defaults from the (canonical) type of the pathname presented as the default. For example, if you type newfnt.bfd, you write a BFD file, whereas if you type newfnt.bin, you write a BIN file. Font Editor complains if you supply a file type that is not a valid font file type for the machine you are using. Using [Write File] writes out a BFD file by default from a font description in the Symbolics Machine's virtual memory. The default directory is the system screen fonts directory; the default file name is *font*.bfd, where *font* is the current font being edited. Pressing W is equivalent to using [Write File].

It is sometimes necessary to write out font files of exotic types, whose file types (as expressed in the name of the file) are not indicative of the format of the font. For instance, you might already have a sfnt.bfd, and want to write your file to sfnt.temp. Since Font Editor cannot determine the font format from this file type, you must specify the font format explicitly. This is done by using [Write File (R)]: Font Editor offers a menu specifying file types. Click on the file type involved: Font Editor then prompts for a pathname and writes the file. Font Editor writes the file, however, according to the format specified by the menu, not by the file type.

Command List

The following is a listing of all Font Editor commands. The first part of this listing describes the commands available via the command menus and the keyboard. When a keyboard character exists duplicating a menu command, it is given in addition after the command name. The second part of this section describes the effect of clicking on various panes and mouse-sensitive areas of the Font Editor frame.

Many of the keyboard commands take *numeric arguments* to specify some number or character. Numeric arguments are entered by typing a decimal number before the command character. The numeric argument is echoed in the prompt window as you enter it.

Menu and Keyboard Commands

Configuration and Drawing Transformation

[Configure] — Pop up a menu of frame configurations. Two configurations are offered, giving a tall and wide aspect ratio to the drawing pane.

[Grid Size] @ — Set the size of boxes in the draw pane. If a numeric argument is given, it is used as the size. @ sets grid size to the default if given no numeric argument, but [Grid Size] prompts.

[Center View] H — Reposition the display in the drawing pane so that the character box is centered in it.

[Move View] — Reposition the display in the drawing pane by prompting for two mouse-specified points: which point to move and to which point to move it.

[Draw Line] — Draw a line in squares in the drawing pane, in the current drawing mode. Prompt for two endpoints, to be specified with the mouse.

[Draw Spline] — Draw a cubic spline in squares in the drawing pane, in the current drawing mode. Prompt for curve points, to be specified by using [Draw Spline]. Using [Draw Spline (R)] ends the curve.

[Erase All] E — Clear all points (black points) in the current drawing.

[Stretch] K — Stretch or contract a character, horizontally or vertically. [Stretch] is horizontally, [Stretch (R)] is vertically. Font Editor prompts for two points, specifying a row or column to move and to where to move it. From the keyboard, K means horizontal, c-K means vertical. See the section "Stretching and Contracting Drawings in Font Editor".

[Rotate] ⊕ — Rotate the drawing in the black plane. [Rotate] is 90 degrees to the left, [Rotate (R)] 90 degrees to the right, and [Rotate (M)] 180 degrees.

[Reflect] ⇔ Reflect the drawing in the black plane about a coordinate axis or diagonal line through the center of the character box. A menu pops up, asking which.

[Move Black] Move the drawing in the black plane. You are prompted for the target and destination points, which you specify by clicking left on the drawing pane.

Gray Plane Menu Items

[Gray Char] G, also M
Place a character into the gray plane. The keyboard commands accept numeric arguments to specify which character. If none is given, or if you use [Gray Char], you are prompted for a character, which you can supply from the keyboard or the Character Select menu. If you use [Gray Char (R)], you are offered a Choose Variable Values choice window to select the character, font, and scaling. For the keyboard commands, CONTROL causes Font Editor to prompt for a font name, and META causes it to prompt for scale factors.

[Clear Gray] Clear the entire gray plane.

[Swap Gray] Exchange the drawings in the gray and black planes.

[Move Gray] Move the drawing in the gray plane. You are prompted for two points, to be specified via the mouse, a point to move and a point to which to move it.

[Add in Gray] Combine the drawing in the gray plane into the black plane. Using [Add in Gray] inclusive-or's the gray drawing into the black drawing. Using [Add in Gray (M)] inclusive-or's the gray drawing into the black drawing, and clears the gray drawing. [Add in Gray (R)] pops up a menu of other combination modes.

Outside World Interface Menu Items

[Edit Font] F Pick a font to edit. You are prompted for the font name. Use [Edit Font (M)] to copy an existing font as the first step of making a new font.

[List Fonts] [List Fonts] lists all of the loaded fonts. [List Fonts (R)] lists all of the loaded fonts and fonts on the file computer. The display is mouse-sensitive; clicking left on any item begins editing that font.

[Save Char] S Store the character being edited back into the font in the Lisp environment. It is stored as the character that the status pane indicates it to be.

[Rename Char] c-C Rename the current character; make it seem as though you are now editing a different character, but retain the drawing. You are prompted for the character, which you can supply from either the keyboard or the Character Select menu. The keyboard command accepts a numeric argument to specify the character.

[Show Font] D Display all characters in the font being edited. The display is mouse-sensitive, and clicking left on a character begins editing that character.

[Set Sample] V Prompt for the sample string to be displayed in the font being edited in the sample pane, and set it.

[Read File] R Read in a file of font definitions. Prompts for a pathname. [Read File] computes the font file type from the file type of the pathname given. The default is always BFD. [Read File (R)] pops up a menu that offers the file types: BFD, KST, BIN, AC, AL, PXL, or Any. The file specified by the pathname given will be interpreted according to that format, regardless of file type.

[Write File] W Writes a file of font definitions. Prompts for a pathname. [Write File] computes the font file type from the file type of the pathname given. The default is always BFD. [Write File (R)] pops up a menu that offers the file types: BFD, KST, BIN, AC, AL, PXL, or Any. The file specified by the pathname given will be written in that format, regardless of file type.

[EXIT] Q Bury the Font Editor, and return to whatever you were doing when you last invoked Font Editor.

[HELP] HELP or ? Display a long message giving documentation of Font Editor.

Evaluating Forms from Font Editor

Font Editor uses the ESCAPE key to evaluate a Lisp form.

Keyboard-only Commands

The following commands are accessible only from the keyboard. They are mainly concerned with the nonmouse cursor, or general interaction with the subsystem.

\ Turn the nonmouse cursor on, and move it one position up the screen. A numeric argument tells to move it other than one position. c-\ and m-\ mean 2 and 4 positions, respectively, and c-m-\ means 8.

/ Same as \, but moves the nonmouse cursor down.

[Same as \, but moves the nonmouse cursor left.

] Same as \, but moves the nonmouse cursor right.

.	When the nonmouse cursor is on, complement the black square under it.
REFRESH	Redraw the drawing pane. Useful in case of perceived problems.
c-REFRESH	Clear the screen and refresh all panes in the Font Editor frame.
ABORT	Abort any command while it is prompting, waiting for either mouse or keyboard input.
C	Begin editing a character: prompt for the character, and begin editing it. Normally, you simply select a character from the Character Select menu or the [Show Font] display. C accepts a character specification as a numeric argument.

Mouse Sensitivities

This section describes the result of clicking the mouse on various portions of the FED frame other than the command menus.

The Drawing Pane

Click left	Draw a black square in the current draw mode, which is shown by the Draw Mode menu. It continues drawing as the mouse is moved as long as the left button is held down. Pressing CONTROL while drawing means temporarily go into [Clear Points] mode (META means [Set Points] mode); neither changes any points.
Click middle	Change the draw mode, cycling through the three possible draw modes.
Click right	Only meaningful when the mouse is over a boundary of the character box. "Pick it up" and begin moving it as the mouse is moved, as the right button is held down.

The drawing pane has a scroll bar at its left edge.

The Draw Mode Menu

Clicking any button on one of the draw modes selects that draw mode until it is next changed by clicking on this menu, or clicking middle on the drawing pane.

The Sample Pane

Clicking any button on the sample pane prompts for a new sample string.

The Character Select Pane

Clicking left on any character in the character select pane begins editing it. The character select pane can also be used to answer any command that is prompting for a character.

The Font Parameters Menu

Clicking left on any item in the Font Parameters menu opens it for editing. You are expected to type a new decimal number. As soon as you press RETURN, the altered parameter is stored in the font in the Lisp environment.

The Register Pane

Click left	On an empty register, store the current black plane drawing in that register. On a nonempty register, retrieve the drawing in it into the black plane, and store the current black plane drawing into the upper-leftmost register.
Click right	Pop up a menu allowing the register you clicked on to be loaded from either plane (regardless of whether or not it is empty) or retrieved to either plane.

The List Fonts and Show Font Displays

These displays are mouse-sensitive. Clicking left on a font in the [List Fonts] display begins editing it; clicking left on a character in the [Show Font] display begins editing that character.

Zmail

Overview of Zmail

Zmail is a display-oriented mail system for Genera. Using Zmail, you can send and receive mail, archive your mail in disk files, and operate on groups of messages selected according to very flexible criteria. Note that Zmail is not a facility for exchanging immediate, interactive messages with another user; that facility is called Converse. See the section "Converse".

Zmail provides the Zmacs editing commands for composing and editing messages. See the section "Zmacs".

Issuing Zmail Commands

There are three types of Zmail commands:

1. Menu commands, issued by clicking the mouse on a menu.

2. Keyboard commands, issued by pressing one or more characters on the keyboard.

3. Extended commands, issued by pressing m-X and then typing the command in the mini buffer.

   While you are typing in the minibuffer, you can use several special characters:

   - RUBOUT
     Deletes last character typed.

   - ABORT
     Aborts the m-X command. The minibuffer disappears and Zmail is ready for a new command.

   - COMPLETE
     Completes as much of the command as possible. (For example, typing "mov COMPLETE" would complete to "Move", because there are several commands beginning with "move". If you then type " to d COMPLETE", this completes to "Move to Default Previous Point" because no other commands begin with "Move to d".)

   - RETURN
     Performs completion (like COMPLETE) and then executes the command (if it is able to do completion).

   - c-?
     Shows the possible completions for the partial command being entered.

   - c-/
     Shows the command names containing the string typed.

   - HELP
     Gives information on the special characters and show possible completions.

   c-m-Y (Repeat Last Mini Buffer Command) and c-m-sh-Y (Repeat Last Matching Mini Buffer Command) work in Zmail just as they do in Zmacs. See the section "Repeat Last MiniBuffer Command (m-X) Zmail Command". See the section "Repeat Last Matching MiniBuffer Command (m-X) Zmail Command".

   There are other special characters and commands available in the minibuffer. See the section "Zmacs".

   m-X is not the only command that uses the minibuffer; most commands that prompt for keyboard input (for example, for entry of filenames) use the minibuffer. Such commands often specify a default for the data they prompt for; if so, you can get the default by just pressing RETURN to the minibuffer.

You can also press c-m-Y to yank the default into the minibuffer for editing. Commands or subcommands that use the minibuffer can always be aborted by pressing ABORT when the minibuffer appears.

One point to keep in mind is that many of the commands have options and defaults that you can customize for yourself; see the section "Setting and Saving Zmail Options".

Format Conventions for Zmail Commands

Zmail command descriptions are formatted as follows:

Command (*How*) — This represents the description of the command *Command*. *How* tells how to invoke the command.

Here are the different kinds of Zmail commands:

Menu — The command is an item to be clicked on in the top-level command menu. See the section "Top-Level Interface to Zmail".

Editor Menu — (Mail-mode commands only.) The command is an item to be clicked on in the editor menu that you get by clicking right in mail mode. See the section "Mail Mode in Zmail".

Kbd — The command is a character or key to press at the keyboard, such as c-m-SPACE or ABORT.

m-X — The command is an extended command. Type m-X (or just X) followed by the command, exactly as written. See below for an example description that includes more information on m-X.

Summary Window — The command is a click to be made on the summary window, such as click Left on Summary Line.

For example, the description of a Zmail menu item looks like this:

[Next] Zmail Menu Item

[Next] — This would describe the left click on [Next].

[Next (M)] — This would describe the middle click on [Next]. Frequently the behavior of the middle click can be set in your Zmail Profile, see the section "Zmail Profile Options".

[Next (R)] — And this would describe the right click on [Next].

The descriptions of command typed from the keyboard look like this:

n J (Kbd) — This is what the description for the J command looks like. The "*n*" means that J can take an optional numeric argument. To give the argument, type an integer (positive, negative, or zero) before pressing J. (The number is echoed in the mode-line window. If you mistyped the number you were entering, press c-G (before typing J) and start over.)

n Move to Default Previous Point (m-X)

This is the description for a m-X command that takes a numeric argument. To give an argument, type an integer, then X or m-X, then the words Move to Default Previous Point, then press RETURN. After you press the X, the mode line changes to a Zwei minibuffer into which you type the m-X command. (If you have typed an argument, 53, for example, a note appears saying "Arg = 53".)

Online Help for Zmail

Some online documentation is available. This comes in five forms:

- Explanations displayed automatically: Often, useful information about what Zmail is doing is automatically displayed on the screen. This information is usually displayed in the mode line.

- Mouse documentation line: Tells what clicking the mouse buttons would do with the mouse in its current position. You can read short documentation for many commands by watching the mouse documentation line as you move the mouse around the screen.

- Describe Command (m-X): Prompts for the name of a m-X command and displays its help documentation.

- Apropos (m-X): Prompts for a character string and displays a list of the m-X commands containing that string in their names.

- HELP key: While at top level, documentation on any top-level command is available by pressing HELP and then typing a character or key, or clicking on a command from the command menu. For example, to get information on the N command, press HELP N. (See Figure 98.)

  For information on the Next command, press HELP and then click (any button) on [Next]. (See Figure 99.)

  To learn about the Move to Default Previous Point (m-X) command, press HELP X and then type Move to Default Previous Point and press RETURN. (See Figure 100, which shows the screen just before RETURN was pressed.)

  Some of the documentation states that some command normally does such-and-such, but "is controlled by *...-...-...*". What this means is that the exact action performed by the command is an option that you can set in your profile. See the section "Customizing Zmail".

If you press HELP *, you get a list of Zmail commands and short explanations. The command names you are given are the M-X names; the list also tells if the command is available from the keyboard or command menu.

While in mail or edit mode, you can get documentation on the keyboard versions of mail or edit mode commands by pressing HELP C for Keyboard commands, or HELP D for M-X commands, followed by the command itself.

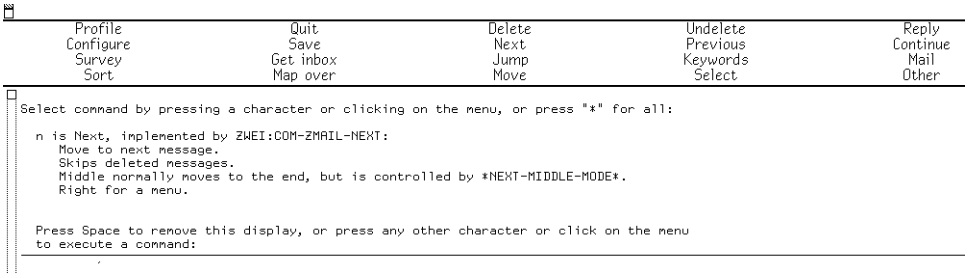


Figure 98. Help for a Keyboard Command

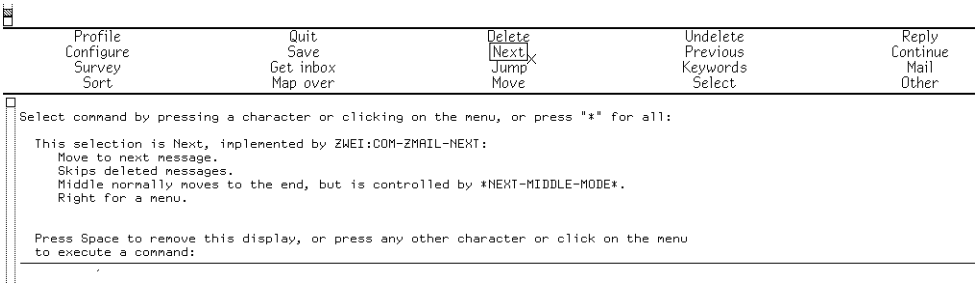


Figure 99. Help for a Menu Command

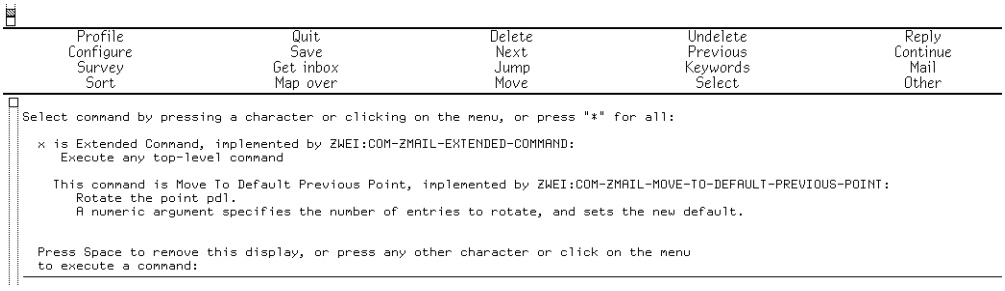


Figure 100. Help for a M-X Command

Zmail Architecture

Zmail runs in two processes. The foreground process is the main process; the background process performs file operations. Only one main Zmail window exists; by contrast, many editor windows or Lisp Listener windows can coexist. Input into text buffers is processed by Zwei, the text-handling subsystem used by the editor. Zmail submits mail to and receives mail from a *mailer* program running on a mail server. See the section "Symbolics Store-and-Forward Mailer".

Conceptual Architecture

The objects Zmail manipulates are of the following types:

messages — Composed of a *text field* and a number of *header fields*. The text field is the body of the message, while the headers supply routing information such as sender, recipient(s), date, and so forth. Messages have a variety of *properties* (for example, Answered, meaning that you have replied to the message) and can have any number of *keywords*, which are simply user-defined tags. The message you are examining at any particular time is called the *current message*.

mail buffers — Named groups of messages. Each mail buffer is associated with a disk file from which it was read, or to which it will be saved, or both. It bears the same name as its associated file. Each mail buffer stores various attributes specifying its format and other properties. You can set or alter these by using [File Options] in Profile Mode. See the section "Zmail Profile Options".

mail collections — Named groups of messages drawn from one or more mail buffers. (A single message exists in exactly one buffer and any number of collections.) The name of the collection indicates how it was created. Mail collections allow you to group related messages from one or more mail buffers. Collections cannot be saved on disk. However, you can copy the collection to a buffer and save the buffer. See the section "Saving, Expunging, Killing, and Renaming Zmail Messages".

mail sequences — Buffers and collections. The last mail sequence selected is called the *current mail sequence*.

mail files — Mail buffers saved on disk. When mail buffers are invoked, their associated files are automatically read in if they are not present already; you never actually manipulate the file itself with Zmail. Mail buffers and mail files are analogous to editor buffers and text files.

inbox files (or *inboxes*)

The files in which new mail appears. When you ask for your new mail, the contents of your inbox are appended to your *default mail file* and displayed. (Thus, unless you delete or move them, messages are saved in your default mail file. Zmail does not use a single file as both default mail file and inbox, as some mail systems do.)

Figure ! shows the relationships among objects of the above types.

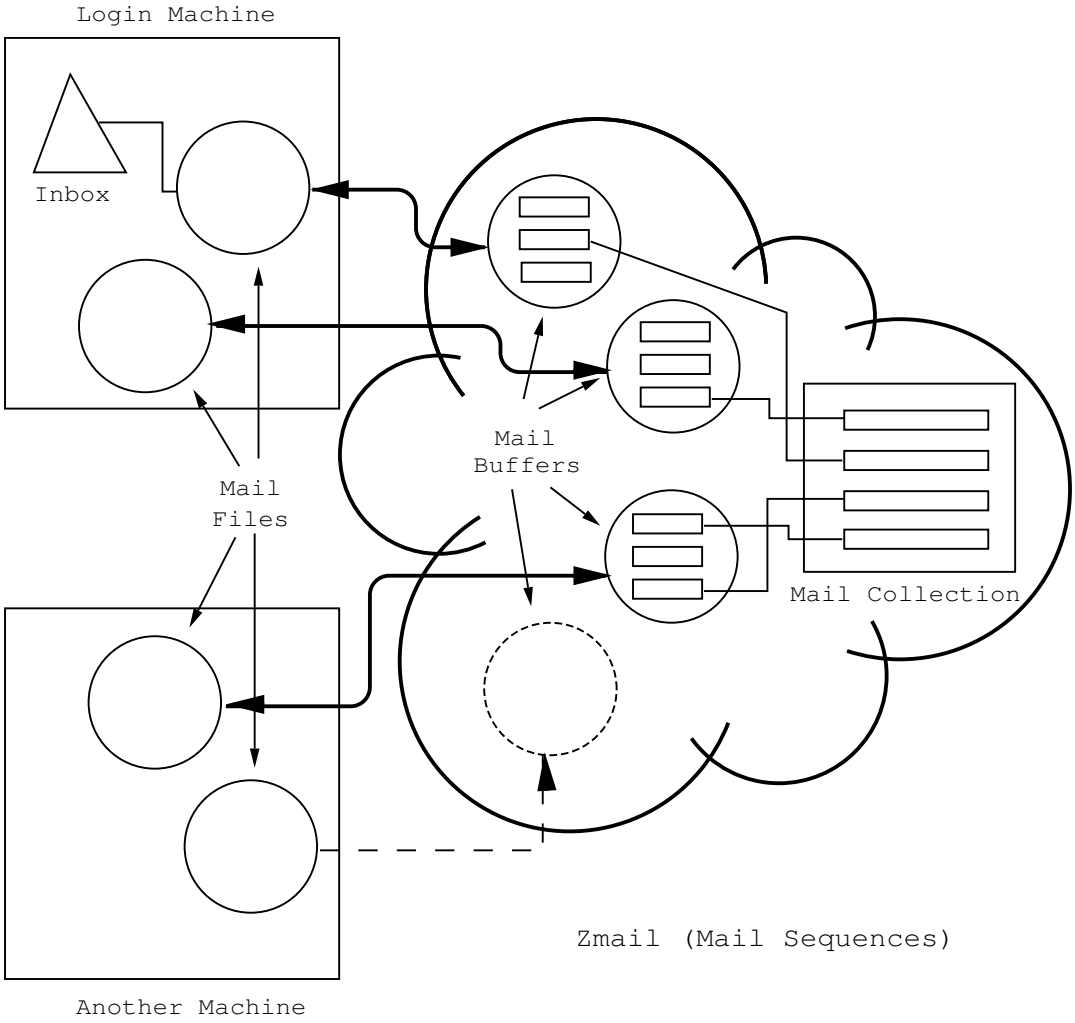


Figure 101. Messages, buffers, files, and collections.

Zmail Inboxes

The relationship of Zmail inboxes to other Zmail objects is a dynamic one, defined by the operation of the [Get inbox] command, which works like this:

1. Zmail starts reading your default mail file into a mail buffer (if it has not been read in already), and that buffer is selected as the current buffer.

2. Zmail checks to see if your inbox exists. If not, you have no new mail and Zmail displays a message to that effect. If your inbox does exist, it is renamed according to the limitations of the file system, and in such a way that it is evident that it is Zmail's temporary file, and its contents are read in by the background process and appended to the mail buffer.

3. While you read your mail, the background process saves the new version of the buffer onto the disk and deletes the renamed inbox. When this is done can be controlled in your profile. See the variable **zwei:*inhibit-background-saves***.

When Zmail checks to see if your inbox exists, it checks first for the existence of a renamed inbox. Thus the renaming ensures that no mail is lost due to a system crash. The next use of [Get inbox] after a crash results in the old renamed inbox being included first, and after that is processed, the inbox containing newer mail is renamed and read.

Top-Level Interface to Zmail

Zmail accepts input from the keyboard and the mouse. Output is displayed on, and mouse input is accepted from, the display shown in Figure !. This display has four regions:

Summary window

The *Summary Window* displays a line for each message in the current sequence, with an arrow indicating the current message (see Figure !).

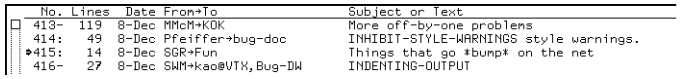


Figure 103. Zmail Summary Window

The information provided in the summary line is:

`No.` The *message number*. Whenever Zmail displays a list of messages, it numbers them for easy reference. The numbers refer only to the position of the message in the list, so when you list subsets of the mail file, the messages show up with different numbers. And when you delete or rearrange messages, the numbers change accordingly.

status letter — The *status letter* is the letter or symbol following the message number. Possible status letters are:

- `-` The message has not yet been displayed.
- `:` The message has been or is being displayed.
- `A` The message has been answered.
- `D` The message has been deleted.

(The above list is in reverse order of precedence; that is, a deleted message is marked `D` whether or not it has been answered.)

`Lines` — The message length in lines.

`Date` — The date the message was sent.

`From→To` — The sender (`From` field) and as much of the recipients (`To` field) of the message as will fit, summarized on either side of the →. A missing name before or after the arrow means the message was from or to you. For example, →`PJF,,MJH` represents a message from you to PJF, yourself, and MJH. Only the To: recipients are listed, not the Cc: or Bcc: recipients. See the section "Sending Your Mail". See the section "Commands for Sending Mail".

Keywords — The keywords attached to the message are enclosed in braces.

`Subject or Text` — The `Subject:` field of the message, or in the absence of a `Subject:` field, the first non-blank line of text in the message.

Command Menu

The *Command Menu* provides a mouse-sensitive menu of the most useful top-level commands (see Figure !). In Zmail documentation, when we refer to "[Get inbox]", for example, we mean the Get inbox command in this menu. Some of these commands (for example, [Delete]) apply only to the current message.

Figure 104. Zmail Command Menu

Message window

The *Message Window* displays the current message (see Figure !). The message window is an editor buffer.

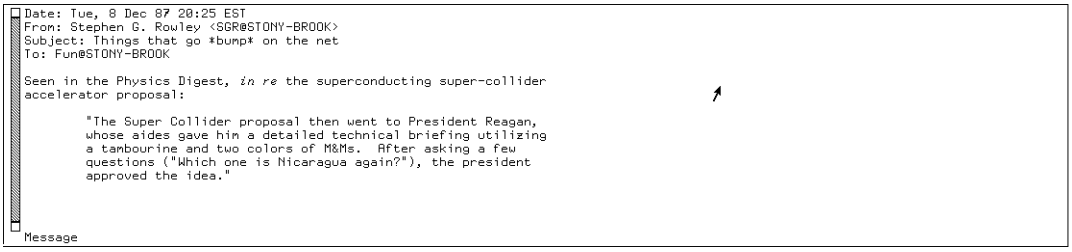


Figure 105. Zmail Message Window

Initially, there is no current message; instead, there is a short note explaining how to read and send mail. When you read your mail, the first new message becomes the current message; if there is no new mail, the first old message is the current message. As you move around the mail file to inspect other messages, they are selected as the current message and displayed.

Zmail Minibuffer

The minibuffer contains the mode line. It is also where some short notifications get displayed.

Zmail mode line

The *Mode Line* gives status information about Zmail and about the current message, including its properties and keywords.

The various information in the mode line is:

Program status — The mode the program is in. Possibilities are:

`Zmail` — Zmail is at top level.

`Zmail Mail` — Zmail is in mail mode, in which mail is sent. Following the word `Mail` is the mode in which the message to be sent is being edited, for example `(text)`. The editor mode is followed by either `Message`, `Headers`, or `Mail`, indicating which window the cursor is in. (For a description of these windows, see the explanation of the `c-X 0`, `c-X 1`, and `c-X 2` commands. See the section "Configuring and Selecting Zmail Windows". See the section "Sending Your Mail". See the section "Replying to Mail".)

`Zmail Profile` — Zmail is in Profile mode, in which you can customize Zmail. See the section "Customizing Zmail". Following the word `Profile` is the name of your Zmail init file, in which the customizations are stored.

`Zmail Marking` — Zmail is in Marking mode, executing the Mark Survey command.

`Zmail Editing Message`
: Zmail is in Editing Message mode, in which you can edit your copy of a previously received message.

Current mail file
: The name of the current mail file, or "`No current mail file`" if there is none.

Current message number/total number of messages

Message properties
: Properties describing the current message, in parentheses. Possible properties are:

`unseen`	Message is now being displayed for the first time
`deleted`	Message has been marked for deletion
`recent`	Message was new mail in the current session
`last`	Message is the last in the file
`filed`	Message has been copied to another file
`answered`	Message has been answered
`forwarded`	Message has been forwarded
`redistributed`	Message has been redistributed
`badheader`	Message has a bad header

Keywords
: Any keywords that have been saved on this message, in braces.

--More ...--
: Indicates that there is more of the message off the screen.

--More Below--
: There is more text following this screen. Use `SCROLL` to see it.

--More Above--
: There is more text before this screen. Use `m-SCROLL` to see it.

--More Above and Below--
: You are in the middle of the message.

Second mode line

The second mode line gives useful information on what the program is doing at various times. For example, when Zmail detects new mail in your inbox, a message telling you that you have new mail appears. Other messages that appear in the second mode line tell you what file the program is reading or writing, what error just occurred (Zmail flashes the screen also), or what certain keys do (for example, `END` and `ABORT`). It is a good idea to check the mode lines if you are unsure where you are in the program or how to get elsewhere.

The Summary and Message Windows can both be scrolled using the *scroll bar*. See the section "Scrolling". The Summary Window can also be scrolled forward by pressing `c-m-V` and backward by pressing `c-m-sh-V`. The Message Window can be scrolled forward with the `SCROLL` key, or by pressing `SPACE`, and backward by `m-SCROLL` or `m-V`.

Basic Zmail

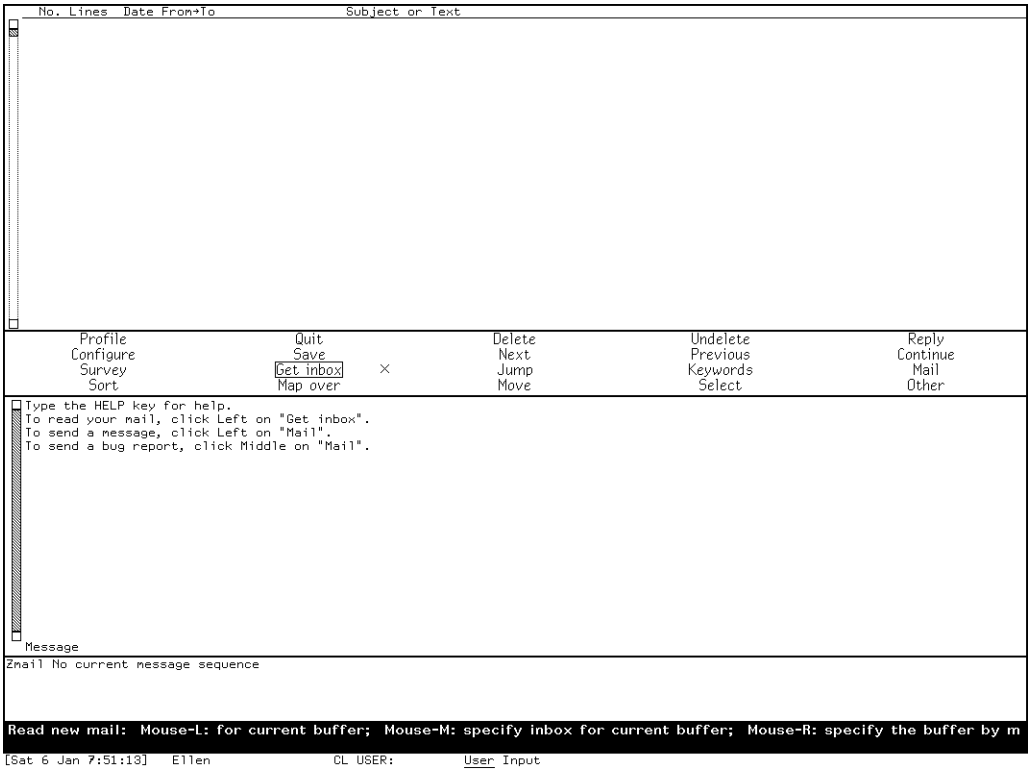


Figure 102. Main Zmail window.

Entering Zmail

Zmail can be started in several ways:

- By pressing `SELECT M` (the most common way)
- By giving the command Select Activity Mail
- By typing `(zl:zmail)` to a Lisp Listener
- By clicking on [Mail] in the System menu

When started via `(zmail)`, Zmail begins reading in your primary mail file and inbox (if any), see the section "Commands for Reading Mail".

When invoked using the `SELECT` key, the command processor, or System menu, Zmail displays an explanatory message and allows commands that do not require a mail file. Click on [Get inbox] in the Zmail menu or press `G` to read in your mail file; click on [Mail] or press `M` to send mail.

You can start up Zmail from your init file by using the function **zwei:preload-zmail**.

Exiting Zmail

The usual way to exit Zmail is to save your mail file by clicking on [Save] and then selecting another program using the SELECT key or the System menu. Clicking right on [Save] pops up a menu listing the loaded mail files, see the section "[Save] Zmail Menu Item". Pressing S does the same thing as clicking left on [Save], except that Zmail asks you to confirm the operation in case you pressed S by mistake. You can turn off this query in your Profile, see the variable **zwei:*ask-before-executing-dangerous-zmail-commands***.

Figure 106. [Save (R)]

Another way to exit is to click on [Quit], or press Q. This saves your mail files just as [Save] does, and then returns you to the window from which you selected Zmail. Clicking Right on [Quit] pops up a menu. See the section "[Quit] Zmail Menu Item".

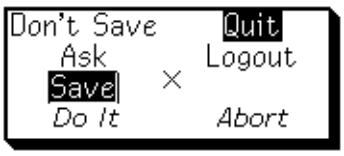


Figure 107. [Quit (R)]

Commands for Sending Mail

This section describes how to send and reply to mail. Included are brief descriptions of the Mail and Reply commands, which are used for sending various types of mail. For complete descriptions of these commands:

See the section "Zmail Mail Commands".
See the section "Zmail Reply Command".

For a complete description of the commands available in *mail mode*, in which you actually write your mail: See the section "Mail Mode in Zmail".

To send a message, click on [Mail], which is displayed in the command menu.

[Mail] or M (Kbd) Starts up a window for composing a mail message.

[Mail (M)] Starts up a window for composing a bug report. You can control the behavior of clicking Middle in your profile. See the variable **zwei:*mail-middle-mode***.

[Mail (R)] Calls up a menu of mail sending operations.

To reply to the current message, click on [Reply].

[Reply] or R (Kbd) Starts up a window to reply to the current message. You can customize the window configuration. See the variable **zwei:*reply-window-mode***.

[Reply (M)] Starts up a window to reply to the current message with the message being replied to included. You can control the behavior of click middle in your profile. See the variable **zwei:*middle-reply-mode***.

[Reply (R)] Calls up a menu of reply options.

Either [Mail] or [Reply] place you in *mail mode*. This is an editor buffer and most Zmacs editing commands are available. See the section "Mail Mode in Zmail". See the section "Zmail Reply Command".

Commands for Reading Mail

There are two types of commands for reading mail:

- Commands that read your inbox file and either load the contents into Zmail or display the contents on the screen.
- Commands that you use as you read your mail.

This section describes both types of mail reading commands, first the [Get Inbox] command and the other commands that read your inbox and then the commands that you might use as you read your mail.

Commands to Read Your Inbox

The commands to read your inbox are:

[Get Inbox] Reads in the inbox associated with the current sequence. See the section "[Get Inbox] Zmail Menu Item".

Check for New Mail (M-X)
Checks the inbox for the current sequence for new mail.

Show Mail (m-X) Displays the contents of the inbox for the current sequence. See the section "Show Mail (m-X) Zmail Command".

In reading your mail, there are two files involved: your *mail file*, which contains messages you have already seen, and your *inbox*, which contains messages you have not seen yet. If you have never run Zmail before, Get Inbox offers to create a mail file for you. Zmail renames your inbox file to indicate that the messages have been read into Zmail, and starts loading them. When you save your mail file, any renamed inboxes are deleted. If your machine should crash after you have loaded your new mail but before you have saved it, the renamed inbox is still there and the messages are read in again when you do Get Inbox after you log in again.

The loading is done in a background process, so the display of the first new message can happen rapidly. The background process continues to read in the mail file and inbox (and write out the updated mail file, if necessary). You can read and reply to your new mail while this is going on, but operations over the entire collection of new messages or your whole mail file do not work until the entire loading process is complete. If you try to use a command that operates over the entire collection of messages, for example, Select Conversation by References, the background process forces the processing of the inbox and mail file into the foreground where it completes (while you wait) and then the command is executed.

New messages have the (unseen) and (recent) properties and the "-" status letter.

(unseen) A message is unseen until it is displayed for the first time. unseen messages are marked by a "-". When a message has been displayed, it is marked by a ":".

(recent) Applies to any message that was read in as new mail since the last expunge, whether it has been seen or not.

Commands to Read Your Mail

Commands to Use While Reading Mail

[Delete] or D (Kbd) Deletes the current message and move to the next.

[Next] or N (Kbd) Moves to the next message.

[Previous] or P (Kbd) Returns to the previous message.

For more details on moving from message to message or selecting groups of messages: See the section "Zmail Message Movement Commands". See the section "Zmail Mail Collections".

[Reply] or R (Kbd) Replies to the current message. See the section "Replying to Zmail Messages".

Click right on summary line
: Pops up a menu of operations on a message. See the section "Zmail Message Summary Line".

Undigestify (m-X)
: Separates a digest into its messages. See the section "Undigestify (m-X) Zmail Command".

For more suggestions about ways to handle your mail: See the section "Managing Your Mail".

Zmail Message Summary Line

You can perform a large number of mail handling operations by clicking on the summary line of a message.

Mouse-L
: Selects the message as current.

Mouse-M
: By default, deletes or undeletes the message (toggles the delete flag). You can select the action for the middle click in your profile. See the variable **zwei:*summary-mouse-middle-mode***.

Mouse-R
: Pops up a menu of commands to operate on the message. Where appropriate, clicking left, middle, or right on the item in the menu has the same effect as clicking on the Zmail menu item of the same name. The choices are:

  Keywords
  : Adds keywords to this message. See the section "[Keywords] Zmail Menu Item".

  Delete or Undelete
  : Deletes or undeletes this message.

  Reply
  : Starts a reply to this message. See the section "[Reply] Zmail Menu Item".

  Move
  : Moves this message. See the section "[Move] Zmail Menu Item".

  Concatenate
  : Prompts for the number of a message to concatenate with this message.

  Filter
  : Lists the predefined filters that characterize this message and then starts a collection based on the one you select.

  Forward
  : Forwards this message.

  Redistribute
  : Redistributes this message.

  Survey Conversation
  : Displays the summary lines of all the other messages in this conversation.

  Select Conversation
  : Starts a collection of all the messages in this conversation.

Mail Commands

The Mail command and its variants are used for most mail sending operations: sending normal mail, sending bug reports, forwarding and redistributing mail you have received, and sending local mail. The most common mail operation not handled by the Mail command is replying to a message you have received; this is done using the Reply command. See the section "Zmail Reply Command".

Summary of Mail Commands

[Mail] Starts a new mail message. (M) and (R) offer options. See the section "[Mail] Zmail Menu Item".

M (Kbd) Starts a new mail message. See the section "M (Kbd) Zmail Command".

Bug (m-X) Starts a bug report. This inserts information about your machine's software versions for bug-tracking purposes. See the section "Bug (m-X) Zmail Command".

Forward (m-X) Forwards the current message. See the section "Forward (m-X) Zmail Command".

F (Kbd) Forwards the current message. See the section "F (Kbd) Zmail Command".

Redistribute Message (m-X)
Sends this message to some new recipients. See the section "Redistribute Message (m-X) Zmail Command".

Redirect Message (m-X)
Sends the current message to some new recipients, removing some others. See the section "Redirect Message (m-X) Zmail Command".

These commands all put you into mail mode.

Mail Mode in Zmail

When you compose a message, either a new message or a reply to a message you have received, you are in *mail mode*. The configuration of your screen varies depending on whether you are sending a new message or operating on a message you have received. If you are starting a message you are placed in *one-window* mode, that is, just a header window and a single message window. If you are replying to a message the configuration depends on how you have set up your profile. The default is *two-window* mode, that is, one window containing the message to which you are replying and the other for your reply. You can *yank* the message being replied to into your reply (See the section "c-X c-Y (Kbd) Yank Replied Messages Zmail Command".) or you can set an option in your profile to automatically do the yank when you reply. See the variable **zwei:*reply-window-mode***.

This section lists the commands available in mail mode. In addition, since the windows in mail mode are editor windows, most Zwei commands are available.

Many commands are available via the editor menu (click right on any of the editor windows in mail mode). These commands are marked "(Editor Menu)".

Except as noted, all these commands can be used regardless of which window (Headers, Mail, or Message) is selected.

This section also includes a description of the Continue command, a top-level command whose use is closely related to mail mode.

Altering Header Fields

The commands in this section provide a convenient way to add or alter various header fields. Click right while composing a mail message to get a menu of these items. Of course, since the headers window is just an editor window, the usual editor commands can be used instead of the commands listed here. Note also that the word `Subject:` can be typed in as `S:`, a convenience if you choose not to use the Add Subject Field (`m-X`) command. See the section "Zmail Header Formats".

Add `To` Field
Add `Cc` Field
Add `Fcc` Field
Add `From` Field
Add `In-reply-to` Field
Add `Subject` Field
Add `File-References` Field
Change Subject Pronouns

Configuring and Selecting Zmail Windows

(Note: The Add xxx Field commands are also window selection commands, in that they select the headers window. See the section "Altering Zmail Header Fields". In addition, you can select a window by clicking left on it.)

The explicit window configuration and selection commands are:

Add More Text (Editor Menu)
: Selects the mail window.

`c-X O` (Kbd)
: Selects another exposed window. See the section "`c-X O` (Kbd) Zmail Command".

`END` (Kbd)
: Adds more text or sends the message. See the section "`END` (Kbd) Zmail Command".

`c-X 0` (Kbd)
: Selects zero-window mode. See the section "`c-X 0` (Kbd) Zmail Command".

`c-X 1` (Kbd)
: Selects one-window mode. See the section "`c-X 1` (Kbd) Zmail Command".

c-X 2 (Kbd) Selects two-window mode. See the section "c-X 2 (Kbd) Zmail Command".

Saving and Restoring Message Drafts

If you are composing a long, complicated message, you might wish to save intermediate drafts of it. Or perhaps you want to be able to interrupt your work and come back later. The draft message and draft file facilities provide a convenient way to do this.

Zmail Draft File Facility

Using the commands listed below, draft files allow you to save messages you are composing into disk files. Draft files are written out to disk immediately, making them useful as protection against a crash. (Note: only one message can be stored per disk file; reusing the file name writes a higher-numbered version.)

[Save Draft File] (Editor Menu)

c-X c-S (Kbd) Saves the message being composed in a disk file. The first time it is used, it prompts for entry of a filename; subsequently, it uses the same filename.

[Write Draft File] (Editor Menu)

c-X c-W (Kbd) Saves the message being composed in a disk file. Prompts for entry of a filename.

[Restore Draft File] (Editor Menu)

c-X c-R (Kbd) Restores a previously saved draft. The current contents of the Headers and Mail windows are lost.

[Continue (R)] Pops up a menu of messages you have composed. Click right on [All Drafts].

[Restore Draft File]
Prompts for a filename of a saved draft and enters mail mode with the Headers and Message windows restored from the file.

Zmail Draft Message Facility

The draft message facility allows you to save message drafts in mail files using the commands listed below. Since mail files are not written out until explicitly requested, draft messages are unsuitable for protection against crashes. See the section "Exiting Zmail". But since they sit visibly at or near the end of your mail file, draft messages are good when you wish to interrupt your work and return later. A draft message is harder to forget than a draft file would be.

[Save Draft As Message] (Editor Menu)

Save Draft As Message (m-X)

c-X c-m-S (Kbd) Saves the text of the message being composed as a message. To specify a specific buffer, specify a numeric argument of 2. If the message has already been saved, Zmail does not resave it unless you specify a numeric argument of 4 (or c-U). The arguments actually are actually dealt with bit-wise, so an argument of 6 has the combined effect of an argument of 2 and an argument of 4.

Save Draft As Message (m-X) prefers the current sequence over the default sequence if the current sequence happens to be a buffer.

[Continue (R)] Pops up a menu of messages you have composed. Click right on [All Drafts].

[Restore Draft Message]
Enters mail mode with the Headers and Mail windows restored from the current message, if it is a draft message. If it is not, flashes the screen and ignores the Continue command.

[Restore Draft Message (R)]
Waits for you to click on a draft message in the summary window or type a message number in the mini-buffer, then enters mail mode with the Headers and Mail windows restored from that message. (If the selected message is not a draft message, Zmail flashes the screen and ignores the Continue command.)

Reply (Menu) *n* R (Kbd)
If the current message is a draft message, any form of the Reply command enters mail mode with the Headers and Mail window restored from the draft message. (If the current message is not a draft message, the Reply commands operate as described. See the section "Zmail Reply Command".)

Click Right on Summary Line

Note: Once mail mode has been reentered, it is just as if you had used the Mail command and retyped the message. All the mail mode commands operate as expected; in particular, END from the Mail window mails the message and ABORT returns you to Zmail top level without sending the message.

Leaving Mail Mode in Zmail

You can leave mail mode in two ways: by sending the message, or by aborting.

If you send the message, Zmail normally responds "Message sent" and returns you to top level. If there is a problem, Zmail tells you about it and remains in mail

mode to allow you to fix things up. Typical problems are omitting the To: field, trying to send mail to a nonexistent user, or mistyping a user name.

After you have sent the message, you can use Show Draft Dispositions (m-X) to find out which mail server sent your message, to whom, at what time.See the section "Show Draft Dispositions (m-X) Zmail Command".

After a message has been sent, you can edit and resend it, perhaps to different recipients, by using the Continue command. See the section "Continuing Completed or Aborted Zmail Messages".

If you abort, Zmail says Aborting, use the "Continue" command to continue. and returns to top level. You can continue using the Continue command.

END (Kbd) — Sends the message or add more text. If pressed while in the Mail window (or the Message window in zero-window mode), sends the message. Otherwise, selects the Mail window to allow you to add more text.

c-END (Kbd) — Always sends the message, whether pressed in the headers window or the mail window.

The behavior of the END key can be controlled in your Profile.

See the variable **zwei:*draft-editor-end-key-treatment***.

[Send Message] (Editor Menu) c-ESCAPE (Kbd)
Sends the message.

ABORT (Kbd) [Abort Send] (Editor Menu) c-] (Kbd)
Aborts mail mode.

Continuing Completed or Aborted Zmail Messages

The commands in this section allow you to reenter mail mode to continue editing messages already written. Already written messages are of four types:

- Messages that were sent successfully.
- Messages that were aborted. See the section "Leaving Mail Mode in Zmail".
- Draft messages saved in draft files.
- Draft messages saved as messages.

This section describes how to continue sent and aborted messages. All the messages you have composed during your current Zmail session, whether you sent them or not, are saved by Zmail and you can return to them and edit them and transmit or retransmit them. To save messages you are composing from one Zmail session to another, you must save them as drafts. Draft messages are continued in other ways. See the section "Saving and Restoring Zmail Message Drafts".

There are three ways to return to the last message you edited:

- clicking left on [Continue]
- pressing the RESUME key
- pressing C

To return to other messages, you use [Continue]: Clicking middle on [Continue] selects the last unsent (aborted) message you edited. Clicking right on [Continue] pops up a menu of Unsent Drafts, Sent Drafts, and All Drafts. When you click on one of these, a list of the header lines of that type of message pops up, allowing you to select individual messages. See the section "[Continue] Zmail Menu Item".

Note: Once mail mode has been reentered, it is just as if you had used the Mail or Reply command and retyped the message. All the mail mode commands operate as expected; in particular, END from the Mail window mails the message and ABORT aborts the send.

Commands for Including Files and Prepared Text in Messages

n c-X c-Y (Kbd) Yanks the message(s) being replied to into the buffer. (Used most often when replying to the current message.) If in two-window mode, go into one-window mode. Indent the yanked message unless an argument *n* is given. The arguments to c-X c-Y control the indentation and the pruning of headers, as follows:

Argument	*Options*
none	Indentation, pruning per the *Prune headers of yanked messages* profile option.
1	No indentation, pruning per *Prune headers of yanked messages*.
2	Indentation, pruning per reverse of *Prune headers of yanked messages*.
3	No indentation, pruning per reverse of *Prune headers of yanked messages*.

c-X Y (Kbd)

[Prune Yanked Headers] (Editor Menu)

Deletes the less essential headers of a message that was yanked in via c-X c-Y. Leaves only the Date: and From: headers; these are sufficient to identify the message. The profile option *Prune headers of yanked messages* controls the automatic pruning of message headers yanked into a reply. See the variable **zwei:*prune-headers-after-yanking***. The default is to not prune headers.

Insert File (M-X) Prompts for a pathname and inserts the contents of the file in the mail buffer.

Zmail Reply Command

The Reply command and its variants are used for replying to mail you have received. The Reply command is like the Mail command, except that it displays the original message, and it sets up the headers automatically, based on the headers of the original message.

The various forms of Reply differ in two ways: who the reply is sent to (this is called the *reply mode*), and what display format is used (the *reply window mode*). The reply mode affects the contents of the To: and CC: fields written by the Reply command. The default is to reply to everybody in the headers field, with headers like this:

To: *old From*
To: *old To*
Cc: *old Cc*

You can control reply mode in your Profile. See the variable **zwei:*reply-mode***.

The possible reply window modes are:

Two-windows Displays the original message and the reply being composed. (It uses three windows: Message, Headers, and Text. It is called Two-windows because the little Headers window does not count.)

One-window Displays only the reply. (Uses two windows, Headers and Text.)

Yank Displays only the reply, but first yanks the text of the original message, indented, into the text of the reply.

You can specify the reply window mode you prefer in your profile. See the variable **zwei:*reply-window-mode***.

The reply commands are described below. The descriptions are simply an indication of the reply mode and reply window mode used by each command.

[Reply] Two-windows/All.

[Reply (M)] Two-windows/Sender.

[Reply (R)] Pops up a menu of reply modes and reply window modes. (See Figure !.)

n R (Kbd) *n*=1, Two-windows/Sender. *n* omitted, Two-windows/All.

Click Right on Summary Line (Summary Window)
Pop up a menu, one entry of which is Reply. Left, Middle, and Right clicks on this entry have the same effect as corresponding clicks on the word [Reply] in the command menu. In addi-

tion, the message replied to is selected as current, if it is not already.

A few notes concerning Reply:

- It is possible to add an In-reply-to field to the reply to identify the original message. See the section "Zmail Header Formats". See the section "Altering Zmail Header Fields".

- Replying to a message gives it the `(answered)` property and the `A` status letter.

- Replying to a draft message simply continues it. See the section "Saving and Restoring Zmail Message Drafts". Replying to a COMSAT or XMAILR (mail server) message retries the failed message, rather than replying to anything. If the problem is a nonexistent address at another host, you are prompted to supply a corrected address.

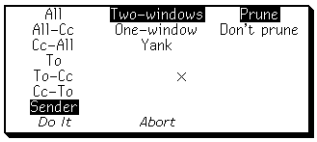


Figure 108. [Reply (R)]

Message Movement and Deletion

This section describes how to move around in your mail sequence, how to locate specific messages, and how to delete and undelete messages.

Message Movement Commands

Simple Message Movement Commands

The simple commands to move around in a mail sequence, selecting messages either by position (next, previous, or by number) or by other characteristics, are:

[Next] or `N` (Kbd) — Selects the next undeleted message. See the section "[Next] Zmail Menu Item".

n `c-N` (Kbd) — Selects the *n*th next message in the sequence, whether or not it is marked for deletion. If *n* is omitted, selects the next message, whether or not it is deleted.

[Previous] or P (Kbd)
: Move to previous undeleted message. See the section "[Previous] Zmail Menu Item".

n c-P (Kbd)
: Selects the *n*th previous message in the sequence, whether or not it is marked for deletion. If *n* is omitted, selects the previous message, whether or not it is deleted.

Click Left on Summary Line
: Selects the message whose summary line was clicked on. See the section "Zmail Message Summary Line".

n J (Kbd)
: Jumps to message number *n*, even if it is marked for deletion. If *n* is omitted, jumps to the first message in the sequence. Z J jumps to the last message in the sequence.

[Jump]
: Jumps to an arbitrary message using filters or the message stack. See the section "[Jump] Zmail Menu Item".

Find String (m-X) or c-F (Kbd)
: Prompts for a string and selects the next message containing that string. See the section "Find String (m-X) Zmail Command".

Occur (m-X)
: Prompts for a string and displays lines of text containing that string. You can select the message containing the string by clicking on the line of text. See the section "Occur (m-X) Zmail Command".

Moving Among Zmail Messages Using the Message Stack

Sometimes when you have been jumping around the mail file a lot, perhaps using commands like J or Click Left on Summary Line, you find that you would like to go back to the last message you read. If the mail file is large and you have moved great distances, you might no longer remember where you were. Rather than force you to make scribbled notes, Zmail keeps an eight deep stack of messages from which you have jumped, called the message stack. The commands are similar to the commands for the point-pdl (stack) in the editor. See the section "What is a Zmacs Region?".

For example, suppose you are reading message 45 and then select message 22 from the summary line. The message stack looks like this:

45
.
.
.

Zmail has noted the fact that you were reading message 45 by pushing this information on the message stack. Now you use [Next] and you are reading message 23. (The message stack does not change, because it is easy for you to undo the effect of using [Next]; the idea of the message stack is to help when you have been moving around in a more arbitrary way.) You then type `58J` and message 58 becomes the current message. The stack now looks like this:

```
23
45
.
.
.
```

If you now want to return to where you were — message 23 — but do not remember the message number, you can press `c-U c-SPACE`. Message 23 becomes the current message and the message stack is:

```
45
.
.
.
23
```

This is called popping the message stack, because the top element is popped off and used as the current message. As you can see, it also gets tucked under the bottom of the stack. To get back to message 45, press `c-U c-SPACE` again. The message stack is now:

```
.
.
.
23
45
```

If you had wanted to get from message 58 directly to 45 without looking at 23, you could have pressed `c-U c-U c-SPACE c-U c-SPACE`. The `c-U c-U c-SPACE` command pops the 23 from the top of the message stack, but instead of becoming the current message, the 23 is simply discarded (not the message itself, just the stack entry.)

Stack entries are actually internal pointers, not message numbers. This means that if a message number changes — because other messages were expunged or the file was sorted — the message stack still points to the correct message, even though its number changed. And if a message is expunged, all stack references to it disappear.

Note that the stack entries keep track not just of the message itself, but also of what mail file it is in. This is helpful if you are using multiple mail files.

The following are the commands for using the message stack:

c-SPACE — Push the current message onto the stack. See the section "c-SPACE (Kbd) Zmail Command".

c-U c-SPACE — Pop the top message from the message stack and make it the current message.

c-U c-U c-SPACE — Pop the top message from the message stack and discard it.

[Jump (M)] — Selects an arbitrary message from the message stack. See the section "[Jump] Zmail Menu Item".

n Move to Point (m-X)
: Exchanges the current message and the top of the message stack. See the section "Move to Point (m-X) Zmail Command".

n Move to Default Previous Point (m-X)
: With an argument *n*, performs the same rotation as *n* c-m-SPACE and makes *n* the new default argument. Without an argument, uses the default. (The initial default is 3.)

Message Deletion Commands

Messages can be deleted when they are no longer wanted. Deletion does not actually mean the removal of a message; rather it means flagging a message for later removal. Messages flagged for deletion bear the status letter D and have the deleted property; these messages are passed over by the Next and Previous commands. The actual removal of deleted messages is called "expunging" and is done by the Save and Quit commands. See the section "Exiting Zmail".

The deletion and undeletion commands are:

[Delete] — Deletes the current message. (L), (M), and (R) offer options. See the section "[Delete] Zmail Menu Item".

D (Kbd) — Deletes the current message. Takes numeric argument. See the section "D (Kbd) Zmail Command".

c-D (Kbd) — Deletes current message and moves to previous message.

Delete Duplicate Messages (m-X)
: Deletes duplicate messages in the current sequence. See the section "Delete Duplicate Messages (m-X) Zmail Command".

[Undelete] — Removes the delete flag from a message. See the section "[Undelete] Zmail Menu Item".

U (Kbd) — Removes the delete flag from a message. See the section "U (Kbd) Zmail Command".

Click Middle on Summary Line
: Toggles Delete flag. See the section "Zmail Message Summary Line".

Other Zmail Commands

Commands for Editing the Recipient List of a Message

If you are editing a message with a long list of recipients, it is occasionally useful to be able to manipulate the addresses.

`h-F` (Kbd)	Move forward over an address.
`h-B` (Kbd)	Move backward over an address.
`h-T` (Kbd)	Transpose the two surrounding addresses.
`h-K` (Kbd)	Kill the next address.
`h-Rubout` (Kbd)	Kill the previous address.

Entering Mail Mode Recursively

`c-X M` (Kbd)	Enters mail mode recursively; the window configuration remains the same, but the Headers and Mail windows are reinitialized as if the Mail command had just been executed (Headers window contains the word "`To:`" followed by a blinking cursor; Mail window is empty.) Exiting recursive mail (either by sending the message or by aborting) returns to the higher level mail.

Adding Bug Lists to Zmail

You can add a new bug-mail recipient to the list of bug recipients. Two mechanisms are available.

1. Use the **:bug-reports** option to **defsystem**.

```
(:bug-reports (:name system
               :mailing-list mailing-list
               :documentation documentation-string))
```

name appears on the Zmail menu. *mailing-list* is the name of the mailing list to use. *documentation-string* is the mouse-line documentation for the menu item. For example,

```
(defsystem print
    (:pretty-name "In-House Printers"
     :default-pathname "sys: print;"
     :maintaining-sites :acme
     :default-module-type :lisp
     :bug-reports (:name "Print"
                   :mailing-list "bug-printer"
                   :documentation  "Report a bug in the hardcopy facility.")
  ...)
```

2. Use the function **zwei:add-bug-recipient**.

zwei:add-bug-recipient *name* &optional *documentation menu-name addressee-name*
Function

zwei:add-bug-recipient adds a new recipient to the menu available from [Mail (M)]. All arguments are strings. *name* is the name of the mailing list or recipient of the bug note; *documentation* appears in the mouse documentation line; *menu-name* is the name as it should appear on the menu. *addressee-name* is an explicit mail address. If *addressee-name* is not specified, it defaults to Bug-*system-name*. **zwei:add-bug-recipient** uses the site option **host-for-bug-reports** to determine the rest of the address.

The **:bug-reports** option in **defsystem** interacts with **zwei:add-bug-recipient** so you can specify the appropriate bug reporting mechanism for your applications system. See the section "**:bug-reports** option for **defsystem**".

Encrypting Messages

Zmail supports encryption. Commands are available both when you are composing mail and when you are reading mail. Encrypted messages contain a new header field to indicate that they contain encrypted text.

Encrypt Text (m-X) Encrypts a message. Use it after you have completed the message draft but before you send it. Zmail prompts for an encryption key that the recipient must provide in order to decrypt the message. This key can consist of plain alphanumeric text only. Punctuation or other funny characters are ignored. Upper and Lower case are equivalent. It converts the draft to a form that you cannot read. Decrypt Text is also available for message drafts. Both of these commands appear on the draft editor menu.

Decrypt Message (m-X)
Prompts for the encryption key and then displays an encrypted message as plain text. By this operation, you are only viewing the plain text form; use a numeric argument to store the plain text version in the mail file.

Text yanked by Forward and Reply prompts for a decryption key rather than yanking unreadable text.

The only encryption algorithm currently supported is the NBS algorithm, used by Hermes.

Getting Out of Trouble in Zmail

This chapter describes what to do if Zmail seems to be stuck and how to undo a command.

Recovering From Stuck States in Zmail

Zmail is a complex program and sometimes becomes stuck. This section lists a few common problems and what to do about them.

- *Everything looks correct, but Zmail does not respond to commands.* — Select another window and reselect Zmail. The most convenient way to do this is to press SELECT L SELECT M. Pressing c-m-ABORT might also work; this throws you back to top level, aborting any other command (for example, Mail, Profile). Before pressing c-m-ABORT, you might first try pressing ABORT.

- *Zmail does not respond to commands, and the process state is* Wait Forever. — Press SELECT L SELECT M or c-m-ABORT.

- *Zmail does not respond to commands, and the process state is* Output Hold. — Press FUNCTION ESCAPE. The window that appears might be in the Debugger; follow the instructions below the line "An error has occurred, and Zmail has entered the Debugger". After exiting the Debugger, a background window might remain on the screen overlaying part of the normal Zmail window; to deexpose it, press FUNCTION c-T.

- *Zmail does not respond to commands, and the process state is* Arrest. — Press FUNCTION - A (that is three keystrokes).

- *An error has occurred, and Zmail has entered the Debugger.* —
  Press ABORT to get out of the Debugger; this exits from one command level. For example, if you are in mail mode and an error occurs, ABORT gets you out of the Debugger and leaves you in Mail mode. You are not forced back to top level.

  Before you press ABORT, you might wish to send a bug report. Do this by pressing c-M. This puts you in a mail window with appropriate information about the system and machine you are using included in the message. Finish the message with an explanation of the circumstances that led up to the error. Send the bug report by pressing END.

- *A window pops up telling you that an error occurred in the background process.* — Press FUNCTION-0-S and see what the error is. A window should pop up with the Debugger in it. Typically it is a file system error or a host-down error for the file server containing your mail, but it could also be a program bug. Follow the instructions as for An error has occurred ...; pressing ABORT restarts the background process and puts you back into Zmail.

- *Another window is partially overlaying the main Zmail window.* — Click left on the main Zmail window. If this does not work, try pressing FUNCTION c-T, which gets rid of "temporary" windows such as pop-up menus.

- *Zmail obeys commands, but typeout remains on the screen following an error.* — Press REFRESH or FUNCTION REFRESH. It might also help to select another window and reselect Zmail (for example, press SELECT L SELECT M). If you are at Zmail top level, you might also try using [Configure] (to display only the message), and then [Configure] again (to display both message and summary).

- *After an error, Zmail does not obey commands, and the process state is* Nil. — Press FUNCTION 1 W. If the process state is still Nil, press SELECT L SELECT M.

- SELECT M *flashes the screen and refuses to select Zmail.* — Use the System menu: shift click right to get the menu, then use [Mail].

- *Zmail is irreparably stuck.* — Enter a Lisp Listener (SELECT L) and type Initialize Mail. See the section "Initialize Mail Command". Caution: any mail sequences currently in memory are lost. This operation reloads Zmail without disturbing the rest of the system.

- *The mouse is broken.* — Almost all Zmail mouse commands have keyboard or m-X equivalents. If the mouse is broken, you can use the keyboard and m-X commands.

Undoing Commands in Zmail

If you are in the middle of a command you did not mean to use, you can *abort* it. If you have executed a command that prompts for keyboard input or wants you to select messages from the summary window, press ABORT. If the command is asking you to choose something from a menu, click on Abort if that is a choice, or move the mouse outside the menu and see if it goes away. If you are inside mail mode, press ABORT. If you are in edit mode, press END. If you are choosing a filter or defining a filter or universe, use [Abort]. If you are choosing a universe, move the mouse outside the menu. If you are in profile mode, use [Exit]. If all else fails, pressing c-m-ABORT works, but might be a bit drastic.

If you execute a command or send a message and then change your mind, there are several options to try to *undo* the action:

Undo (m-X) — Undoes the last command; see the section "Undo (m-X) Zmail Command".

Redo (m-X) — Undoes the effect of the last Undo (m-X).

Revoke Message (m-X) — Tries to "get back" a message that has been sent; see the section "Revoke Message (m-X) Zmail Command".

[Continue (R)] — Allows you to resend a message you have already sent, giving it a *supersedes* header; see the section "[Continue] Zmail Menu Item".

Managing Your Mail

When you start getting more than a few messages a day, it becomes difficult to keep things in order. Messages requiring future action begin to pile up and just sorting through old messages leaves no time for new. For situations like this, Zmail provides the following management capabilities:

- Classifying messages by adding keywords to them
- Sorting messages based on their subject or other characteristics (filters)
- Working with groups of related messages (mail collections)
- Storing groups of related messages into separate mail files

Manipulating Messages

This chapter summarizes useful techniques for manipulating messages within a mail sequence.

Replying and Remailing

Replying

To reply to a message, click on [Reply] or press R. Zmail initializes the headers of the reply for you: the Subject is copied from the original message; the To and Cc fields include the original sender and recipients of the message. (The exact set-up depends on the mouse button you click and the options in your profile.) You can see the original message while you write your reply.

People frequently include some or all of the original message, indented four spaces, in their reply. You can yank (copy) the message into your reply by pressing c-X c-Y after starting your reply. To prune some of the less useful headers from a yanked message, press c-X Y. c-X c-Y takes numeric arguments which control in-

dentation and header pruning. See the section "Commands for Including Files and Prepared Text in Messages".

To reply to several messages at once use the following procedure:

1. Select a conversation using Select Conversation (m-X).

2. Click on [Map Over / Reply] in the main Zmail menu.

3. Yank all the messages in the conversation into the reply using c-X c-Y.

By setting profile options, you can make the [Reply] command do most of this automatically. See the section "Zmail Options for Replying to Mail ".

Forwarding, Redistributing, and Redirecting

To forward or redistribute a message, clicking right on [Mail]. [Forward] lets you edit the message or add to it. You can also invoke Forward by pressing F. [Redistribute] simply prompts for addressees and sends the message with the *Redistributed-by*, *To*, and *Date* fields added.

You can redirect a message using Redirect Message (m-X). Redirecting allows you to remove some or all of the original recipients and send the message to new recipients. See the section "Redirect Message (m-X) Zmail Command".

Moving a Message to a File

You can move a message to a file by clicking on [Move] or by pressing O. Pressing O prompts for filename to which to move the current message The message is first moved to a sequence and then to the file when you save out your files. Using [Move] moves the message to the *default move-destination file*. The initial setting for the default move-destination file can be defined using the profile editor (see the variable **zwei:*default-move-mail-file-name***). After you use [Move] the default move-destination is the last file to which a message was moved. Clicking right on [Move] pops up a menu offering a list of your mail files and several options for moving the message. See the section "Saving a List of Mail Files".

Hardcopy Message Command

Hardcopy Message (m-X)
: Hardcopies the current message. Note that you can also click Right on [Other] in the Zmail menu and select Hardcopy Message. Additionally, you can click Right on the message summary line, and then click Right on [Move] and select Hardcopy.

Editing

It is sometimes handy to edit a message saved in your mail file. To do so, select it as current and press c-R or click left on the message window. Press END when you finish editing.

Clicking m-Left on a file reference in the Zmail header window edits the file in an available Zmacs buffer.

Reordering

You can rearrange a mail sequence in two ways: by sorting the messages or by appending messages to one another.

If you click right on [Sort] a menu of sort keys and directions pops up. (See the section "[Sort] Zmail Menu Item".)

You can click right on the summary line of a message you wish to concatenate to another; then click on [Concatenate]. Clicking right on [Concatenate] allows you to choose to which message it gets appended. (See the section "Zmail Message Summary Line".)

You can place the messages you want to combine in a collection. (See the section "Creating a Mail Collection".) Then you can click on [Map over / Concatenate] to combine them. (See the section "[Map Over] Zmail Menu Item".) The messages that get appended are deleted.

Operating on Zmail Messages Referred to by the Current Message

Often, when you receive a reply to a message, you want to delete the original one or refer back to it. If your mail files are more than 30 or 40 messages long it might be difficult to find the original message. Zmail can help by searching for messages referenced by the current one.

The current message *references* a message *x* if it includes:

- a citation to *x* in an In-reply-to or References header
- the yanked-in headers of message *x*

Usually, this means that the current message is a reply to message *x*.

The following referenced message commands are available:

Select Referenced Message (m-X)
: Selects the referenced message as current.

Delete Referenced Messages (m-X)
: Deletes the referenced messages.

Append To Referenced Message (m-X)
: Appends this message to the referenced message.

Move In Place Of Referenced Message (m-X)
: Moves this message to where the referenced message is, and deletes the referenced message.

Select References (m-X)
: Creates a mail collection of all messages referenced by the current message.

Select Conversation by References (m-X)
: Defines a conversation and selects it as a collection. This command is very similar to Select References.

Delete Conversation By References (m-X)
: Deletes all the messages in a conversation.

Select All Conversations by References (m-X)
: Selects messages to which any message in the sequence refers, or that refer to any message in the sequence, recursively. See the section "Select All Conversations By References (m-X) Zmail Command".

Append Conversation by References (m-X)
: Append messages to which this message refers, or which refer to this message, recursively.

The commands with the word "reference" in their names use hash tables rather than searching. With a numeric argument, the Reference commands offer a menu of universes for searching.

If the current message has references to several messages, Select Referenced Message, Append To Referenced Message, and Move In Place Of Referenced Message ask which message to choose. Delete Referenced Messages and Select References choose all referenced messages.

To find the referenced message(s), Zmail looks in the current sequence. If the message is not there, Zmail tells you about the references not satisfied. If given a numeric argument, Zmail pops up a menu of other sequences to search first.

Often, though, you know in *advance* where referenced messages can be found. For example, you might store all your messages about hardware in a particular file. If you get a message about hardware, the messages it refers to are probably also about hardware, and thus they are probably in that file. You can give Zmail this type of knowledge by setting the *filter-universe alist* in profile mode. See the section "Filter-Universe Alist".

See the section "Testing Zmail Message Characteristics". See the section "Defining Zmail Message Search Spaces".

Classifying Messages

Zmail allows you to classify and categorize messages by adding keywords to them. Keywords are useful in many ways, among them:

Topic Indicators — Indicate the major topic of the message. If your work involves designing natural language interfaces, for example, you might use keywords such as `dictionary`, `parser`, and `syntax-checker`. The topic indicators you need depend on the sort of messages you get.

Classifiers — Indicate the type of message. For example, you might use keywords such as `bug`, `feature-request`, `documentation-bug`, and `issue` to categorize messages as bug reports, requests for features, reports of documentation bugs, and issues under discussion.

Status Flags — Indicate the status or priority of the message. For example, you might use a keyword such as `to-do` to flag messages that require you to do something and a keyword such as `timing-out` to flag messages on which you are awaiting action from other people. You could use `P1`, `P2`, and `P3` to indicate the priority of a message requiring further action.

Setting Zmail Keywords

[Keywords] — Adds the last used keyword(s) to the current message.

[Keywords (M)] — Adds keywords according to which filters the message satisfies. See the section "Zmail Filters".

[Keywords (R)] or L — Allows you to select or specify keywords for the current message.

You can use keywords in association with filters (see the section "Zmail Filters"). to semi-automatically tag messages. For example, you can associate filters and keywords as follows:

Filter	*Keyword(s)*
Grammar	Syntax-Checker
Dictionary	Dictionary
Parser	Syntax-Checker, Parser

With this scheme, a message about the grammar constructions the parser understands would get the keywords `Syntax-checker` and `Parser`. if you use [Keywords (M)].

You can save such filter-keyword associations in your Zmail Profile. To set the filter-keyword alist, click middle on [Filters] or [Keywords] in profile mode. Using [Filters (M)] allows you to alter the associations of a given filter; using [Keywords (M)] allows you to alter the associations of a given keyword. You will probably want to save the alist (and the filter definitions) in your profile, see the section "Zmail Profile Options".

For more information, see the section "[Keywords] Zmail Menu Item".

Saving Keywords

You can store keywords, so that they appear on the keyword menu before you have ever used them. This is useful if you anticipate needing particular keywords in the future. To do so, click left on [Keywords] in profile mode. A small editing buffer pops up containing the names of the mail files loaded into your Zmail. If there are any keywords already used in those files, they are listed on the line with the appropriate file. You can add or delete keywords from the those listed as well as add file-keyword lists to this buffer. The format looks like this:

```
acme>kjones>babyl.text::bugs,documentation-bugs,grammar,parser
```

The keywords are actually stored in the individual mail files. The list of keywords stored in a particular mail file includes all keywords associated with any message ever in the file, plus any you add using [Keywords]. The keywords list displayed in the menu is the union of the lists in all mail buffers.

Zmail Filters

Filters are sets of criteria to use in testing messages (see the section "Testing Zmail Message Characteristics"). You can use filters in association with keywords to sort messages or to move messages of a particular type to a separate mail buffer or collection. There are two types of filters:

1. Predefined filters

2. User defined filters

You can use predefined filters by clicking right on [Survey] or by clicking on Filters in the menu produced by clicking right on the message summary line.

You can define your own filters using New Filter in the menu produced by [Survey (R)], see the section "Creating Zmail Filters".

Mail Buffers

Listing Zmail Buffers, Mail Files and Collections

Your current Zmail session consists of mail buffers (with associated mail files) and collections. Collectively these are referred to as *sequences*. You can list all the sequences in your current Zmail session as well as any mail files (known to your profile) that have not yet been read in. You do this with List Sequences (m-X). The items on the list are mouse sensitive.

Selecting Mail Buffers and Files

To select another mail buffer or file, use [Select (R)] (see the section "[Select] Zmail Menu Item"). Click on the name of the desired buffer or file (if it appears) or use [Read/Create file] to specify the name of a file. Specify the name of the file to be read into a buffer. The buffer then takes the name of the file. The following are all possibilities:

- *The buffer exists:* it is selected.
- *The buffer doesn't exist, but a file of the same name does:* the file is read into a buffer and selected.
- *Neither exist:* a new buffer is created. Saving the buffer creates a new file.

Using [Select (L)] returns you to the previously selected sequence. Subsequent left clicks alternate between the two sequences.

`c-m-L` is like [Select (L)]. With an argument of 0 it works like [Select (R)]. With an argument of 1 or greater, it works as in Zmacs and selects from the stack of previously selected sequences (see the section "Changing Buffers").

Copying a Message to Another Buffer

[Move (R)]	Pops up a menu of mail files from which you select an existing file or collection or create a new one to which to move the current message. See the section "[Move] Zmail Menu Item".
[Move]	Copy current message to same buffer as the last move.
[Move (M)]	Move the message to the file(s) corresponding to any filters you have defined.

A filter-mail file alist associates a single mail file with each filter in the list. Using [Move (M)] moves the message to the files corresponding to the filters in the list satisfied by the message.

For example, if you associated your Hardware filter with the file HARDWARE.XMAIL and a Software filter with SOFTWARE.XMAIL, you could use [Move (M)] to move your messages to the appropriate mail buffer. With two or three mail files and filters, this is a very powerful tool.

To move a group of related messages to another mail buffer, you have to use mail collections (see the section "Zmail Mail Collections").

As with [Select], if the buffer doesn't exist, it is read in or created.

Saving a List of Mail Files

Your can store a list of mail files in your Zmail profile, so that their names appear in the various mail file menus. Use [Mail files] in profile mode. Zmail does not load the files on this list automatically (see the function **zwei:preload-zmail**), it

just makes the names easily accessible by placing them in the menu offered by [Select (R)]. As you request Zmail to load files into your Zmail, their names appear in bold at the top of the menu.

Mail Collections

To work with a group of related messages, you first put them all in a mail collection.

Creating a Mail Collection

There are three ways to create a mail collection:

1. Starting with a single message

2. By using filters

3. By marking individual messages

Creating a Mail Collection Starting with a Single Message

To start a new mail collection with the current message, click right on [Move] (see the section "[Move] Zmail Menu Item"). The items in the [Move] menu that relate to creating mail collections are [New Collection] and [Recycled Collection]

If you want to give the collection a name, click on [New Collection]. Zmail prompts you for a name for the new collection. This name shows up in the menu of mail sequences for subsequent [Move] and [Select] commands.

If you do not care about naming the collection, click on [Recycled Collection]. Zmail will create a temporary collection. The first such collection is named Temp, the second Temp-1, and so on. If you kill one of these collections, its name is reused (*recycled*) for the next temporary collection.

Creating a Mail Collection by Using Filters

You can use filters to create a collection in two ways:

1. Using predefined filters associated with the message. Click right on the summary line of a message and then click on Filters. A list of filters based on the header fields of the message pops up. You select one of these filters and Zmail searches through the current sequence for any messages that satisfy that filter and selects them along with the current message as a collection.

2. Choosing a filter from Filter Selection Display. Click right on [Select] and then click on Filters. Select one of the filters from the display. The messages in the current sequence that satisfy that filter are selected as a collection.

To include messages from more than one sequence, select or define a universe by clicking left on Universes in the Filter Selection Display. See the section "Selecting Zmail Universes". See the section "Defining Zmail Message Search Spaces". Messages from the sequences in the universe are then filtered to select a new collection.

Creating a Mail Collection by Marking Individual Messages

The menu produced by clicking right on [Select] offers the item [Mark Survey]. See the section "[Select] Zmail Menu Item". Clicking on this permits you to mark the summary lines of messages in the current sequence to be selected as a collection. You mark messages using the mouse or the keyboard. Clicking right or pressing `END` terminates marking and selects the marked collection. The mouse clicks and keyboard commands are:

Mouse click	*Keyboard command*	*Meaning*
Left	`SPACE`	Toggle marking of message for inclusion in the collection.
	`c-M`	Marks all messages for inclusion in the collection.
	`c-sh-M`	Unmarks all messages.
	`c-N`	Move summary cursor forward.
	`c-sh-N`	Toggles the marked state of the current message and then moves the summary cursor forward to the next message. (Equivalent to `Space` followed by `c-N` but fewer keystrokes.)
	`c-P`	Move summary cursor backward.
	`c-sh-P`	Toggles the marked state of the current message and then moves the summary cursor backward to the previous message.
	`c-V`, `c-m-V`	Scroll summary window forward.
	`m-V`, `c-m-sh-V`	Scroll summary window backward.
	`HELP`	Print this help text.
Middle	`ABORT`	Abort the mark survey operation.

Right	END	Finish marking and select the marked collection.

Operating on a Mail Collection

Once you have a collection there are a number of things you can do with it:

Rename it
: Rename Sequence (M-X) prompts you for a new name for the collection.

Select any collection as current sequence
: Click right on [Select]. See the section "[Select] Zmail Menu Item".

Add message to collection
: Click right on [Move]. Message is not deleted if it is just moved to a collection.

  Click right on its summary line, then click on [Move].

Remove message from a collection
: Click right on its summary line, then click on [Remove]. ([Remove] exists as a menu choice only when the current sequence is a collection). The message disappears from the collection but is not deleted from the buffer the collection is drawn from.

To do something to every message in a mail collection (or mail buffer) use [Map over (R)]. See the section "[Map Over] Zmail Menu Item".

Three particularly useful commands are:

[Map over / Move (M)]
: Like using [Move (M)] for each individual message; that is, each message is moved to the appropriate file, based on the filter-mail file alist. See the section "Testing Zmail Message Characteristics".

[Map over / Move / By Individual Filters]
: A synonym for [Map over / Move (M)].

[Map over / Keywords (M)]
: Like using [Keywords (M)] for each individual message; that is, each message is given the appropriate keywords, based on the filter-keyword alist. See the section "Setting Zmail Keywords".

Operating on a Message in a Mail Collection

Remember that mail collections are sequences of messages drawn from mail buffers. The message exists in its original mail buffer. Therefore, *any change you*

make to the message appearing in the mail collection (for example, adding keywords to it) is reflected in the mail buffer, and vice versa.

A single message can exist in several mail collections (some of which could have been created by filtering or marking on *another* mail collection). Changes made to any image of the message are reflected in the buffer and all collections in which it appears.

Action	*Effect*
Deleting a message	All images of message marked as deleted.
Expunging buffer	Deleted message disappears from buffer containing actual message and all collections in which it appears.
Expunging collection	Deleted message disappears from that collection only. (The actual message and all remaining images are still marked as deleted.)

Operating on a Group of Messages

To do something to a group of messages, create a temporary collection containing just those messages)see the section "Creating a Mail Collection by Marking Individual Messages"). Then you can use [Map Over] to operate on the collection. See the section "Operating on a Mail Collection".

Saving, Expunging, Killing, and Renaming

To expunge and save your mail sequences

[Save] or `S`	Expunges any buffers or collections with deleted messages and saves all buffers. See the section "[Save] Zmail Menu Item".
Start Background Save (`m-X`)	Suppresses background mail checks and starts a save in the background. This allows you to compose and send mail messages while the save is being done.
[Quit] or `Q`	Expunges any buffers or collections with deleted messages, saves all buffers, and returns to the window from which Zmail was called. See the section "[Quit] Zmail Menu Item".
[Save (M)] or `E`	Expunge current sequence.
[Save (R)]	Pops up a menu of all your mail sequences. From the menu you can determine what [Save] or `S` would do and either modify it or make it happen. The menu also allows you to use [Kill] on a buffer or collection, that is, simply get rid of Zmail's image of it. Files on disk are not affected.

To rename a mail file, perhaps because the host it usually is stored on is down, use the Rename Sequence (m-X) command. Zmail prompts you for a new filename for the buffer and you can then save the file to a different location.

To turn collections into mail files use [Map over / Move] (see the section "[Map Over] Zmail Menu Item"). to change the collection into a real buffer.

Hints for Using Keywords, Mail Collections, and Mail Files

Using the mechanisms described in this chapter is an art. Here are some suggestions.

Familiarize yourself with the range of options in profile mode (see the section "Zmail Profile Options"). Try out different settings.

Decide on some useful topic, classifier, and status keywords and store them in your default mail file. Start using them; new ones added later will be stored automatically. After you've gained some experience with them, define some filters and create a filter-keyword alist so you can add keywords with click middle.

When your mail file starts getting big - 100 messages is certainly big for a default mail file, 200 for others - split it into two files by following these steps:

1. Filter on some message attribute to make a mail collection. If you've chosen your keywords well, you can just use a keyword filter. (To simply split the file into old and new messages, use [Before] on the Filter Creation Display.)

2. If you used a keyword filter in step 1, use [Map over / Unkeywords] to remove the keyword you filtered on, since everything in the new collection has that keyword.

3. Use [Map over / Move / Find file] to move the collection to another file.

4. Use [Save] to save your buffers and dispose of the collection.

When you have two or more mail files, create a filter-mail file alist so that you can use [Move (M)]. Create a filter-universe alist so that you can use the referenced message commands more effectively.

With files, keywords, and alists set up, your response to a new message might be among the following:

- Delete it.
- Reply to it.
- Put some keywords on it.
- Move it to another file.

- Look at the message it refers to.
- Delete the message it refers to.

If you have a lot of new mail, you might not want to read it in the order in which it arrived.

1. Use [Unseen] on the Filter Selection Display to put the new mail in a mail collection.
2. Use [Map over / Move (M)].
3. Read the new mail in each of the files to which it was moved.

(You must have a Filter-Mail File alist set up in order to use this procedure.)

Reference Information

Fundamental Techniques

Customizing Zmail

The Profile command allows you to customize Zmail by setting various display and command options to your personal taste.

You can also customize the Zmail user interface by using the (m-X) Set Key command to temporarily bind a Zmail command to a keystroke. For example, you can use Set Key to temporarily bind the Append Conversation by References command to s-S.

You can set an option temporarily or permanently, the latter by saving the option in your *Zmail Profile*.

Classes of options you can set in your *Zmail Profile* include the following:

- Format used for hardcopies of messages
- Mail-file attributes
- Lists of mail files and other objects that Zmail knows about at startup
- Associations between certain objects
- Clicking Middle for many top-level commands
- Screen configurations

- Default actions taken when reading, sending, replying to, or forwarding mail
- Command Tables

Customizing is done in *profile mode*, entered by clicking on [Profile] in the command menu at top level. The profile mode display (Figure !) shows the text of your profile and the current settings of various options.

Setting and Saving Zmail Options

Option settings are stored in seven distinct places:

1. *The Zmail environment:* the way the options are actually set at the moment.
2. *The defaults:* the way the options are actually set before you alter them.
3. *The editor buffer:* the in-memory buffer of your profile.
4. *The source version of your profile:* on disk.
5. *The compiled version of your profile:* also on disk.
6. *Mail buffers:* options associated and stored with the individual mail buffers.
7. *Mail files:* options associated with a mail buffer saved as a file.

Enter profile mode by clicking on [Profile] in the Zmail menu. The top portion of the window looks like Figure !. The User Options pane works like an Accept Variable Values menu. You click on the various values to change options. The boxed items above and below the User Options pane are menu items that bring up additional menus for general operations on your mail files or on your profile.

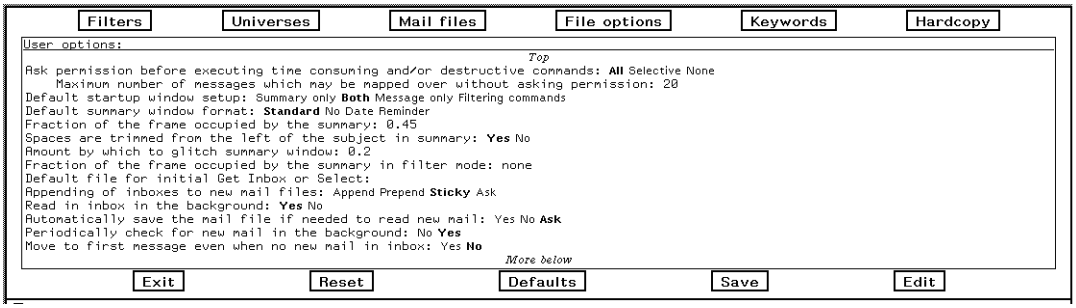


Figure 109. Profile Mode Menu and Interaction Pane

The lower half of the window is an editor buffer into which Lisp forms are inserted automatically when you select options in the User Options pane. This is what is

saved as your zmail-init.lisp file. You do not have to edit this file yourself; Zmail takes care of that for you.

The simplest way to use profile mode is:

1. Make the changes you want using the menu items or user options window. For a list of the various options and what they mean: See the section "Zmail Profile Options".

2. Click on [Exit] to leave profile mode. Check to see that you like your changes.

3. To save your changes, reenter profile mode and click Left on [Save]. Answer *yes* to any questions about inserting changes or recompiling your file. At this point Lisp code corresponding to your option settings is stored in your profile. Options changed using [File options] or [Keywords] are stored in the individual mail buffers and are saved when you save the particular mail file.

What [Save] actually does is move option settings from the environment (where you altered them in the first step) to the editor buffer, then from the editor buffer to the source copy of your init file, and finally from the source file to the compiled file (by recompiling).

You can undo all the settings you have made by clicking on [Defaults], which returns all the variables to their system defaults. You can reset all the variables to the values in your init file by clicking on [Reset], which loads your init file again.

Testing Zmail Message Characteristics

Filters are logical predicates that apply to messages. They take a message as input and return a True or False answer based on its characteristics. For example, a filter might test whether the message was sent to a particular person or on a particular date. If the answer is True, the message is said to *satisfy* the filter.

Zmail commands use filters in one of two ways:

1. The [Survey], [Jump], and [Select] commands form subsets of all messages that satisfy a particular filter. ([Select] forms a mail collection; the other two form the subset implicitly.)

2. The [Keywords] and [Move] commands act upon a single message in a particular way depending upon which filters the message satisfies.

Selecting Filters

Filters are of two types:

1. *Predefined filters* are simple, and come in four varieties:

   - [All] is a filter satisfied by any message.

- *Keyword filters* test whether the message has a particular keyword or any keywords.

- *Property filters* test whether the message has a particular property, such as Answered.

- *Header filters* test whether the message headers meet specified characteristics, such as a Subject field that includes the word "Lisp".

2. *User-defined filters* are arbitrarily complex logical expressions whose operands are predefined filters and other user-defined filters.

When you invoke a Zmail command that uses a single filter (a command that forms a subset of messages) you select the filter you want from the menu shown in Figure !. By first using [Not], you can negate the action of the filter you choose, that is, select those messages that *fail* to satisfy the filter.

Creating Filters

Filters are Lisp functions, constructed using the menu-based programming tool shown in Figure !. This display can be obtained in two ways:

- By using [New filter] in Figure !: use this when none of the existing (pre- or user-defined) filters is suitable.

- By using [Filters] in profile mode (Figure 109) before using [New filter]: use this to define a filter whose utility you anticipate before you actually need to use it. The filter will be saved in your profile. See the section "Saving Zmail Filters".

The Filter Creation Display is divided into three main sections:

- The *summary window*. You can click left on a message's summary line to select predefined filters based on its characteristics.

- The menu items are the primary tool for defining the filter. The menu items are divided into four rows:

  ° A row of *programming* items, [Not], [And], [Or], and [Close]. The first three are logical functions; the last closes a level of parentheses in the expression being constructed.

  ° A *documentation* item, [Documentation], which adds documentation to a filter. This documentation appears as the mouse documentation line when you are pointing with the mouse to that filter in the filter selection menu.

  ° A row of *processing* items, [Sample], [Done], and [Abort]:

    [Sample] Displays the summary lines of messages that satisfy the filter you have defined so far.

    [Done] Exits definition mode and executes the command that called for the filter.

    [Abort] Aborts the command.

  ° Two rows of *filter menus*, similar to the Filter Selection Display. These allow one filter to call another.

- The *editor buffer* displays the filter as it is being created.

So, how do you actually define a filter? Let's walk though an example. Suppose we want a filter that selects messages dealing with hardware. The first thing to do when defining a filter is to use the menu item above the editor buffer: right for a menu of existing filters to edit, or click left and give a name to create a new filter. Let's call this filter "Parser".

The next step is to determine the explicitly definable characteristics of the messages we're looking for. In this case, we might decide that messages about the parser are either from PJF (but only if dated after 2/5/86), or contain the word "parser" in the Subject field. Expressed in Lisp, the filter looks like:

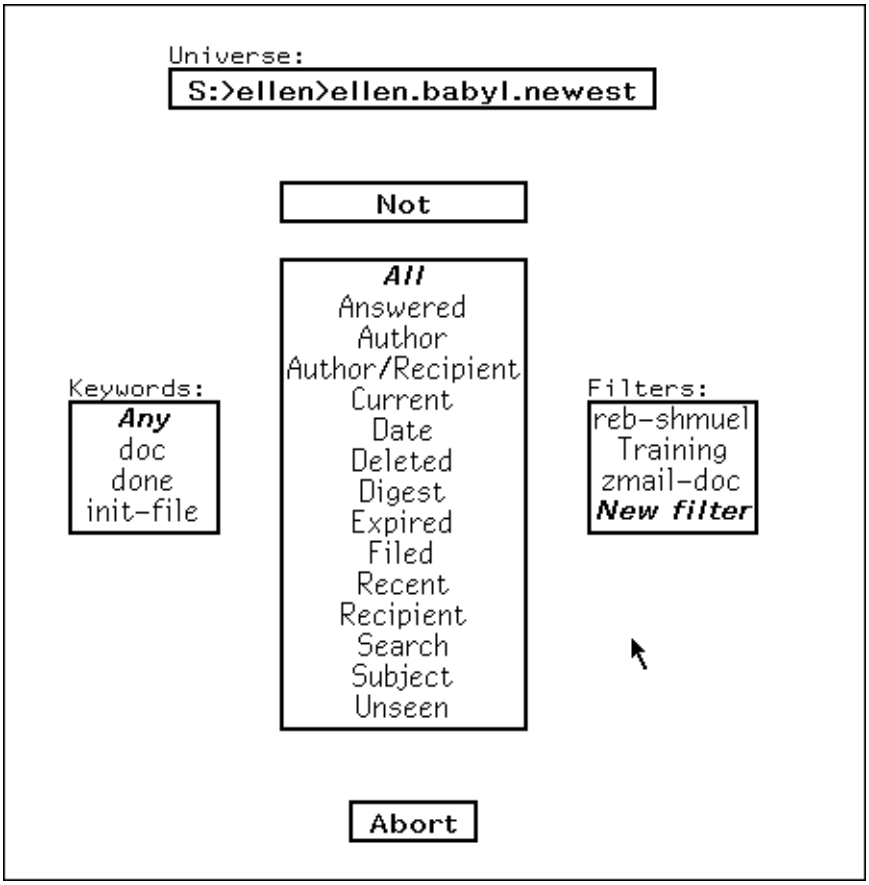


Figure 110. Filter Selection Display.

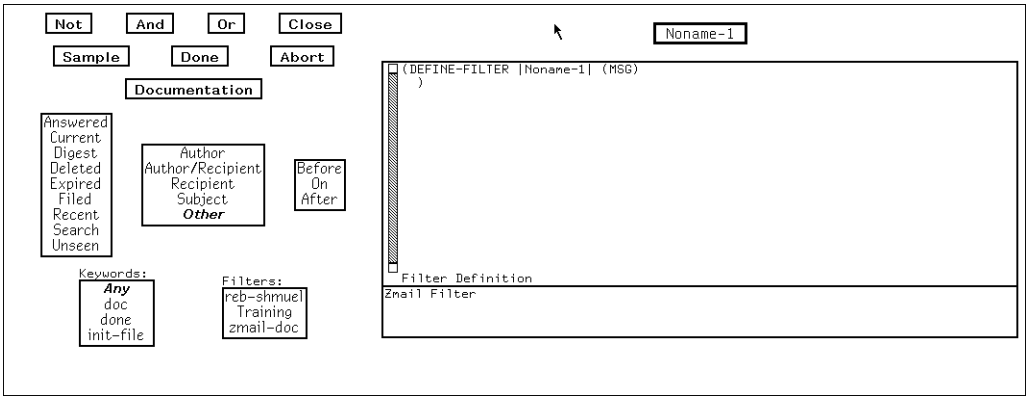


Figure 111. Filter Creation Display.

```
(DEFINE-FILTER |Parser| (MSG)
  "Messages relating to the Parser"
  (OR (AND (MSG-HEADER-RECIPIENT-SEARCH ':FROM #"PJF")
           (MSG-DATE-GREATERP "5-Feb-86"))
      (MSG-HEADER-SEARCH ':SUBJECT #"Parser")
      ))
```

To create it, we follow these steps, clicking on the items in the menu:

- [Or]
- [And]
- [From], type `PJF`, and press `RETURN`
- [After], type `2/5/86`, and press `RETURN`
- [Close]
- [Subject], type `parser`, and press `RETURN`
- [Documentation] type in a mouse documentation string followed by `RETURN`
- [Done]

Notice the [Close] command. It *closes* the [And]. The optional mouse documentation string, added in the next to last step, is very useful; it shows up as the mouse documentation line for the filter whenever you are asked to select a filter from a menu.

Note that If you click on a message in the summary window after clicking on [Author], [Author/Recipient], or [Recipient] in the filter selection menu, and there is only one possible address, a small confirming menu appears.

Saving Zmail Filters

Any filters you have defined during the current login session show up in the filter menus, but they are gone when you cold boot the next time. To save a filter permanently, you must save it in your profile.

1. Enter Profile mode (See the section "Setting and Saving Zmail Options".)
2. Click left on [Filters]. A menu pops up listing the filters you have saved in your profile plus any you have defined in your current Zmail session. The filters that are already saved in your profile are highlighted.
3. Click on the filter(s) you want to save. They appear highlighted.

4. Click on [Do It].

5. After the menu disappears, click on [Save] to save your profile.

The filters you have selected are then saved.

Defining Message Search Spaces

Universes are programs that define sets of messages. For example, the universe "Hardware" could be the set of messages in two mail files, PRINTER-HARDWARE.BABYL and 3640-HARDWARE.BABYL. Universes are dynamic objects; if the contents of one of the files were to change, the contents of "Hardware" would change. Like filters, universes come in both predefined and user-defined varieties. Universes are implemented using flavors. You can define arbitrary universes; see sys: zmail; universe.lisp for information.

Zmail commands use universes in one of two ways; in both cases, the universe acts as a search space:

- The [Survey], [Jump], and [Select] commands use universes to define the set of messages from which a filter extracts its subset.

- The referenced message commands use universes to find messages related to one you are looking at. See the section "Operating on Zmail Messages Referred to by the Current Message".

Selecting Universes

When you use a universe in conjunction with a filter ([Survey], [Jump], or [Select] command), you do so by using the universe menu item in Figure 110 prior to selecting a filter. This menu item displays the universe to be used with the filter you select; the usual default is the rest of the current mail file. Using the universe menu item causes a menu to pop up similar to the one in Figure !.

Using this menu, you can choose one of the following predefined universes:

- The messages in a particular mail file, buffer, or collection.

- The union of messages in all files, buffers, and collections listed in the menu.

- The union of messages in all buffers and collections.

- The messages in the current buffer or collection following the current message.

- The messages in the current buffer or collection preceding the current message.

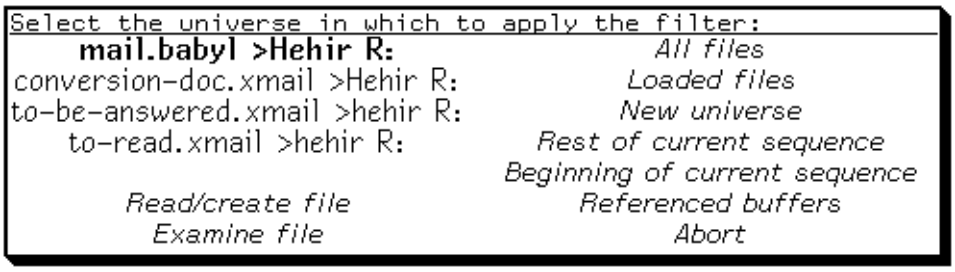


Figure 112. Universe Selection Display.

Creating Universes

New universes are defined using a menu tool similar to the Filter Creation Menu. You can obtain this display in two ways:

- By using [New universe] in Figure 112: use this when none of the existing (predefined or user-defined) universes is suitable.
- By using [Universes] in profile mode (Figure 109) before using [New universe]: use this to define a universe whose utility you anticipate before you actually need to use it. The universe will be saved in your profile. See the section "Saving Zmail Universes".

The Universe Creation Display is quite similar to the Filter Creation Display. By clicking on menu items, you construct a universe as unions, intersections, and complements of mail files, buffers, collections, and universes. These include the special universe [Current] (the current sequence), and the special universe [Loaded] (all loaded buffers). The precise definitions of the set operations are as follows:

Union A message is in the union of *n* universes if it is in any of the universes in the union.

Intersection A message is in the intersection of *n* universes if all contain the same message (not a copy from another buffer).

Complement A message is in the complement of a universe if it is not in that universe, but is in [Loaded].

One caveat about the use of mail collections in universes: be sure that the collection you name actually exists when you use the universe. A universe used several times or stored in your profile might refer to a collection that no longer exists. If so, Zmail attempts to find a mail file with the same name as the collection. This results in an error.

Saving Universes

To save a universe in your profile, use [Universes] in profile mode. When you save your profile, the universes you select are saved. Those universes, plus any you have defined during the current login session, show up in the universe menus.

Filter-Universe Alist

The filter-universe alist is an *association list* of filters and universes. It tells Zmail where to look for the referenced message: if the current message satisfies a filter on the list, the corresponding universe is searched. If other than one undeleted message is found in the search by Select Referenced Message, Append To Referenced Message, or Move In Place Of Referenced Message, Zmail pops up a menu of the messages.

As an example, suppose you kept your hardware-related messages in the files PRINTER-HARDWARE.BABYL and 3640-HARDWARE.BABYL. The union of these files is the universe "Hardware". See the section "Defining Zmail Message Search Spaces". On the alist, we pair the filter "Hardware" with the universe "Hardware". See the section "Creating Zmail Filters". Henceforth, if the current message concerns hardware, Zmail looks for references to the message in the two files/buffers that make up the universe.

To set the filter-universe alist, use [Filters (M)] or [Universes (M)] in profile mode. While in profile mode, you should also use the User Options Window to set your options so that replies you send automatically include either an In-reply-to field or the yanked-in message itself. That way, people receiving your replies can use the referenced message commands.

Zmail Profile Options

The profile menu allows you to customize Zmail. Here are the various options in the order in which they appear in the menu. Many of them are self-explanatory.

Zmail Option for Controlling Queries for Destructive Commands

zwei:*ask-before-executing-dangerous-zmail-commands* *Variable*

Profile Option: Ask permission before executing time consuming and/or destructive commands.

Controls whether Zmail asks for confirmation before starting a time consuming or potentially dangerous operation.

The choices are:

`All`	The default. All dangerous commands should ask for permission.
`Selective`	Only the dangerous commands listed in the "Commands which will ask permission" profile option ask for permission.

None — No dangerous commands ever ask for permission. This includes Map Over and means that the setting of "Maximum number of messages which may be mapped over without asking permission" is ignored. See the variable **zwei:*too-many-msgs-query-threshold***.

zwei:*selected-dangerous-zmail-commands* *Variable*

Profile Option: Commands which will ask permission.

Controls the list of commands considered dangerous when typed from the keyboard. This option depends on the setting of "Ask permission before executing time consuming and/or destructive commands" (**zwei:*ask-before-executing-dangerous-zmail-commands***). If "Ask permission ..." is **selective**, you can select the commands that you want to have query you before execution from this list. If "Ask permission ..." is **all** or **none**, this option does nothing.

The commands in the list are:

zwei:com-zmail-save-mail-file
: `c-X c-S`

zwei:com-zmail-kill-sequence
: `c-X K.`

zwei:com-zmail-quit
: `Q` from the keyboard.

zwei:com-zmail-save-all-mail-files
: `S` from the keyboard.

zwei:com-zmail-expunge-sequence
: Expunge from anywhere.

zwei:com-zmail-map-over
: Map Over from anywhere.

zwei:*too-many-msgs-query-threshold* *Variable*

Profile Option: Maximum number of messages which may be mapped over without asking permission.

Controls how large a collection of messages you can operate on automatically without confirming the command. The default is 20.

Zmail Options for Window Configuration

zwei:*default-initial-window-configuration* *Variable*

Profile Option: Default startup window setup

Controls the configuration of your Zmail windows when you invoke Zmail for the first time. The choices are: Summary only, Both (the default), Message only, and Filtering Commands.

zwei:*default-summary-template* *Variable*

Profile Option: Default summary window format

Specifies the format for message summary lines. The possible values are:

Format	*Headings for Summary Window*
Standard	`No. Lines Date From→To Subject or Text`
No Date	`No. Lines From→To Subject or Text`
Reminder	`No. Date Time Subject or Text`

zwei:*summary-window-fraction* *Variable*

Profile Option: Fraction of the frame occupied by the summary

Controls the percentage of the screen occupied by the summary window in the default window configuration. The default is 45 percent, which means that the command window divides the screen in half, with approximately equal sized summary and message windows above and below respectively.

zwei:*summary-subject-trim-spaces* *Variable*

Profile Option: Spaces are trimmed from the left of the subject in summary

zwei:*summary-scroll-fraction* *Variable*

Profile Option: Amount by which to glitch summary window

zwei:*filter-summary-window-fraction* *Variable*

Profile Option: Fraction of the frame occupied by the summary in filter mode

Zmail Options for Reading Mail

zwei:*zmail-startup-file-name* *Variable*

Profile Option: File read in at startup

Your mailbox file.

zwei:*new-mail-file-append-p* *Variable*

Profile Option: Appending of inboxes to new mail files

Controls the order in which messages appear in a new mail file you create. The choices are:

Append	New mail files append messages.
Prepend	New mail files prepend messages.
Sticky	(The default.) New mail files inherit whether they append messages from the current buffer.
Ask	You are queried when creating a new mail file as to whether it appends messages.

zwei:*complete-get-inbox-in-background* *Variable*

Profile Option: Read in inbox in the background

zwei:*autosave-if-inbox-requires-save* *Variable*

Profile Option: Automatically save the mail file if needed to read new mail.

Controls the whether or not your mail file is written out when you do Get Inbox on systems that do not support multiple renamed inboxes (UNIX and ITS).

The choices are:

Yes	Automatically write out the file when a Get Inbox is done.
No	Do not write out the mail file automatically. Get Inbox tells you that you cannot read new mail until you have saved your mail buffer.
Ask	Tells you that your mail file must be saved before getting new mail and asks your permission to save it.

zwei:*inhibit-background-mail-checks* *Variable*

Profile Option: Periodically check for new mail in the background

Controls the checking of your inbox for new mail. If you leave it *yes* (**nil**, the default), Zmail will check periodically and notify you if there is new mail. If you change it to *no* (**t**), this action will be inhibited.

zwei:*always-jump-after-get-new-mail-from-inbox* *Variable*

Profile Option: Move to first message even when no new mail in inbox

zwei:*always-select-saved-current-msg* *Variable*

Profile Option: Reselect previous current message even if current message in sequence

When set to **t** (the *yes* response to the profile question), saves your place in a sequence when you select another sequence. For example: you come across an interesting message (say, #200) in your babyl file and do Select Conversation by References. Messages #200, #225, #250, and #300 are selected. You look at the messages and reselect your babyl file. If **(zwei:*always-select-saved-current-msg*)** is **t**, you are returned to message #200 in your babyl file. If **(zwei:*always-select-saved-current-msg*)** is **nil** (the default), you are returned to message #300 in your babyl file.

zwei:*run-gmsgs-p* *Variable*

Profile Option: Run GMSGS before getting new mail

If you keep your mail on an ITS host at M.I.T. and use the GMSGS facility, this option allows you to use it from Zmail.

zwei:*gmsgs-other-switches* *Variable*

Profile Option: Other switches to supply to GMSGS server

Allows you to specify arguments to the GMSGS command if you keep your mail on an ITS host at M.I.T.

Zmail Options for Saving Mail

zwei:*query-before-expunge* *Variable*

Profile Option: Show headers and ask before expunging deleted messages

Controls whether you are asked for confirmation when messages are being expunged from the mail file. If you leave it *no* (**nil**, the default), you are not asked. If you change it to *yes* (**t**), the headers of the messages to be expunged are displayed and you are asked to confirm the expunge.

zwei:*inhibit-background-saves* *Variable*

Profile Option: Automatically save buffer after reading inbox

Controls the automatic saving of your mail buffer. If you leave it *yes* (**nil**, the default), your buffer is automatically written out when your inbox has been read. If you change it to *no* (**t**), the buffer is not saved until you save it explicitly.

To control the query on the `S` (Save from Keyboard) Command: See the variable **zwei:*ask-before-executing-dangerous-zmail-commands***.

Zmail Options for Sending Mail

zwei:*mail-middle-mode* *Variable*

Profile Option: Middle button on Mail command

Controls the action of [Mail (M)]. The choices are: Bug (send a bug message, the default), Mail, Forward, Redistribute, and Local.

zwei:*default-mail-window-configuration* *Variable*

Profile Option: Default window configuration when mailing

Allows you to specify how the window is configured in mail mode.. The choices are:

Both	The message is composed in the the lower (message) window, the command pane and the summary window remain as usual.
Experimental	The message is composed in the lower (message) window. The command pane is divided into three sections providing a variety of operations.
Send	(The default.) The screen is divided into two windows, the header window and the message window.
Message only	The screen consists of one window with the headers at the top.

zwei:*draft-editor-end-key-treatment* *Variable*

Profile Option: Effect of `END` key in *Headers* window.

Controls what `END` does when you press it in the headers window when you are composing a message. The choices are:

Send	Pressing `END` sends the message, pressing `c-END` moves you to the message window.
Both Send	Pressing either `END` or `c-END` sends the message.
Add Text	(The default) Pressing `END` moves you to the message window. Pressing `c-END` sends the message.
Both Add Text	Pressing either `END` or `c-END` moves you to the message window.

Pressing `END` in the message window always sends the message.

zwei:*header-window-nlines* *Variable*

Profile Option: Number of lines (or fraction) occupied by headers in mail mode

Controls the number of lines in the header window in mail mode. The default is 3.

zwei:*prompt-for-missing-headers* *Variable*

Profile Option: Use the minibuffer to read missing headers

zwei:*require-subjects* *Variable*

Profile Option: Require subjects on outgoing messages

Controls whether or not you are required to supply a `Subject:` line on messages you send. The choices are:

Yes (**t**, the default.) Require a `Subject:` line on each message.

No (**nil**) Do not require a `Subject:` line. You can add one yourself if you want one.

On bug reports Require a `Subject:` line on bug reports but not on other messages.

Initial but not required
Supply a `Subject:` line in the generated header for a message, but do not require that it be used.

zwei:*send-header-format* *Variable*

Profile Option: Format of headers sent

The choices are: Short, Long, Include personal (the default), and Use original.

zwei:*local-mail-header-force* *Variable*

Profile Option: Header force for local messages

Controls the format of headers on messages sent on your local system. The possibilities are none, RFC733 (Arpanet standard), Network, and ITS (M.I.T. Incompatible Time Sharing). The default is ITS.

zwei:*local-mail-include-subject* *Variable*

Profile Option: Local mail starts out with a subject

zwei:*default-reply-to-list* *Variable*

Profile Option: Default initial Reply-To list.

Allows you to specify a list of recipients of replies to messages.

zwei:*default-cc-list* *Variable*

Profile Option: Default initial Cc list

Allows you to specify a list of recipients of copies of messages you send.

zwei:*default-bcc-list* *Variable*

Profile Option: Default initial Bcc list

Allows you to specify a list of recipients of "blind carbon copies" of messages you send.

zwei:*default-fcc-list* *Variable*

Profile Option: Default initial Fcc list

Allows you to specify a list of files to which copies of your messages are sent.

zwei:*default-mail-buffer-major-mode* *Variable*

Profile Option: Default major mode when composing messages.

Controls the editing mode in which mail buffers and reply buffers start out. The choices are:

Text
Fundamental
Lisp

The default is Text. You can set the mode as you can for a Zmacs buffer. See the section "Zmacs Major Modes".

zwei:*default-draft-file-name* *Variable*

Profile Option: Default file for saving draft

Allows you to specify a pathname to use for saving draft messages.

zwei:*mail-file-for-drafts* *Variable*

Profile Option: Mail file to store drafts in

Zmail Options for Replying to Mail

zwei:*reply-mode* *Variable*

Profile Option: Default reply to

Controls the automatic generation of to and cc fields in the header of a reply. The default is to reply to all addressees in the original message. Possible reply modes are:

All — Send the reply to everyone who saw the original message. Headers are:

`To:` *old From*
`To:` *old To*
`Cc:` *old Cc*

That is, the `To:` field of the reply becomes the old `From:` and `To:`, and the `Cc:` field of the reply becomes the old `Cc:`.

All-Cc — Reply is primarily for original sender, but is of interest to all who saw the original message. Headers are:

`To:` *old From*
`Cc:` *old To*
`Cc:` *old Cc*

Cc-All
Reply is primarily for original recipients, but is also of interest to original sender and `CC:` recipients. Headers are:

`To:` *old To*
`Cc:` *old From*
`Cc:` *old Cc*

To — Like All, but omit the original `CC:` recipients. Headers are:

`To:` *old From*
`To:` *old To*

To-Cc — Like All-Cc, but but omit the original `CC:` recipients. Headers are:

`To:` *old From*
`Cc:` *old To*

Cc-To — Like Cc-All, but but omit the original `CC:` recipients. Headers are:

`To:` *old To*
`Cc:` *old From*

Sender — Reply is just for the sender of the message. Headers are:

`To:` *old From*

zwei:*1r-reply-mode* *Variable*

Profile Option: Default reply with argument of 1 to

Controls the automatic generation of `to` and `cc` fields in the header of a reply when a reply command is given an argument of 1. The default is to reply only to the sender of the message. For an explanation of the choices: See the section "Zmail Reply Command".

zwei:*middle-reply-mode* *Variable*

Profile Option: Default reply to for middle button

Controls the automatic generation of `to` and `cc` fields in the header of a reply you click middle on Reply. The default is to reply to the sender of the message. For an explanation of the choices: See the section "Zmail Reply Command".

zwei:*reply-window-mode* *Variable*

Profile Option: Default reply window setup

Two windows	(The default.) The message you are replying to is displayed in the upper window. You compose your reply in the lower window.
One window	The message you are replying to is not displayed.
Yank	The message you are replying to is included in your reply.

zwei:*middle-reply-window-mode* *Variable*

Profile Option: Default reply window setup for middle button

Controls the configuration of the windows in reply mode when you click middle on Reply. The default is two windows.

zwei:*reply-header-format* *Variable*

Profile Option: Format of headers inserted for reply

The choices are: Short (the default), Long, Include personal, and Use original.

zwei:*generate-in-reply-to-field* *Variable*

Profile Option: Automatically generate In-reply-to fields

Controls whether the headers on a reply will contain an `In-reply-to:` field, referencing the original message. If you leave it *yes* (**t**, the default) an `In-reply-to:` field is generated. If you change it to *no* (**nil**) this field is not generated.

zwei:*dont-reply-to* *Variable*

Profile Option: People not to reply to

Allows you to specify a list of addresses to avoid sending a reply to automatically. For example, if a message were broadcast to a large mailing list asking a question, you probably want to reply only to the sender, not the entire mailing list.

Zmail Options for Including Messages in a Reply

zwei:*one-window-after-yank* *Variable*

Profile Option: Just show headers and text after yanking in message

Controls the window configuration in a reply when the text of the message being replied to is included. If you leave this *yes* (**t**, the default), only one window is used after the message being replied to is yanked into the reply. If you change this to *no* (**nil**), both windows are kept, even though the text of the message being replied to is included in the reply window.

zwei:*prune-headers-after-yanking* *Variable*

Profile Option: Prune headers of yanked messages

Controls how much of the header information is kept on messages included in replies. If you leave it *yes* (**t**, the default), only the `date` and `from` lines are kept. If you change it to *no* (**nil**), the entire header of the included message is kept.

Zmail Options for Forwarding Messages

zwei:*forwarded-message-begin* *Variable*

Profile Option: Format line before forwarded messages

Allows you to specify a string to use to introduce a forwarded message.

zwei:*forwarded-message-separator* *Variable*

Profile Option: Format line between forwarded messages

Allows you to specify a string to use in between two forwarded messages.

zwei:*forwarded-message-end* *Variable*

Profile Option: Format line after forwarded messages

Allows you to specify a string to use after a message being forwarded.

zwei:*forwarded-add-subject* *Variable*

Profile Option: Forwarded messages are supplied with a subject

zwei:*reformat-forwarded-msgs* *Variable*

Profile Option: Reformat headers of forwarded messages.

Controls what happens to the headers of messages you forward. The choices are:

Yes	The default. The headers are pruned and reformatted.
No	The headers are left as in the original message. If you are forwarding a message to show someone something about its headers, you should set this option to No before forwarding the message.

zwei:*indent-forwarded-msgs* *Variable*

Profile Option: Indent text of forwarded messages.

Controls whether a message that you forward is indented 4 spaces from the left margin, like messages in replies.

The choices are:

Yes	The forwarded text is indented 4 spaces from the left margin.
No	The default. The text is flush with the left margin, as in the original.

Zmail Options for Deleting Messages and Moving Around

zwei:*delete-middle-mode* *Variable*

Profile Option: Direction to move for click middle on delete

Controls which message to select as current when you delete the current message using [Delete (M)]. The choices are: Backward (the default), Forward, No, Forward/Remove, and Backward/Remove.

zwei:*next-after-delete* *Variable*

Profile Option: Direction to move after delete

Controls which message to select as current when you delete the current message. The choices are: Backward, Forward (the default), No, Forward/Remove, and Backward/Remove.

zwei:*next-middle-mode* *Variable*

Profile Option: Middle button on Next command

Controls the action of [Next (M)]. The choices are:

Next undeleted	Selects the next undeleted message.
Next	Selects the next message in the sequence, whether or not it has been marked for deletion.
Next unseen	Selects the next unseen message.
Next recent	Selects the next message in the recent sequence.
Last undeleted	(The default.) Selects the last undeleted message in the sequence.
Last	Selects the last message in the buffer, whether or not it has been marked for deletion.
Last unseen	Selects the last unseen message in the sequence.
Last recent	Selects the last message in the recent sequence.

zwei:*previous-middle-mode* *Variable*

Profile Option: Middle button on Previous command

Controls the action of [Previous (M)]. The choices are:

Previous undeleted	Select the previous undeleted message.
Previous	Selects the previous message, whether or not it is marked for deletion.
Previous unseen	Selects the previous unseen message in the sequence.
Previous recent	Selects the previous message in the recent sequence.
First undeleted	(The default.) Selects the first undeleted message in the sequence.
First	Selects the first message in the sequence, whether or not it has been marked for deletion.
First unseen	Select the first unseen message in the sequence.
First recent	Select the first message in the recent sequence.

zwei:*map-middle-mode* *Variable*

Profile Option: Middle button on Map command

Controls the action of [Map (M)].

Delete	Deletes all messages.
Undelete	Undeletes all messages.
Type	Types out (displays) all messages in the typeout window.
Find string	Shows lines within messages containing the given string. You can select the message containing a line by clicking on the line

	of text. This provides a handy way to search through a collection for a message you only vaguely remember.
Keywords	Puts specified keywords on all messages. Clicking (L), (M), and (R) on Keywords work just as for [Keywords] in the Zmail menu. See the section "[Keywords] Zmail Menu Item".
Unkeywords	Removes specified keywords from all messages.
Move	Moves all messages to the specified file.
Hardcopy	Hardcopies all messages.
Forward	Forwards all messages (concatenated into one message). See the section "[Mail] Zmail Menu Item".
Redistribute	Redistributes all messages, individually but to the same recipient(s).
Reply	Replies to all messages (concatenated into one message).
Concatenate	Appends all messages to the first message.
Select conversation	Selects messages to which a message in the sequence refers, or that refer to a message in the sequence, recursively; this is implemented by **zwei:com-zmail-select-all-conversations-by-references**.
Undefined	(The default.)

zwei:*summary-mouse-middle-mode* *Variable*

Profile Options: Middle button on summary window

Controls the action when you click middle on a message header in the summary window. The default is Delete/Undelete which means if the message is not deleted, mark it for deletion. If it is marked for deletion, unmark it.

Zmail Option for Ordering Keywords

zwei:*keyword-alist-sort-predicate* *Variable*

Profile Option: Predicate for sorting keywords in keyword menu

Zmail Option for the Format of Mail Files

zwei:*text-mail-file-separator* *Variable*

Profile Option: Line between messages in text mail file

Allows you to specify a format control string to be used to separate messages when you hardcopy a mail file or sequence and have not specified that each message be on a separate page.

For example, to get some white space and a row of dashes, you might use something like this:

```
(format t "~2&----------------------------------------~%")
```

See the section "Formatted Output".

Zmail Options for Moving Messages and Creating Collections

zwei:*default-move-mail-file-name* *Variable*

Profile Option: Default file for moving to a new file

Allows you to specify the pathname of the file to which you usually want to move messages.

zwei:*delete-after-move-to-buffer* *Variable*

Profile Option: Delete message when moved into buffer

Controls the automatic deletion of a message from one buffer when it is moved to another buffer. If you leave it *yes* (**t**, the default) the message is deleted from its original buffer when it is moved to a new one. If you change it to *no* (**nil**) the message appears in both buffers.

zwei:*default-mail-buffer-generation-retention-count* *Variable*

Profile Option: Generation retention count set on newly created mail files

Controls the automatic deletion of copies of a new mail file. If it is left blank (**nil**) no deletion of earlier copies is done. Otherwise, the specified number are kept and others deleted. The UNIX file system does not handle this variable, so if your mail is stored on a UNIX system, leave this variable **nil**.

zwei:*query-before-selecting-empty-sequence* *Variable*

Profile Option: Confirmation is required to select an empty sequence

zwei:*preserve-msg-references-across-expunge* *Variable*

Profile Option: Add header fields to other messages when expunging message

Controls whether the backward and forward references among messages in a conversation should be preserved when a message is deleted and expunged from the middle of the conversation. If you leave it *no* (**nil**, the default), the references will not be preserved. If you change it to *yes* (**t**), appropriate header fields will be

added to the messages referred to by the deleted message or referring to the deleted message so that the conversation continues to hold together.

Zmail Options for Calendar Mode

zwei:*configure-middle-mode* *Variable*

Profile Option: Middle button on Configure

Controls the action of clicking middle on Configure. The choices are: Summary only, Both (the default), Message only, Experimental, Calendar, Month, Four weeks, Week, Year.

zwei:*calendar-mode-week-starts-on-monday* *Variable*

Profile Option: The week starts on Monday rather than Sunday in calendar mode

zwei:*delete-expired-msgs* *Variable*

Profile Option: Automatically delete expired messages

Controls whether or not you are asked before expired reminders in calendar mode are deleted. The default is per file, meaning that you can set it differently for each file.

Some of the menu items in the profile display also write information into your profile. These are:

[Mail Files] Profile Menu Item

[Mail Files]	Other Mail Files. Allows you to add files to the list of mail files to be remembered in your profile.
[Mail Files (M)]	Filter associations. Selects a mail file whose filter associations to edit.
[Mail Files (R)]	Pops up a menu of Other Mail Files and Filter associations.

File Options (Menu)

Select one of your mail files whose file options to edit.

You can set up lists of keywords and associate them with specific mail files or filters by clicking on [Keywords] in the Profile menu.

[Keywords]	Edits keyword list for all your mail files.
[Keywords (M)]	Selects a keyword whose filter associations to edit.
[Keywords (R)]	Pops up a menu of Mail Files Keywords and Filter associations.

Header Formats

There are three header formats known to Zmail: RFC733, Network, and ITS. This section describes the various header fields in each format. You can insert various header fields into the Headers window, which contains the headers for the message being written. See the section "Altering Zmail Header Fields".

A message with bad header format gets the `(badheader)` property.

`Date:` The day, date, and time the message was sent. Generated automatically when a message is sent.

`From:` The user name and host name of the sender of the message. Generated automatically. If you choose, you can explicitly provide this field, in which case a `Sender:` field is automatically generated with the user name and host name of the sender. This is useful if you send a message from a machine logged in under someone else's name; give a `From:` field with your user name in it.

`To:` The user names and possibly host names of the primary recipients of the message. Depending on the mail server, one can also include names of mailing lists (distribution lists) and file names in the `To:` lists. If a mailing list name is included, the message is sent to everyone on the mailing list; if a filename is included, the message is sent to the file.

`CC:` A list of secondary recipients of the message, in the same format as the `To:` field.

`BCC:` For "blind carbon copies". The field contains recipient names. The recipients in a `BCC:` field do not appear in the copy of the message that is delivered to the ordinary recipients; they do appear in the copy that is delivered to `BCC:` recipients.

`FCC:` For filing a copy of a message that is being sent. The recipients see the field in the message. For example,

```
FCC: F:>JHW>MAIL>OUTGOING.BABYL
```

The file has to exist already; FCC: cannot result in a file being created.

`BFCC:` For filing a "blind" copy of a message that is being sent. The recipients of the message do not see the `BFCC:` field. For example,

```
BFCC: F:>JHW>MAIL>OUTGOING.BABYL
```

The file has to exist already; `BFCC:` cannot result in a file being created.

`File-References:` One or more pathnames, separated by commas. This is useful when you want to direct someone to a file. The pathname becomes the default for the Compile File, Load File, Edit File, Show File, Format File and Hardcopy File commands.

`Included messages:` A collection of all message ID fields; added by Zmail when user concatenates messages.

`Included references:`
A collection of all "in-reply-to" fields; added by Zmail when user concatenates messages.

`Forward-References:`
Field added by Zmail when the variable **zwei:*preserve-msg-references-across-expunge*** is set, so that conversations remain intact even when some messages are expunged. See also `Backward-References:`.

`Backward-References:`
Field added by Zmail when the variable **zwei:*preserve-msg-references-across-expunge*** is set, so that conversations remain intact even when some messages are expunged. See also `Forward-References:`.

`Encrypted:` For flagging the message as containing encrypted text. Zmail generates this header field itself when it is sending a message. The value of the field is the name of the kind of encryption that was used.

`Subject:` A line of text giving the subject of the message.

`In-reply-to:` An identification of the message being replied to. The message is typically identified by giving the `Message-ID:`, or, in its absence, the contents of its `Date:` and `From:`fields, but different mail systems form this field in different ways.

`Sender:` The user name and host name under which the message was sent, when different from the `From:` field; automatically inserted if a `From:` field is given (see above).

`Redistributed-to:` The recipients of the redistributed message; a list in the same format as the `To:` field. `Resent-to:` is a synonym.

`Redistributed-by:` The name of the user who redistributed the message. `Resent-by:` is a synonym.

`Redistributed-date:`
The date the message was redistributed. `Resent-date:` is a synonym.

`Expiration-date:` A date, intended as the date on which some mail systems will automatically delete the message.

`Reply-to:` An address, in the same format as the `To:` field. Intended as the address to which to send replies to this message, when that is different from the `From:` or `Sender:`.

`Message-ID:` A unique character string that distinguishes this message from all others.

`Supersedes:` If you retransmit a message, Zmail gives the message a new message-id and offers to add a `Supersedes:` field referring to the original message. You can control this behavior in your Zmail Profile by using the **zwei:*add-supersedes-and-comments-when-retransmitting*** Zmail Profile Option.

`Comments:` A multi-line field whose value is read from the minibuffer. The profile option **zwei:*add-supersedes-and-comments-when-retransmitting*** controls whether or not Zmail asks for permission to read the `Comments:` in the minibuffer. If **zwei:*add-supersedes-and-comments-when-retransmitting*** is Yes, Zmail goes directly to reading `Comments:` in the minibuffer after you supersede a message. See the section "Zmail Header Formats".

System Dependencies

Disk File Names

The files discussed in this document (mail files, default mail files, inboxes, renamed inboxes, and source and compiled init files) have distinctive file names that vary depending upon the host system you use. The following table gives the names, assuming your user id (login name) is *user-id*. Except as indicated, all files are in your standard login directory (homedir). ("LMFS" means Genera's own file system.) Names in the column *Other mail files* are conventional but not required.

System	*Default mail file*	*Other mail files*
LMFS	babyl.text	*.babyl *or* *.xmail
UNIX	mbox *or user-id*.bb	*
TENEX/TOPS-20	*user-id*.BABYL	*.BABYL *or* *.XMAIL
ITS	*user-id* BABYL *or user-id* RMAIL	* BABYL *or* * XMAIL

System	*Inbox*	*Renamed inbox*
LMFS	mail.text	mail.-zmail-text
UNIX	/usr/spool/mail/*user-id* *or* ~/.mail	/usr/spool/mail/*user-id*.zmail ~/.mail/.zmail
TENEX	MESSAGE.TXT;1	MESSAGE.-ZMAIL-TXT
TOPS-20	MAIL.TXT.1	MAIL.-ZMAIL-TXT
ITS	*user-id* MAIL	*user-id* _ZMAIL

System	*Source files*	*Compiled file*
LMFS	zmail-init.lisp	zmail-init.bin
UNIX 4.1	zmail-init.l	zmail-init.bn
UNIX 4.2	zmail-init.lisp	zmail-init.bin
TENEX/TOPS-20	zmail-init.lisp	zmail-init.bin
ITS	ZMAIL >	*user-id* ZMAIL

If your init file is not compiled, or if you delete your compiled file, rename the source file to the name in the column *Compiled file*.

Mail File Formats

Zmail understands five mail file formats: BABYL, RMAIL, KBIN, TENEX, and UNIX. In most cases, the format is transparent to the user. However, the following information is useful if you transfer files between systems.

Zmail recognizes the format of a mail file from its contents, never from its file name, but with the following limitation: certain formats are only recognized on certain systems:

Mail file format	*System(s)*
BABYL	All
KBIN	All
UNIX	UNIX
RMAIL	LMFS, ITS, UNIX
TENEX	TENEX/TOPS-20
VAX/VMS	Not supported at present

To select a mail file whose format does not satisfy these expectations, use Select Arbitrary Format Mail File, which allows you to specify the format explicitly. See the section "Select Arbitrary Format Mail File (m-X) Zmail Command". (Inbox files have a different format on each system, and can only be read on the type of system on which they were written.)

For UNIX, the default file format is RMAIL, so the mail file can be read with either Zmail or GNU Emacs. If you want your default mail file to be a BABYL file (which cannot be processed using the UNIX mail reading program, but which is more useful when using Zmail), the file *user-id*.bb must be created in your home directory and the Mail option in that file set to:

Mail:*homedir*/mbox/usr/spool/mail/*user-id*

Binary Format for Storing Mail Files

KBIN format stores messages as binary data rather than text. In addition to the actual message text, KBIN files contain the parsed representation of the message. As a result, KBIN files are usually between 30% and 50% larger than BABYL or RMAIL files. This means that it normally takes between 30% and 50% longer to save a KBIN file.

However, once a KBIN file is read into your machine, all the information needed by Zmail to process its contents is already present. Zmail does not have to reparse the messages which is where most of the time is actually spent while loading mail files. Thus, KBIN files show a marked improvement in loading times.

The binary format for KBIN files sometimes changes when new versions of Zmail are compiled. When an old format KBIN file is read into a new version of Zmail, a message warning you that the file was written with an older version of Zmail is printed. The file's format is updated automatically when you write it out again. Files written with a particular version of Zmail cannot be read in older versions.

Converting Existing Mail Files to KBIN Format

Existing mail files can be converted to use KBIN format by the following procedure:

Enter Zmail and click on [Profile] in the command menu.

1. Click on [File Options] in the profile frame and select a mail file that you want to convert to KBIN format.

2. If the mail file's pathname extension reflects its format, Zmail automatically renames it to KBIN. For example, if your mail file is named `KJones.babyl`, Zmail renames it to `KJones.kbin`. No attempt is made to automatically rename files named `babyl.text`. If your mail file is named `babyl.text`, you should rename it yourself to `mail.kbin`. Do this by clicking on the pathname.

3. Click on the KBIN format.

4. Click on [Do It]. If you have renamed the mail file, Zmail automatically updates any references to it in your profile and reminds you to click on [Save] to permanently record these changes in your profile. Zmail then announces that it is converting the mail file's format and asks you to stand by. If your mail file is large, the conversion might take some time.

Repeat these steps for each mail file that you wish to convert to KBIN format.

Then, to make the conversion permanent:

1. Click on [Save] in the Profile Menu. Answer yes to the queries to insert changes and, optionally, recompile your profile.

2. Click on [Exit] to leave the profile editor.

3. Click on [Save] in the Zmail menu or press `S` to actually write out the converted mail files.

Support for Internet domain addressing

Zmail supports the Internet RFC822 domain-addressing formats, for the purpose of parsing and replying to messages with domain-format addresses in their headers. If the machine name is registered in the ARPA network host table, that name is used in the address. If the machine name is not registered, the network address is used, in the form [*address*], where *address* is the four integer numbers which specify the host in Internet addressing.

Features Not Supported by Zmail

Some mailers and file formats do not support all of the features described in this document.

UNIX and RMAIL format do not support keywords, properties, or file attributes. TENEX format does not support keywords or file attributes. Mail buffers for files in these formats can use these features, but the information is not saved in the disk copy.

VAX/VMS mail is presently unsupported; that is, VMS format inboxes and mail files cannot be read or written. However, mail files in any of the four standard formats can be stored on VMS.

Dictionary of Zmail Commands

dbg:*character-style-for-bug-mail-prologue* *Variable*

Creates the bug-report banner inserted into the text of bug messages, enabling you to choose the font. The default is NIL.NIL.TINY, specifying a small font for the bug-report banner.

To display a bug-report banner in a small font you can type the following:

```
(setq dbg:*character-style-for-bug-mail-prologue*
      (si:character-style-for-device-font 'fonts:quantum si:*b&w-screen*))
```

To display a bug-report banner in a large font you can type the following:

```
(setq dbg:*character-style-for-bug-mail-prologue*
      (si:parse-character-style '(nil nil :huge)))
```

You can also type the following to specify a particular font:

```
(setq dbg:*character-style-for-bug-mail-prologue* '(nil nil :huge))
```

. (Kbd) Zmail Command

. (period) Scrolls back to the beginning of the current message.

c-X 0 (Kbd) Zmail Command

c-X 0 (Kbd) Zero window mode. The Message window on the *top-level display* is used for the message being composed. When the c-X 0 command is issued, the screen is restored to its format at top level, except the Message window displays the headers and text of the message being written. (See Figure !.)

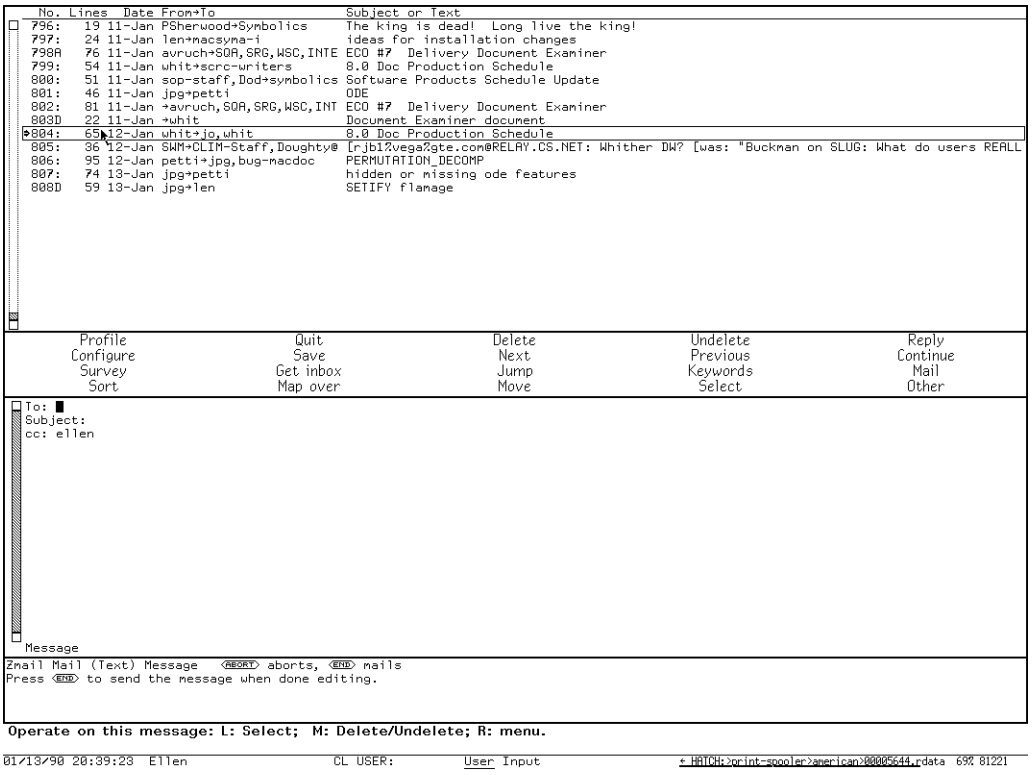


Figure 113. Mail Mode Display (Zero-Window Mode)

c-X 1 (Kbd) Zmail Command

c-X 1 (Kbd) One-window mode. The mail mode display is configured with two windows, Headers and Mail, used for the headers and text of the message being written.

c-X 2 (Kbd) Zmail Command

c-X 2 (Kbd) Two-window mode. The mail mode display is configured with three windows, Message, Headers, and Mail, which display the current message (which is the message being replied to, if using the Reply command), and the headers and text of the message being written.

Add Cc Field Zmail Command

[Add Cc Field] (Editor Menu)

n c-X C (Kbd) Adds another CC: recipient. Positions cursor at the end of the CC: field, set up to add another name. (Creates a CC: field if there is not one already.) With an argument *n* = 0, positions cursor at beginning of CC: field. With a negative argument, deletes the CC: field.

Add Fcc Field Zmail command

[Add Fcc Field] (Editor Menu)

n Add Fcc Field (m-X)
Adds another FCC: recipient. Positions cursor at the end of the FCC: field, set up to add another name. (Creates an FCC: field if there is not one already.) With an argument *n* = 0, positions cursor at beginning of FCC: field. With a negative argument, deletes the FCC: field.

Add From Field Zmail command

[Add From Field] (Editor Menu)

n Add From Field (m-X)
Creates or replaces From: field. Creates a From: field and positions cursor for entry of user name. If a From: field exists already, it is deleted and replaced. With an argument *n* = 0, po-

sitions cursor at beginning of From: field. With a negative argument, deletes the From: field.

Add File-References Zmail Command

[Add File Reference Field] (Editor Menu)

Add File References (m-X)

Creates a File-References: field and positions the cursor for entry of a pathname or pathnames separated by commas. Pathnames in the file-references field can be operated on directly from Zmail by the Zmail file manipulation commands:

Compile File (m-X)
Edit File (m-X)
Format File (m-X)
Hardcopy File (m-X)
Load File (m-X)
Show File (m-X)

Add In-reply-to Field Zmail command

[Add In Reply To Field] (Editor Menu)

Add In Reply To Field (m-X)

Creates In-reply-to: field. Creates an In-reply-to: field specifying the message being replied to. This command can be used only if mail mode was entered using one of the Reply commands. You can control the generation of [In-reply-to:] fields in your Profile. See the variable **zwei:*generate-in-reply-to-field***.

Add Message References Zmail Command

Add Message References (m-X)

Add references to the current message. Prompts for a reference or references. Terminate by pressing END. Normally, typing a message reference is too cumbersome, however, so you can click Left on a message in the summary area to use its Message-ID immediately, or you can click Middle on a message in the summary area to have its Message-ID inserted into the minibuffer. Using Mouse Middle is especially useful if you mean to add several message references at once because you do not have to reissue this command several times — you can just accumulate all of the references in the minibuffer at once and then finally press END.

Add More Text Zmail Command

Add More Text (Editor Menu)

`c-X A` (Kbd) Select the Mail window.

Add References Field Zmail Command

Add References Field (Editor Menu)

Add References (`m-X`)
Adds the message-id of each message in the current sequence to the message being composed.

Add Subject Field Zmail command

[Add Subject Field] (Editor Menu)

n `c-X S` (Kbd) Creates or replaces `Subject:` field. Creates a `Subject:` field and positions cursor for entry of text. If a `Subject:` field exists already, delete and replace it. With an argument *n* = 0, positions cursor at beginning of `Subject:` field. With a negative argument, deletes the `Subject:` field.

Add To Field Zmail command

[Add To Field] (Editor Menu)

n `c-X T` (Kbd) Adds another `To:` recipient. Positions cursor at the end of the `To:` field, set up to add another name. (Creates a `To:` field if there is not one already.) With an argument *n* = 0, positions cursor at beginning of `To:` field. With a negative argument, deletes the `To:` field.

Append Conversation By References (`m-X`) Zmail Command

Append Conversation by References (`m-X`)
Append messages to which this message refers, or which refer to this message, recursively.

Append To Referenced Message (`m-X`) Zmail Command

Append To Referenced Message (m-X)
: Appends this message to the referenced message.

Apropos (m-X) Zmail Command

Apropos (m-X)
: Prompts you for a character string and returns a list of the m-X commands containing that string in their name or the first line of their help documentation. You can use Apropos (m-X) in mail or edit mode also.

Bug (m-X) Zmail Command

Bug (m-X)
: Send a bug report. Prompts for the name of a bug list to send to, then puts you into mail mode with the To: field set to that name. The mail window is selected and the information from your herald identifing what version of the software you are using is inserted at the beginning of the message. You can now type in your bug report and send the message.

  You can control the character style of the herald information by setting the value of **dbg:*character-style-for-bug-mail-prologue***. See the variable **dbg:*character-style-for-bug-mail-prologue***.

C (Kbd) Zmail Command

C (Kbd)
: Continue the most recently aborted message.

Change Mail File Options (m-X) Zmail Command

Brings up a menu of the options for the current Zmail mail file. This is the same menu that you get when you click on [File Options] in the Zmail profile. See the section "Zmail Profile Options".

Change Subject Pronouns Zmail Command

[Change Subject Pronouns] (Editor Menu)

Change Subject Pronouns(m-X)

Fix up the pronouns in the Subject: field of a reply. "I" is replaced by "you," "you" by "I," "mine" by "yours," "yours" by "mine."

Check for New Mail (m-X) Zmail Command

Check for New Mail (m-X)
: Checks in the foreground for new mail in the inbox(es) associated with the default buffer. This is similar to what the background process does periodically. In addition to printing a message, this command prevents the background process from telling you about the same new mail. Thus the command can also be used as a way of "noticing" new mail that you might have read in the editor or with Show Mail (m-X). The next "you have new mail" message from the background refers to really new mail.

Click Middle on Summary Line Zmail Command

Click Middle on Summary Line (Summary Window)
: Toggle the deleted property of the message whose summary line was clicked on. That is, delete a nondeleted message, undelete a deleted message. Do not select the message as current message. If the current message is being deleted, move to the next undeleted message. See the section "Zmail Message Deletion Commands".

Compile File (m-X) Zmail Command

Compile File (m-X)
: Prompts for a pathname and compiles the file specified by the pathname. The default is the first pathname specified in the File-References: header field.

[Continue] Zmail Menu Item

[Continue]
: Continue the most recently edited message, whether sent or aborted. See the section "Continuing Completed or Aborted Zmail Messages".

[Continue (M)]
: Continue the most recently aborted message.

[Continue (R)]
: Pop up a menu offering Sent Drafts, Unsent Drafts, or All Drafts. Clicking on one of these offers a list of the messages you have composed in your current Zmail session. The mes-

sages are identified by their headers; see Figure !.) Click on a message to continue it. Two other items on the menu are [Restore draft file] and [Restore draft message]. See the section "Saving and Restoring Zmail Message Drafts".

[Restore Draft Message]
Enters mail mode with the Headers and Mail windows restored from the current message, if it is a draft message. If it is not, flashes the screen and ignores the Continue command.

[Restore Draft Message (R)]
Waits for you to click on a draft message in the summary window or type a message number in the mini-buffer, then enters mail mode with the Headers and Mail windows restored from that message. (If the selected message is not a draft message, Zmail flashes the screen and ignores the Continue command.)

[Restore Draft File]
Prompts for a filename of a saved draft and enters mail mode with the Headers and Message windows restored from the file.

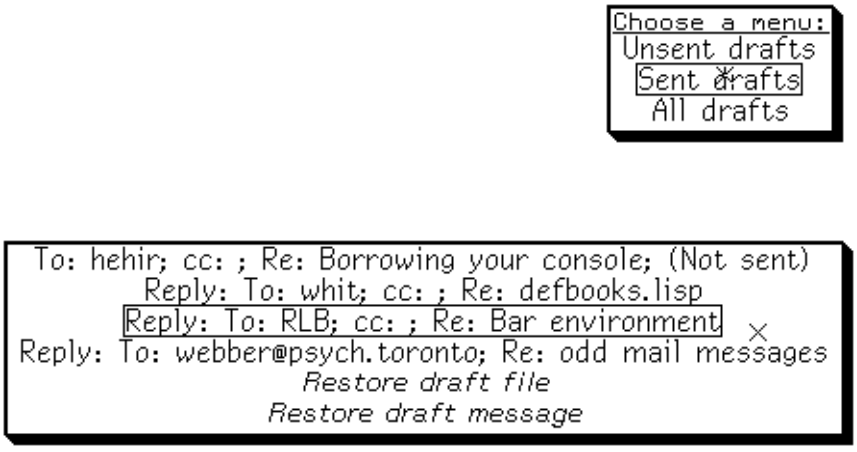


Figure 114. [Continue (R)]

`D` (Kbd) Zmail Command

n `D` (Kbd)
Deletes message *n*. If *n* is negative or if it is greater than the number of messages in file, Zmail complains "`Argument out of range`". If *n* is omitted, `D` deletes the current message.

c-X commands available in Zmail

Zmail now has the appropriate presentation types to allow it to prompt for message sequences in a fashion compatible with prompting for buffer names in Zmacs. That is, you can just type the filename instead of the complete pathname. In addition, in most cases, when you are asked for a mail file buffer, you can also enter the name of any mail file mentioned in your Zmail profile that is not loaded yet.

Try pressing HELP when Zmail prompts you for sequences to see what is acceptable.

Thanks to the above change, Zmail now has c-X commands:

c-X B or [Select] (**zwei:com-zmail-select-sequence**)
: Selects a message sequence. If you invoke it from the keyboard, it prompts for a sequence name in the minibuffer. Clicking Left selects the previously selected sequence. Clicking Middle creates a new collection by filtering. Clicking Right displays a menu of existing sequences, unloaded mail files, and special actions such as Read/Create file, Mark Survey, and so on.

c-X K (**zwei:com-zmail-kill-sequence**)
: Kills a message sequence. It prompts for a sequence name in the minibuffer. If you ask to kill the current sequence, this command asks for the name of another sequence select as the current sequence. If you ask to kill a mail file buffer, you are asked either to save it first or to confirm that you picked the proper sequence.

c-X c-B (**zwei:com-zmail-list-sequences**)
: Lists the current message sequences.

c-X c-F (**zwei:com-zmail-edit-mail-file**)
: Edits a mail file. It prompts for the pathname of a mail file in the minibuffer and reads that mail file into a Zmail mail file buffer. If the mail file does not exist, an empty mail file buffer is created which, when saved, creates the mail file with the requested pathname.

c-X c-R (**zwei:com-zmail-examine-mail-file**)
: Examines an existing mail file. It prompts for the pathname of a mail file in the minibuffer, reads that mail file into a Zmail mail file buffer, and then disables saving of the buffer. The mail file must already exist.

c-X c-S (**zwei:com-zmail-save-mail-file**)
: Saves the current mail file. Without a numeric argument, it expunges deleted messages from the current mail file buffer and writes the updated buffer to the mail file. With a numeric argument, it starts saving the current mail file in a background process. Deleted messages are not expunged. The Abort Background Save command can be used to stop the background save.

c-X c-sh-F (**zwei:com-zmail-edit-file**)
: Edits a file. Prompts in the mini-buffer for a pathname (the default is the first pathname if the File References Field, if any) and selects Zmacs to edit the file.

In addition to the above, c-m-L now invokes **zwei:com-zmail-select-previous-sequence**. When you give c-m-L a zero numeric argument, it prompts for a sequence name in the minibuffer after listing the possible sequences. The entries in the listing are, of course, mouse sensitive.

c-D (Kbd) Zmail Command

c-D (Kbd)
: Delete current message and move to previous undeleted message, like clicking [Delete (M)].

Decode ECO (m-X) Zmail Command

Decode ECO (m-X)

Turns an ascii encoded ECO message back into a temporary file and then offers to load that file. You get an error if the file is not a distribution file.

Decrypt Text (m-X) Zmail Command

Decrypt Message (m-X)
: Prompts for the encryption key and then displays an encrypted message as plain text. By this operation, you are only viewing the plain text form; use a numeric argument to store the plain text version in the mail file.

Delete Duplicate Messages (m-X) Zmail Command

Delete Duplicate Messages (m-X)
: Delete duplicated messages from the mail file, retaining only the first copy of a duplicated message. Two messages are duplicates if and only if they have the same From:, Date:, To: (if any), Cc: (if any), and Subject: (if any) fields. (The other headers and the text of the message are not checked.) Duplicate messages can arise from merging two mail files, for example. See the section "Zmail Message Deletion Commands".

  You can automatically delete duplicate messages from your new mail by adding the following form to your Zmail init file after the automatically generated forms:

```
(login-setq *insert-inbox-hooks* '(:delete-duplicates-new))
```

Then each time your inbox is read, those new messages are searched for duplicates and the duplicates eliminated. Note: since this only searches the new messages, if a message already exists in your mail file and a new copy arrives, this duplication will not be detected.

[Delete] Zmail Menu Item

[Delete] Delete current message and move to next undeleted message. (Do not move if this is last message.) You can select the direction for the move after deleting the current message in your profile. See the variable **zwei:*next-after-delete***.

[Delete (M)] Delete current message and move to previous undeleted message. (Do not move if this is first message.) You can select the direction for the move after delete in your profile. See the variable **zwei:*delete-middle-mode***.

[Delete (R)] Pop up a menu of:

Backward	Delete current message and move to previous undeleted message.
Forward	Delete current message and move to next undeleted message.
Remove	Remove message from this temporary mail file.
No	Delete current message and do not move.

Delete Conversation By References (m-X) Zmail Command

Delete Conversation By References (m-X)
Deletes all the messages in a conversation.

Delete Referenced Messages (m-X) Zmail Command

Delete Referenced Messages (m-X)
Deletes the referenced messages.

Describe Command (m-X) Zmail Command

Describe Command (m-X)
Prompts for the name of a m-X command and displays its help documentation.

Disable Saves for Buffer (m-X) Zmail Command

Disables saves for a buffer.

E (Kbd) Zmail Menu Item

E (Kbd) — Expunges the current sequence, that is removes all the messages marked for deletion.

Edit File (m-X) Zmail Command

Edit File (m-X) — Prompts for a pathname and creates an editor buffer with the specified file in it for editing. The default is the first pathname specified in the File-References: header field.

This command uses the current message's first file reference as the default when asking for the file to be loaded into Zmail.

Encrypt Text (m-X) Zmail Command

Encrypt Text (m-X) — Encrypts a message. Use it after you have completed the message draft but before you send it. Zmail prompts for an encryption key that the recipient must provide in order to decrypt the message. This key can consist of plain alphanumeric text only. Punctuation or other funny characters are ignored. Upper and Lower case are equivalent. It converts the draft to a form that you cannot read. Decrypt Text is also available for message drafts. Both of these commands appear on the draft editor menu.

Enable Saves for Buffer (m-X) Zmail Command

Enables saves for a buffer.

END (Kbd) Zmail Command

END (Kbd) — Add more text or send the message. If typed while in the Message or Headers window, selects the Mail window to allow you to add more text. If the Mail window is already selected, pressing END sends the message. See the section "Leaving Mail

Mode in Zmail". (If typed while in the Message window in zero window mode, sends the message.)

F (Kbd) Command

F (Kbd) — Forwards the current message (using **zwei:com-zmail-forward**). See the description of the Forward option of [Mail (R)].

c-F (Kbd) Zmail Command

c-F (Kbd) — Prompts for a string and selects the next message containing that string (using **zwei:com-zmail-find-string**).

Find String (m-X) Zmail Command

Find String (m-X) — Prompts for a string in the minibuffer and finds the next message containing that string (in text or header) and selects it. If it cannot find a message containing the given string, it flashes the screen.

Format File Zmail Command

Format File (m-X)

Formats the file associated with the pathname you specify. c-U m-X Format File formats the file and sends it to a printer. The default pathname is the first pathname specified in the File-References: header field. If no File-References: field exists, the default is the current mail file.

Forward (m-X) Command

Forward (m-X) — Send a message with current message as its text. Puts you in mail mode with headers window selected. Cursor is prompting you to specify the To: field. The Subject: field is initialized as "[PJF: Forwarded]" (if the original message was from PJF). The mail window contains the headers and text of the current message, followed by a (nonblinking) cursor. Supply the To: field, edit or add headers and text as you wish, and send the message. Forwarding differs from Redistributing or Redirecting in that a new message (with its own unique message id) is created.

(Note: The forwarded message (that is, the current message) is given the (forwarded) property.)

G (Kbd) Zmail Command

G (Kbd) — Reads in your new mail. If your old mail is currently read into Zmail, your new messages are appended or prepended to that buffer. If you are just starting a Zmail session, your new mail is read in and then your old mail is appended or prepended to it. Appending or prepending are controlled by your Zmail profile. See the variable **zwei:*new-mail-file-append-p***.

If your current buffer is not your primary mail file and the buffer has no associated inbox, Zmail prompts for an inbox to read for the current buffer. Inboxes can be associated with mail files other than your primary file by using the [File Options] Profile Menu item.

[Get Inbox] Zmail Menu Item

[Get Inbox] — Reads in your new mail. If your old mail is currently read into Zmail, your new messages are appended or prepended to that buffer. If you are just starting a Zmail session, your new mail is read in and then your old mail is appended or prepended to it. Appending or prepending are controlled by your Zmail profile. See the variable **zwei:*new-mail-file-append-p***. You can have Zmail delete duplicate messages from your new mail when your inbox is read. See the section "Delete Duplicate Messages (m-X) Zmail Command".

If your current buffer is not your primary mail file and the buffer has no associated inbox, Zmail prompts for an inbox to read for the current buffer. Inboxes can be associated with mail files other than your primary file by using the [File Options] Profile Menu item.

[Get Inbox (M)] — Prompts for an inbox name to read into the current buffer. Use this command to recover from file computer crashes that write your inbox in a nonstandard place.

Caution: if you specify a file that is not in the proper format — for example, if you type the name of your primary mail file — you are in trouble. Zmail becomes caught in an error loop, and has to be reloaded (or the machine cold booted).

(Remember, the file you specify is the *inbox* — the file where new mail lives — not the *mail file*, which is where old mail resides.)

[Get Inbox (R)] Pops up a menu of your mail files. You specify the file for which to read the inbox. That buffer is selected and its inbox is read.

H (Kbd) Zmail Command

H (Kbd) Scrolls back to the top ("Head") of the current message.

Hardcopy All Command

Hardcopy All (m-X) Hardcopies all the messages in the current sequence. Note that you can also click Right on [Map Over] and select [Hardcopy] for copying all messages in the current sequence.

Hardcopy File Zmail Command

Hardcopy File (m-X)

Sends the file associated with the pathname you specify to the default printing device. The default is the first pathname specified in the File-References: header field. If there is no File-References: field, the default is the current mail file.

Hardcopy Message (m-X) Zmail Command

Hardcopy Message (m-X) Hardcopies the current message.

Insert File (m-X) Zmail Command

Insert File (m-X) Prompts for a pathname and inserts the contents of the file in the mail buffer.

J (Kbd) Zmail Command

n J (Kbd) Jumps to message number *n*, even if it is marked for deletion. If *n* is omitted, jumps to the first message in the sequence. Z J jumps to the last message in the sequence.

[Jump] Zmail Menu Item

[Jump] Jumps to the message selected based on the filter of the last jump command. There is no initial default, so the first time you want to use jump in a new Zmail session you must click right for the filter menu.

[Jump (M)] Selects an arbitrary message from the message stack. Does nothing if the stack is empty. See the section "Moving Among Zmail Messages Using the Message Stack". Otherwise, displays summary lines for the elements of the message stack, partially overlaying the summary window. (The message numbers displayed are meaningless, but if the current message is on the stack, it is indicated by the usual arrow.) The summary lines are mouse sensitive; clicking Left on a line selects the corresponding message. Pressing ABORT aborts the Jump command; typing or clicking on any other command aborts Jump and executes that command. (See Figure !.)

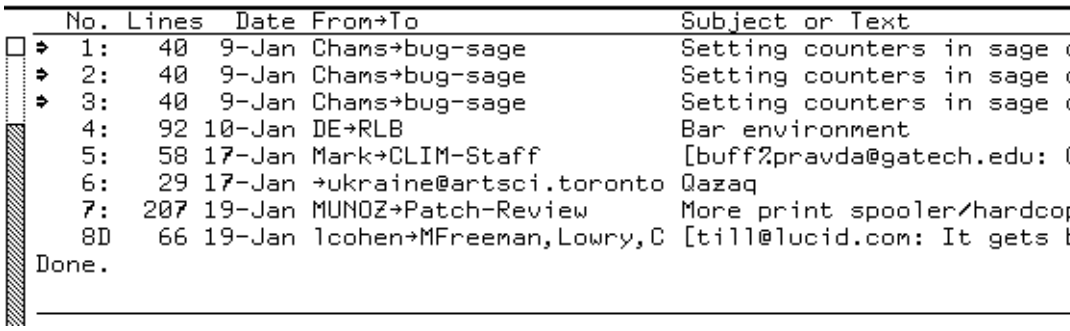


Figure 115. [Jump (M)]

[Jump (R)] Pops up the filter selection menu. You select a filter to use to choose the message to which to jump.

[Keywords] Zmail Menu Item

[Keywords] Adds the last used keywords to the current message. There is no initial default so the first time you want to add keywords to a message in a Zmail session you must click right for the menu. See the section "Setting Zmail Keywords".

[Keywords (M)] Adds the appropriate keywords to the current message automatically. "Appropriate" is determined using a *filter-keyword alist*, similar to the alist used by the referenced-message commands. Each filter is associated with a list of keywords; click-

ing middle on [Keywords] adds the keywords corresponding to all filters which the message satisfies. The mouse documentation line tells you which keywords are to be added, so you can check first. If none are to be added, the mouse documentation line shows nothing for [Keywords (M)].

[Keywords (R)] Pops up a highlighted menu of your keywords, in addition to the entry [New] for adding a new keyword. If you have never specified keywords for any messages, the menu contains only three items: [Do It], [Abort], and [New]. Click on [New] and type a keyword. The keyword appears on the menu, highlighted. Click on [Do It] and the keyword appears in braces on the summary line of the message. Keywords are stored in the mail files of the messages they are attached to. You can specify keyword/mail file associations explicitly in your Profile. See the section "Zmail Profile Options".

Figure 116. Keywords Menu

L (Kbd) Zmail Command

L (Kbd) "Labels" the current message. Prompts in the minibuffer for keywords for the current message (using **zwei:com-zmail-keywords**). You can use the mouse to click on a message in the summary window to add that message's keywords to the current message.

A history of keywords is maintained, so you can use c-m-Y and m-Y to yank back keywords previously typed in that minibuffer.

List Sequences (m-X) Zmail Command

List Sequences m-X Lists all the sequences in your current Zmail session, as well as any mail files saved in your profile that have not yet been read in.

Load File (m-X) Zmail Command

Load File (m-X) — Prompts for a pathname and loads the specified file into the Lisp environment. The default is the first pathname specified in the File-References: header field.

M (Kbd) Command

M (Kbd) — Send a message. Puts you into mail mode, with the headers window selected. See the section "Sending Your Mail". See the section "Mail Mode in Zmail".

c-X M (Kbd) Zmail Command

c-X M (Kbd) — Enters mail mode recursively; the window configuration remains the same, but the Headers and Mail windows are reinitialized as if the Mail command had just been executed (Headers window contains the word "To:" followed by a blinking cursor; Mail window is empty.) Exiting recursive mail (either by sending the message or by aborting) returns to the higher level mail.

Mail Menu Item

[Mail] — Send a message. Puts you into mail mode, with the headers window selected. See the section "Sending Your Mail". See the section "Mail Mode in Zmail".

[Mail (M)] — Send a bug report. Pop up a menu of program names and [Other]. Clicking on a program name puts you into mail mode, with the To: field set up to send a bug report about that program. Clicking on [Other] prompts for the name of a bug list to send to, then puts you into mail mode. In either case, the mail window is selected; the first several lines of text identifies what version of the software you are using. You can now type in your bug report and send the message. See the section "Adding Bug Lists to Zmail".

[Mail (R)] — Pop up a menu of:

- Bug — Send a bug report. Use [Bug] to send report to same bug list as last report; use [Bug (R)] for menu of programs. See description of [Mail (M)]. See the section "Adding Bug Lists to Zmail".
- Mail — Send an ordinary message, like clicking left on [Mail].

Forward — Send a message with current message as its text. You are placed in mail mode with the headers window selected, and the cursor positioned after `to:`. The text of the current message is placed in the mail window inside the delimiters for forwarded messages specified in your profile. See the variable **zwei:*forwarded-message-begin***. You enter the recipient(s) and then press `END`, which leaves you in the mail window so you can add comments to the text of the message if you so desire. Pressing `END` again sends the forwarded message.

Redistribute — Redistribute the current message to other recipients. You are prompted for a new recipient or recipients (separated by commas) to whom to send the message.

Local — Create a new message in the current mail file. A new message draft is created with an `Fcc:` destination of the current inbox. (See Figure !.)

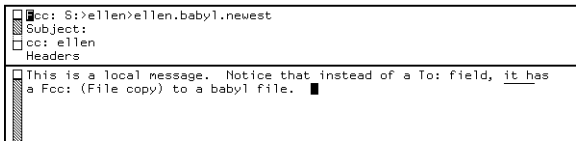


Figure 117. Local Mail

You are placed in mail mode with the headers window selected, and the cursor positioned after "`Subject:`". Type the subject of the message, press `END`, then the text.

Local messages never actually get sent as messages; they are just added to your mail file. They are useful for making notes to yourself.

Mail ECO (m-X) Zmail Command

Mail ECO (`m-X`)

Sends an ECO as a mail message. It prompts for a file containing an encoded ECO. If the file is not yet encoded for mailing, Mail ECO offers to encode it (prompting for a pathname to use to hold the encoded ECO). The encoded ECO is then placed in a message buffer. You can add any comments and terminate the message with `END` as for any other mail message.

Make Encoded ECO File (m-X) Zmail Command

Make Encoded ECO File (m-X)

Encodes a file for distributing as an ECO. It prompts for a source file and an output pathname to hold the encoded result.

[Map Over] Zmail Menu Item

Performs an operation on all messages in the current sequence.

[Map Over] Performs the last map over operation. There is no initial default so in a new Zmail session you must first click right for the menu.

[Map Over (M)] You can specify an operation from the menu provided by [Map Over (R)] to put on this key in your profile. See the variable **zwei:*map-middle-mode***.

[Map Over (R)] Pops up a menu of operations to perform on all messages in the current sequence:

Delete — Deletes all messages.

Undelete — Undeletes all messages.

Type — Types out (displays) all messages in the typeout window.

Find string — Prompts for a string and shows those lines within messages that contain the given string. You can select the message containing the string by clicking on the line of text. This provides a handy way to search through a collection for a message you only vaguely remember. See the section "Occur (m-X) Zmail Command".

Keywords — Puts specified keywords on all messages. Clicking (L), (M), and (R) on Keywords work just as for [Keywords] in the Zmail menu. See the section "[Keywords] Zmail Menu Item".

Unkeywords — Removes specified keywords from all messages.

Move — Moves all messages to the specified file.

Hardcopy — Hardcopies all messages.

Forward — Forwards all messages (concatenated into one message). See the section "[Mail] Zmail Menu Item".

Redistribute — Redistributes all messages, individually but to the same recipient(s).

Reply — Replies to all messages (contatenated into one message).

Concatenate — Appends all messages to the first message.

Select conversation — Selects messages to which a message in the sequence refers, or

that refer to a message in the sequence, recursively; this is implemented by **zwei:com-zmail-select-all-conversations-by-references**.

[Move] Zmail Menu Item

[Move] Moves the current message to the same buffer as last [Move]. There is no initial default so the first time in a Zmail session that you want to move a message, you must click right on [Move] for the menu. Whether or not the moved message is deleted from its original buffer is determined by your profile (the default is delete). See the variable **zwei:*delete-after-move-to-buffer***.

[Move (M)] Moves the current message to a buffer based on the filter it satisfies. If the current message does not satisfy any of your predefined filters, nothing is offered in the mouse documentation line for the middle click.

[Move (R)] Moves the current message to the buffer specified from a menu of all the mail sequences in your current Zmail, any mail files stored in your profile but not yet read into Zmail (see the section "Saving a List of Mail Files"), and five other options:

- New Collection — Starts a new mail collection, prompting for a name for the collection, and copies the current message to that collection.
- Recycled Collection — Starts a temporary mail collection and copies the current message to that collection. See the section "Creating a Mail Collection Starting with a Single Message".
- Read/Create File — Prompts for a mail file, creating the file if it does not exist, and moves the current message to that file.
- Just Text — Prompts for a file name and moves the current message to that file as simple text for editing with Zmacs.
- Hardcopy — Hardcopies the current message.

Move In Place Of Referenced Message (m-X) Zmail Command

Move In Place Of Referenced Message (m-X)
Moves this message to where the referenced message is, and deletes the referenced message.

Move to Default Previous Point (m-X) Zmail Command

n Move to Default Previous Point (m-X)

With an argument *n*, performs the same rotation as *n* c-m-SPACE and makes *n* the new default argument. Without an argument, uses the default. (The initial default is 3.)

Move to Point (m-X) Zmail Command

n Move to Point (m-X)

n c-m-SPACE (Kbd) Without an argument, exchanges the current message and the top of the message stack. (The top of the stack is popped into the current message and the old setting of the current message is pushed onto the stack.) With an argument $n > 1$, rotates the top *n* entries of the list formed from the current message followed by the message stack. ($n = 2$ is equivalent to no argument.) With an argument of 1, rotates the whole list. Negative arguments rotate the other way.

An example: if *n* is 3, Stack[1] is the top of stack, and Stack[2] is the element just below the top of stack, then:

Old Stack	*New Stack*
new current message	old Stack[1]
new Stack[1]	old Stack[2]
new Stack[2]	old current message

N (Kbd) Zmail Command

n N (Kbd) Selects the *n*th next undeleted message in the current sequence. If *n* is omitted, selects the next undeleted message.

c-N (Kbd) Zmail Command

n c-N (Kbd) Selects the *n*th next message in the sequence, whether or not it is marked for deletion. If *n* is omitted, selects the next message, whether or not it is deleted.

[Next] Zmail Menu Item

[Next] Select the next undeleted message in the current sequence.

[Next (M)] Selects the last undeleted message in the current sequence. You can set the action for this click to any of the possible message selection options as in the menu for [Next (R)]. See the variable **zwei:*next-middle-mode***.

[Next (R)] Pops up a menu of choices:

Next undeleted	Selects the next undeleted message.
Next	Selects the next message in the sequence, whether or not it has been marked for deletion.
Next unseen	Selects the next unseen message.
Next recent	Selects the next message in the recent sequence.
Last undeleted	Selects the last undeleted message in the sequence.
Last	Selects the last message in the buffer, whether or not it has been marked for deletion.
Last unseen	Selects the last unseen message in the sequence.
Last recent	Selects the last message in the recent sequence.

O (Kbd) Zmail Command

O (Kbd) "Outputs" the current message to a file, similar to [Move]. Prompts in the minibuffer for a pathname. The default is taken from your profile. See the variable **zwei:*default-move-mail-file-name***.

c-X O (Kbd) Zmail Command

c-X O (Kbd) Select another exposed window. Repeated use cycles through the two or three exposed windows. (Headers and Mail or Message, Headers, and Mail.)

Occur (m-X) Zmail Command

Occur (m-X) Prompts for a string and shows those lines within messages that contain the given string. You can select the message containing the string by clicking on the line of text. This is a handy way to search through a collection for a message you only vaguely remember. It is like the Find String option in the [Map Over] menu. See the section "[Map Over] Zmail Menu Item".

[Other] Zmail Menu Item

[Other] Repeats the last command given. There is no initial default so the first time you want to use one of these commands you must click right to get the menu.

[Other (M)] No option has been assigned to this mouse gesture.

[Other (R)] Pops up a menu of additional commands. Currently it includes:

Show File Prompts for a filename and shows the file in the typeout window.

Hardcopy Hardcopies the current message. Clicking left on Hardcopy uses the default printing device. Clicking right pops up a menu that allows you to select the device and other parameters. The device selected becomes the default for subsequent hardcopy commands.

Rename Sequence
Promps for a name and renames the current sequence to that name.

Whois Prompts for a user-id or surname and shows the information in that person's namespace entry. If you specify *name@host*, it searches the namespace and contacts *host* to obtain the information. It uses the ARPANET Name protocol so if your site is on the ARPANET, you can access ARPANET name servers.

P (Kbd) Zmail Command

n P (Kbd) Selects the *n*th previous undeleted message in the current sequence. If *n* is omitted, selects the previous undeleted message.

c-P (Kbd) Zmail Command

n c-P (Kbd) Selects the *n*th previous message in the sequence, whether or not it is marked for deletion. If *n* is omitted, selects the previous message, whether or not it is deleted.

zwei:preload-zmail &rest *files* *Function*

Starts up Zmail, loading in *files*.

```
(zwei:preload-zmail "wombat:>kjones>mail.text")
```

This gets the mail loading operation underway while you are doing something else.

These are the keyword options to **zwei:preload-zmail**:

:find-file Find the file and load it in for processing.

:examine-file Finds the file and reads it into Zmail but in read only mode.

As an example, the following form can be included in your LISPM-INIT to preload several mail files into Zmail with some of them being read only:

```
(zwei:preload-zmail '(:find-file "y:>palter>mailboxes>palter.xmail")
                    '(:find-file "y:>palter>mailboxes>reminders.xmail")
                    '(:examine-file "y:>palter>mailboxes>junk.xmail")
                    '(:examine-file "y:>palter>mailboxes>digest.xmail"))
```

:hang-when-deexposed
Controls the use of the Zmail background process. Zmail reads and parses the files in the background. If **(:hang-when-deexposed t)** is included at the end of the **zwei:preload-zmail** form, the Zmail background stops after reading the mail files in question without parsing the contained messages. The background parsing will commence as soon as Zmail is selected. The default for **:hang-when-deexposed** is **nil**, so use of **zwei:preload-zmail** without specifying **:hang-when-deexposed** causes mail parsing to begin in the background as soon as the loading is finished.

As an example of the use of **:edit-all-mail-files**, the form

```
(zwei:preload-zmail '(:examine-mail-file #p"LARRY-BIRD:>Palter>mailboxes>digests.kbin")
                    :edit-all-mail-files)
```

will preload Palter's digest `kbin` file with saving disabled and then preload all the other mail files listed in his profile.

Zmail's *Edit File* command, which is used to ask Zmacs to edit a file usually referenced by the current message, is bound to `c-X c-sh-F` at top level.

[Previous] Zmail Menu Item

[Previous] Selects the previous undeleted message in the current sequence.

[Previous (M)] Selects the first undeleted in the current sequence. You can set the action for this click to any of the possible message selection options as in the menu for [Previous (R)]. See the variable **zwei:*previous-middle-mode***.

[Previous (R)] Pops up a menu of choices:

Previous undeleted
Select the previous undeleted message.

Previous	Selects the previous message, whether or not it is marked for deletion.
Previous unseen	Selects the previous unseen message in the sequence.
Previous recent	Selects the previous message in the recent sequence.
First undeleted	Selects the first undeleted message in the sequence.
First	Selects the first message in the sequence, whether or not it has been marked for deletion.
First unseen	Select the first unseen message in the sequence.
First recent	Select the first message in the recent sequence.

[Profile] Zmail Menu Item

[Profile]	Puts you in the Profile window so you can edit your Zmail init file and alter the various profile options. See the section "Zmail Profile Options". See the section "Customizing Zmail".

`q` (Kbd) Zmail Command

`Q` (Kbd)	Expunge and save loaded mail files just like [Save], then return from Zmail to the window from which it was called.

[Quit] Zmail Menu Item

[Quit]	Expunge and save loaded mail files just like [Save], then return from Zmail to the window from which it was called.
[Quit (R)]	Pop up a menu of save and exit options. (See Figure 107.) The menu has two columns; one entry in each column is highlighted. The Save column has the following options: • `Don't Save` - Do not save any files before exiting. • `Ask` - Pop up an Expunge/Save/Kill menu to determine which files to expunge, save, or kill. See the description of [Save (R)]. • `Save` - Expunge and save loaded mail files like [Save]. The Exit column has the following options:

- Quit - Return from Zmail to the window from which it was called, burying the Zmail window.

- Logout - Log out from the machine, then return to the calling window.

Initially, Save and Quit are highlighted; this combination is equivalent to clicking left on [Quit]. Clicking on an unhighlighted entry highlights it and unhighlights the others in its column. Clicking on Do It does the saving and exiting indicated in the menu; clicking on Abort aborts the Quit command.

R (Kbd) Zmail Command

R (Kbd)

Starts a reply to the current message. With an argument of 1, replies only to the sender of the message. You can set the behavior of reply commands with an argument of 1 in your profile. See the variable **zwei:*1r-reply-mode***. The numeric arguments accepted are:

argument	*Reply Window Mode/Reply Mode*
n = 1	Two-windows/Sender.
n = 2	Two-windows/All.
n = 3	Yank/All.

See the section "Zmail Reply Command".

c-R (Kbd) Zmail Command

c-R (Kbd)

Puts you in an editing window with the current message. This is the same as clicking left on the message window. The headers of the message are expanded to their full form in the editing window.

Redirect Message (m-X) Command

Redirect Message (m-X)

Redirects a message to a different group or individual. It prompts for recipients to be removed from the recipient list, then for recipients to be added, and finally for a comment explaining the redirection. Then it emends the message, adding the new recipients, removing any you indicated should be removed, and inserting any comment you supplied about the redirection. It sends this emended message to the new recipi-

ents and sends a second message to the original recipients, referencing the original message and informing them that it has been redirected. It also revokes the original message (See the section "Revoke Message (m-X) Zmail Command".) Finally, it updates the recipient fields of the message in your current sequence and prompts you to reply to the redirected message immediately.

Redirecting a message is different from redistributing a message in that it reroutes the message and any subsequent conversation to a new set of people. Redistributing just sends out additional copies of the original message, it does not automatically include the new recipients in any subsequent conversation. Redirecting is particularly useful for tracking bug messages and directing them to the appropriate mailing lists.

The redirected message (that is, the current message) is given the (redirected) property; but this property is not written out for Babyl files.

Redistribute Message (m-X) Command

Redistribute Message (m-X)

Redistribute the current message to other recipients. Prompts in the mode line for entry of the recipients of the redistributed message. The recipients you specify receive a copy of the current message with three additional header fields (Redistributed-to:, Redistributed-by:, and Redistributed-date:) describing the redistribution. Redistributing differs from Forwarding in that the original message is passed on, with its original message id, to additional recipients; no new message is created. Redirecting (See the section "Redirect Message (m-X) Zmail Command".) is similar to Redistribute except that it also reroutes the entire conversation and is a two step process, some recipients are removed and others added.

(Note: The redistributed message (that is, the current message) is given the (redistributed) property.)

Redo (m-X) Zmail Command

Redo (m-X) — Undoes the effect of the last Undo (m-X).

Rename Sequence (m-X) Zmail Command

Rename Sequence (m-X)

Prompts for a new name for the sequence. If the sequence is a buffer with an associated file name, Rename Sequence renames the file.

Repeat Last Matching MiniBuffer Command (m-X) Zmail Command

Repeat Last Mini Buffer Command (m-X)

c-m-sh-Y Repeat Last Matching Minibuffer Command

Yanks back and repeats the last minibuffer command that includes a string you specify. m-sh-Y yanks back previous commands that contain the same string.

Repeat Last MiniBuffer Command (m-X) Zmail Command

Repeat Last Mini Buffer Command (m-X)

c-m-Y Repeat Last Minibuffer Command

Repeats a recent minibuffer command. It yanks the displayed default if there is one; otherwise, it yanks the last thing typed in this context. A numeric argument *n* yanks the *n*th previous one. An argument of 0 lists the history of elements typed in the minibuffer.

For a similar command with string-matching, see the section "Repeat Last Matching MiniBuffer Command (m-X) Zmail Command".

m-Y yanks successively earlier mini buffer commands.

[Reply] Zmail Menu Item

[Reply] Starts a reply to the current message with the reply window mode Two-windows and the reply mode All. See the section "Zmail Reply Command".

[Reply (M)] Starts a reply to the current message with the window mode Two-windows and the reply mode Sender.

[Reply (R)] Pop up a three-column menu of reply options.

The first column offers choices for recipients of the reply. The second column offers window configurations. The third column controls the pruning of headers on included messages. Initially, All, Two-windows, and Prune are highlighted; this combination is equivalent to using [Reply]. Clicking on an unhighlighted entry highlights it and unhighlights the others in its column. Clicking on `Do It` enters mail mode; clicking on `Abort` aborts the Reply command. You can set the initial choices that appear on the menu in your Zmail init file. See the section "Zmail Options for Replying to Mail ". See the section "Zmail Options for Including Messages in a Reply".

Restore Draft File Zmail Menu Item

[Restore Draft File] (Editor Menu)

`c-X c-R` (Kbd) Restores a previously saved draft. The current contents of the Headers and Mail windows are lost.

Revoke Message (`m-X`) Zmail Command

Revoke Message (`m-X`)

Pops up a menu of all the messages sent in the current Zmail session. You select the one you wish to revoke by clicking on it with the mouse. The last choice in the menu is *Revoke message in current sequence*. Clicking on this choice permits you to select a message to revoke by message number or by clicking on it in the summary window. If the message to be revoked appears in one of your loaded mail files, it is marked for deletion.

Revoke Message adds a *revoke* message to the inbox of each of the recipients of the message to be revoked, that is a message whose header says Revoke Message and gives the message id.

`S` (Kbd) Zmail Command

`S` (Kbd) Expunge (that is, get rid of all messages marked for deletion) and save all loaded mail files that have been modified since the last save, see the section "Zmail Message Deletion Commands". Because it first expunges the mail sequences, thereby destroying information, and because it can be a time consuming operation if your mail files are large, `S` asks for confirmation before proceeding. You can turn off this query in your

Zmail profile by setting the option "Ask permission before executing time consuming and/or destructive commands" to No; see the variable **zwei:*ask-before-executing-dangerous-zmail-commands***.

[Save] Zmail Menu Item

[Save] Expunge (that is, get rid of all messages marked for deletion) and save all loaded mail files that have been modified since the last save. See the section "Zmail Message Deletion Commands".

[Save (M)] Expunge the current mail file or sequence.

[Save (R)] Pop up a multiple choice Expunge/Save/Kill window. (See Figure 106.)

Each row of the menu lists a loaded mail file and boxes for three choices: Expunge, Save, and Kill. An x in the Expunge box means expunge the file; an x in the Save box means save the file; and an x in the Kill box means kill the loaded copy — *not* the disk copy — of the file, that is, make Zmail forget about the file. The initial configuration of x's and blanks shows what using [Save] would do, which is to expunge files with deleted messages and save files modified since the last save.

Clicking left or right on a box complements its status, removing an x if present, adding one if not. Marking a file for expunging or saving clears the Kill box; marking a file for killing clears the Save and Expunge boxes. Clicking on `Do It` performs the selected Expunge, Save, and Kill operations; clicking on `Abort` aborts the Save files command.

Save Current Buffer (m-X) Zmail Command

Save Current Buffer (m-X)

Checks to see if the current sequence has been modified and saves it if necessary. It does not do an expunge before saving.

Save Draft as Message Zmail Command

[Save Draft As Message] (Editor Menu)

Save Draft As Message (m-X)

c-X c-m-S (Kbd) Saves the text of the message being composed as a message. To specify a specific buffer, specify a numeric argument of 2. If the message has already been saved, Zmail does not resave

it unless you specify a numeric argument of 4 (or c-U). The arguments actually are actually dealt with bit-wise, so an argument of 6 has the combined effect of an argument of 2 and an argument of 4.

Save Draft As Message (m-X) prefers the current sequence over the default sequence if the current sequence happens to be a buffer.

Save Draft File Zmail Menu Item

[Save Draft File] (Editor Menu)

c-X c-S (Kbd) Saves the message being composed in a disk file. The first time it is used, it prompts for entry of a filename; subsequently, it uses the same filename.

[Select] Zmail Menu Item

[Select] Selects the previously selected mail collection. There is no initial default so the first time in a Zmail session that you want to select another mail file, you must click right on [Select] for the menu.

[Select (M)] Selects messages by a filter, using the Filter Selection Display. See the section "Selecting Zmail Filters".

[Select (R)] Selects the mail sequence specified from a menu of all the mail sequences in your current Zmail, any mail files stored in your profile but not yet read into Zmail (see the section "Saving a List of Mail Files"), and four other options:

Read/Create File Prompts for a mail file, creating the file if it does not exist.

Examine File Prompts for a mail file and reads it in No Save mode.

Mark Survey Allows you to select messages by their summary lines to create a new collection. See the section "Creating a Mail Collection by Marking Individual Messages".

Filter Pops up the Filter Selection Display. See the section "Selecting Zmail Filters".

Select All Conversations By References (m-X) Zmail Command

Select All Conversations By References (m-X)
Selects messages to which a message in the sequence refers, or that refer to a message in the sequence, recursively; this is implemented by **zwei:com-zmail-select-all-conversations-by-**

references. It is equivalent to appending together all sequences gotten from Select Conversation By References (m-X) for each message in the current sequence. An argument gives a menu of universes to search. The command defaults to loaded files. You can also perform this operation using [Map Over (R) / Select Conversation].

Select Arbitrary Format Mail File (m-X) Zmail Command

Select Arbitrary Format Mail File (m-X)
Prompts for a mail file and then for the format to read that file in. Use this command if you need to read or create a mail file that is not in the standard format for the machine on which it is stored.

Select Conversation By References (m-X) Zmail Command

Select Conversation by References (m-X)
Defines a conversation and selects it as a collection. This command is very similar to Select References.

Select Referenced Message (m-X) Zmail Command

Select Referenced Message (m-X)
Selects the referenced message as current.

Select References (m-X) Zmail Command

Select References (m-X)
Creates a mail collection of all messages referenced by the current message. The collection also includes messages referenced by the referenced messages, messages referenced by *them*, and so forth.

Show Draft Dispositions (m-X) Zmail Command

Show Draft Dispositions (m-X)
Displays a list of the messages you have sent in your current session, including to whom the message was sent, the subject

line, the time and date of transmission, and which mail server handled the message. Drafts of messages that have been sent can be retrieved for retransmission. See the section "Continuing Completed or Aborted Zmail Messages". Clicking right on a draft offers a menu that lets you revoke the draft.

Show File (m-X) Zmail Command

Show File (m-X)

Prompts for a pathname and displays the specified file. The default is the first pathname specified in the File-References: header field.

Show Mail (m-X) Zmail Command

Show Mail (m-X) A command for showing your inbox file. It uses the standard mail pathname for your home directory. When no new mail has been delivered recently, it reports "No new mail". This command uses Show File.

Show Printer Status Zmail Command

Show Printer Status (m-X)
Prompts for the name of a printer and displays its print queue.

[Sort] Zmail Menu Item

[Sort] Sorts the current sequence using the same sort keys as the previous sort. The default is Forward by Date.

[Sort (M)] No option has been assigned to this mouse gesture.

[Sort (R)] Pops up a highlighted menu of sort keys.
You can sort Forward or Backward and by several other keys. Selecting Backward by Date sorts your mail into "most recent first" order.

Start Background Save (m-X) Zmail Command

Start Background Save (m-X)
Checks to see if the current sequence has been modified and saves it in the background if necessary.

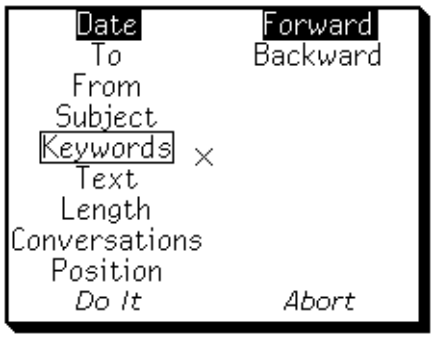


Figure 118. [Sort (R)] Menu

Start/End of Summary Window Commands

The commands, m-X Start of Summary Window (c-m-<) and m-X End of Summary Window (c-m->) move the summary window to the beginning and end of the current sequence without affecting the current message.

[Survey] Zmail Menu Item

Displays the summary lines of messages in the current sequence. The display is mouse sensitive. See the section "Zmail Message Summary Line".

[Survey] Displays summary lines of all the messages in the current sequence. This is an easy way to semi-automatically scroll through all your messages.

[Survey (M)] Displays summary lines of the messages in the same conversation as the current message.

[Survey (R)] Pops up the filter selection menu so that you can specify a filter to select the summary lines to be displayed. See the section "Selecting Zmail Filters".

c-SPACE (Kbd) Zmail Command

c-SPACE Push or pop the message stack, depending on the argument. There are three meaningful forms:

c-SPACE Push the current message onto the stack. (Does not change the current message.) The stack can hold up to eight elements; pushing onto a full stack causes the bottom element to be lost.

A message is automatically pushed on the stack by Zmail whenever you use a command that causes or can cause movement from the current message, except the following:

N, c-N
P, c-P
D, c-D
[Next] or [Previous]
[Delete], [Delete (M)], or [Delete (R)]
[Next (R)] — ([Next] and [Next undeleted] options.)
[Previous (R)] —
([Previous] and [Previous undeleted] options.)
n J — (When *n* is equal to the current message number.)
Clicking left on the summary line of the current message.

None of these commands moves very far from the current message, unless they move over a long string of deleted messages. Also, none of the message stack commands automatically push a message on the stack.

c-U c-SPACE — Pop the top message from the message stack and make it the current message. (The element popped is also tucked back under the stack as the new bottom element.) If the stack is empty Zmail flashes the screen.

c-U c-U c-SPACE — Pop the top message from the message stack and discard it. The setting of the current message does not change. (The element popped is also tucked back under the stack as the new bottom element.) If the stack is empty Zmail flashes the screen.

U (Kbd) Zmail Command

n U (Kbd) — Undeletes message number *n*. If *n* is negative or larger than the number of messages in file, it complains "Argument out of range". If message *n* is not deleted, it complains "Message not deleted." If you omit *n*, it is the same as [Undelete]. See the section "[Undelete] Zmail Menu Item". See the section "Zmail Message Deletion Commands".

[Undelete] Zmail Menu Item

[Undelete] (Menu) — Start at current message and searches backward for a deleted message, undeletes it, and select it as the current message. Complain "No deleted messages" if there are none. [Undelete (M)] and [Undelete (R)] are the same as clicking left on [Undelete]. See the section "Zmail Message Deletion Commands".

Undigestify (m-X) Zmail Command

Undigestify (m-X) Converts a "standard arpanet" digest message into smaller messages. These messages are inserted into the current buffer right after the digest message. A References: header is added to the original message pointed to all the exploded messages, allowing use of the conversation commands to select the digest messages into a collection, delete them all, and so forth.

Undo (m-X) Zmail Command

Undo (m-X) Undoes the last nontrivial, potentially destructive command; Using Undo (m-X) successively undoes earlier and earlier commands.

For example, after using the Sort menu command, Undo (m-X) restores the previous order of messages in the file.

Write Draft File Zmail Menu Item

[Write Draft File] (Editor Menu)

c-X c-W (Kbd) Saves the message being composed in a disk file. Prompts for entry of a filename.

c-X Y Prune Yanked Headers Zmail Command

c-X Y (Kbd)

[Prune Yanked Headers] (Editor Menu)

Deletes the less essential headers of a message that was yanked in via c-X c-Y. Leaves only the Date: and From: headers; these are sufficient to identify the message. The profile option *Prune headers of yanked messages* controls the automatic pruning of message headers yanked into a reply. See the variable **zwei:*prune-headers-after-yanking***. The default is to not prune headers.

c-X c-Y (Kbd) Yank Replied Messages Zmail Command

n c-X c-Y (Kbd) Yanks the message(s) being replied to into the buffer. (Used most often when replying to the current message.) If in two-window mode, go into one-window mode. Indent the yanked message unless an argument *n* is given. The arguments to c-X c-Y control the indentation and the pruning of headers, as follows:

Argument	*Options*
none	Indentation, pruning per the *Prune headers of yanked messages* profile option.
1	No indentation, pruning per *Prune headers of yanked messages*.
2	Indentation, pruning per reverse of *Prune headers of yanked messages*.
3	No indentation, pruning per reverse of *Prune headers of yanked messages*.

Yank Current Message (m-X) Zmail Command

Yanks Current Message (m-X)
: Yanks the current message into the message being composed.

Converse

Introduction to Converse

Converse is a facility for communicating interactively with other logged-in users. A message sent with Converse pops up on the screen of the recipient almost instantaneously. The recipient has the choice of replying right in the pop-up window, entering Converse to reply, or doing nothing.

The Converse interactive message editor is operated by a window with its own process. Converse keeps track of all of the messages that you have received or sent. The Converse window shows all of the messages that have been sent or received since the machine was cold booted.

Messages sent between you and another user are organized into a *conversation*. Conversations are separated from each other by a thick black line. Within each conversation are all messages, outgoing and incoming, arranged in chronological order, and separated by thin black lines.

You can use Converse to look at conversations, send messages, and receive messages. Converse is built on the Zwei editor, so you can edit your message as you type it, or pick text up and move it around between one message and another, or among messages, files, and pieces of mail.

To enter Converse, do one of the following:

- Press SELECT C.
- Give the Command Processor command Select Activity Converse.

- Evaluate **(zl:qsend)**.
- Click on [Select / Converse] in the System menu.
- Answer `C` in the Converse pop-up window when a message arrives.

Using Converse

Sending and Replying to Messages with Converse

When you enter Converse for the first time, the window is empty except for a blank message at the top of the screen, starting with `To:` (see Figure !).

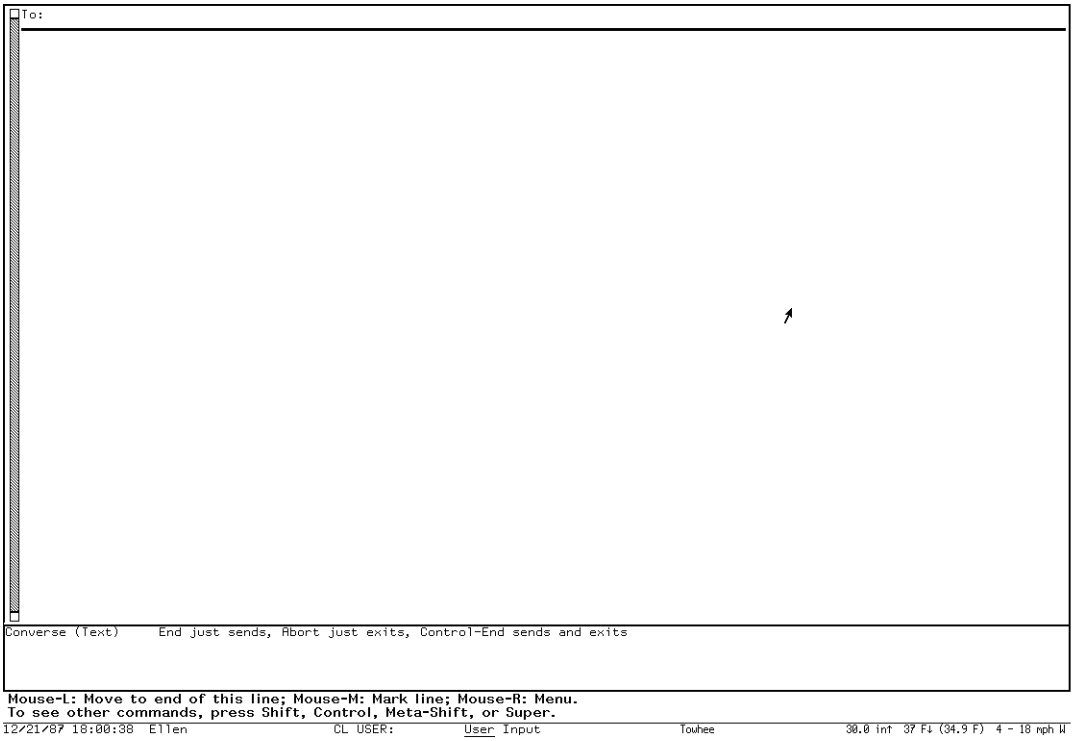


Figure 119. A Fresh Converse Window

You start a message by filling in a recipient after the `To:`, pressing `RETURN` and then typing the message text. It is not necessary to know what machine the person is using, but if you do know and give the recipient as *name@host* the message is sent considerably faster, since it is not necessary to search the namespace to find the machine (see Figure !). To send the finished message, press `END`.

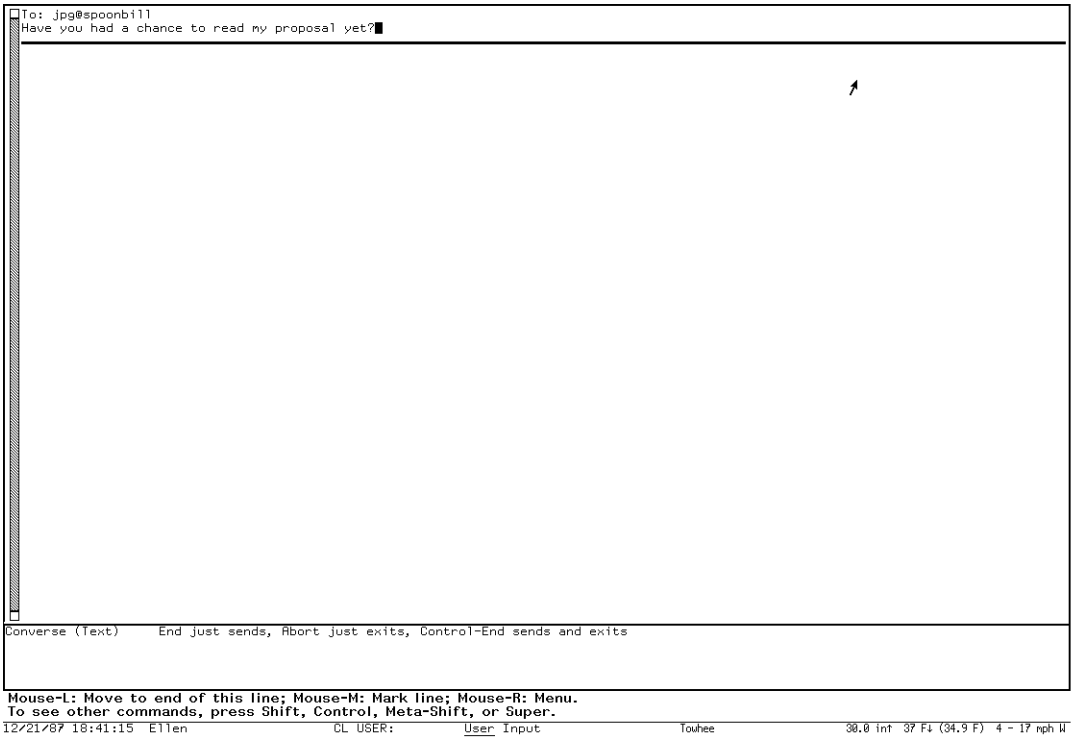


Figure 120. A Converse Message About to be Sent

When the message has been sent successfully, it appears as part of a conversation. A blank message remains at the top of the screen, and just below that, a heavy black line delimits the message(s) of the conversation you just started. Just below the heavy black line is another blank message, but this one has the name of the person to whom you sent the message filled in. Below this blank message, separated by a thin black line, the message you just sent appears, with the date and time it was sent.

When the person to whom you sent the message replies, the reply appears in the conversation above the message you sent, and below the blank message. (See Figure 121 .) Your cursor is left in the blank message so you can reply easily.

You use regular editor commands to move around in the Converse window. Two commands, specific to Converse, are particularly useful: `c-m-]` (move to next conversation) and `c-m-[` (move to previous conversation).

You exit from Converse by pressing `ABORT` or by selecting another window. You can also press `c-END` when sending a message to send the message and exit from Converse.

To start a conversation, enter Converse, go to the top of the Converse window and fill in the blank message, starting with the `To:` line to specify the new recipient.

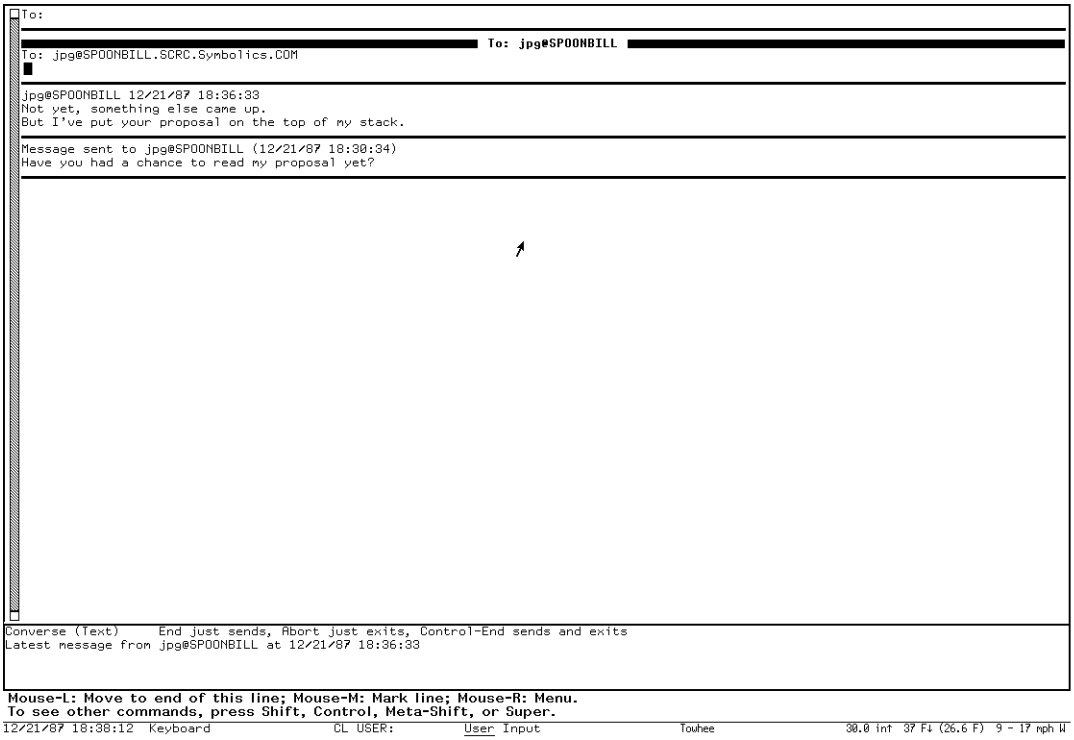


Figure 121. A Converse Conversation

Finish by pressing END to send the message. To send the message and exit Converse, finish by pressing c-END.

To send a message as part of an existing conversation, find that conversation in Converse and fill in the blank message at the beginning of the conversation, finishing by pressing END to send the message, or by pressing c-END to send the message and exit Converse.

You do not have to be in the main Converse window to receive messages. Converse will deliver a message to you in any window. Since this might be annoying, you can customize what happens when a message arrives by using the variable **zwei:*converse-mode***. See the section "Customizing Converse".

When you are in a window other than Converse and a new message arrives, a window pops up at the top of the screen displaying the message. You can respond R to type in a reply, N (for "no action") to make the message window deexpose, or C to enter Converse. Entering Converse has several advantages: you can look over the previous messages in the conversation, and you can use the Zwei editor to help you construct a reply. See Figure !.

Converse remembers all messages that you send or receive, even if you did not use the main Converse window to send them or reply to them.

Figure 122. A Converse pop-up window

Converse lets you know as soon as a message comes in, by beeping or flashing the screen, and if it is supposed to notify you, it does so without waiting for the main Converse process to wake up. In pop-up mode, if the pop-up message window is already in use, an incoming message causes the message window to beep or flash but not to display the message. This is necessary since only one message at a time should pop up. When the message window is deexposed it is reexposed immediately with the new message in it.

If the main Converse window is exposed, a new message is shown there with its conversation; it is never shown via a notification or a pop-up message window. If the main Converse window is exposed but its process is busy (typically, when it is in the Debugger or in an editor command and waiting for typein), Converse beeps or flashes but does not display the message. You can display the message by clearing the Converse process. You can usually clear the Converse process by pressing ABORT.

Converse Commands

Converse has several commands for managing your conversations.

HELP	Displays a summary of Converse commands.
END	Sends the current message. The behavior of this key can be changed by the variable **zwei:*converse-end-exits***.
c-END	Sends the current message and exits from Converse. The behavior of this key can be changed by the variable **zwei:*converse-end-exits***.
ABORT	Exits Converse.

c-M — Mails the current message instead of sending it. This is useful if Converse reports that the person to whom you want to send the message is not logged in anywhere.

c-m-[— Moves to the previous conversation.

c-m-] — Moves to the next conversation.

Delete Conversation (m-X)
Deletes the current conversation from the Converse window.

Write Buffer (m-X) Writes the entire Converse buffer (all conversations) to a file. It prompts for a pathname.

Write Conversation (m-X)
Writes only the current conversation to a file. It prompts for a pathname.

Append Buffer (m-X)
Appends the entire Converse buffer (all conversations) to the end of a file. It prompts for a pathname.

Append Conversation (m-X)
Appends only the current conversation to the end of a file. It prompts for a pathname.

Regenerate Buffer (m-X)
Rebuilds the structure of the Converse buffer. This might be necessary if you damage the buffer in some way, for instance by removing one of the black lines separating conversations. Some error messages might ask you to give this command and try again. The message you are currently typing might be lost, but you can prevent this by putting the text on the kill ring by marking it and using m-W before issuing the m-X Regenerate Buffer command.

Lisp Listener Commands for Converse

Command Processor Commands for Converse:

Send Message **Command**

Send Message *recipient*

Sends a Converse message to the specified recipient.

recipient — *user* or *user@host*. The person to whom to send the message. If *@host* is omitted, all Symbolics machines on your network are

polled to locate *user*.

Send Message prompts for text to send as a Converse message. END terminates and sends the message. See the section "Converse".

Show Messages **Command**

Show Messages *keywords*

Displays the contents of the specified Converse conversations.

keywords :Direction, :From, :Mention Empty Sequences, :More Processing, :Order, :Output Destination, :Query, :Recent, :Start, :Stop, :Summarize, :To

:Direction {Incoming, Outgoing, All, or Default} Whether to show incoming messages, outgoing messages, or all. The Default is Incoming.

:From {*user-or-address*} Show messages from this user or address.

:Mention Empty Sequences
{Yes, No} Whether to mention empty message sequences, for example, you have sent messages to someone but the person did not reply. The default is No, not to mention this. If it is Yes, you see "No messages from *so-and-so*".

:More Processing {Default, Yes, No} Controls whether **More** processing at end of page is enabled during output to interactive streams. The default is Default. If No, output from this command is not subject to **More** processing. If Default, output from this command is subject to the prevailing setting of **More** processing for the window. If Yes, output from this command is subject to **More** processing unless it was disabled globally (see the section "FUNCTION M").

:Order {Forward, Reverse} How to order the message presentation within each conversation. The default is Forward, that is, most recent first.

:Output Destination
{Buffer, File, Kill Ring, None, Printer, Stream, Window} Where to redirect the typeout done by this command. The default is the stream ***standard-output***.

:Query {Yes, No} Whether to ask about each conversation. The default is Yes, to ask.

:Recent {Yes, No} Whether to consider only the most recently exchanged messages in each conversation.

:Start	{*number*} Number of first message to show in a conversation. If there are fewer then *number* messages in the conversation, that conversation is skipped.
:Stop	{*number*} Number of last message to show in a conversation.
:Summarize	{Yes, No} Whether to show the entire message or just a summary. The default is No, to show the entire message. If yes, messages are mentioned but not shown.
:To	{*user-or-address*} Show messages to this user or address.

Lisp Functions to Control Converse:

zwei:qsends-off &optional (*gag-message* **t**) *Function*

Refuses interactive messages. If you give it a string argument, *gag-message*, the variable **zwei:*converse-gagged*** is set to this string and the string is returned to anyone who tries to send a message to you. Otherwise, they just get a note saying that you are not accepting messages. **zwei:qsends-on** turns sends back on and clears **zwei:*converse-gagged***.

zwei:qsends-on *Function*

After using **zwei:qsends-off** to notify interactive message senders that you are not accepting messages, **zwei:qsends-on** allows interactive messages to be received again.

net:notify-local-lispms &optional *message* &key *(:error-p) (:report) (:output-stream)* *Function*

Sends *message* to all Symbolics machines at your site based upon information it gets from the namespace database about the Symbolics machines at the local site. *message* should be a string; if it is not provided, the function prompts for a message. Each recipient receives the message as a notification, rather than as an interactive message.

Keyword arguments are:

:error-p
Setting this keyword to **t** enables the function to report all errors encountered. Specifying **nil** (default) for this keyword enables the function to ignore all errors encountered.

:report
Setting this keyword to **t** (default) enables the function to report whether it succeeded in delivering the message. Specifying **nil** enables the function to only report failures in delivering messages.

:output-stream
Using this keyword enables you to redirect output to a specific stream.

zl:qsend &optional *destination message* *Function*

Sends interactive messages to users on other machines on the network.

destination is normally a string of the form *name@host*, to specify the recipient. If you omit the *@host* part and give just a name, **zl:qsend** looks at all the Symbolics machines at your site to find any that *name* is logged into. If the user is logged in to one Symbolics machine, it is used as the host; if more than one, **zl:qsend** asks you which one you mean. If you leave out *destination* altogether, doing just **(zl:qsend)**, Converse is selected as if you had pressed `SELECT C`.

message should be a string. For example:

```
(qsend kjones@wombat "Want to go to lunch?")
```

If *message* is omitted, **zl:qsend** asks you to type in a message. You should type in the contents of your message and press `END` when you are done.

The input editor is used while you type in a message to **zl:qsend**. So you get some editing power, although not as much as with full Converse (since the latter uses Zwei). See the section "Editing Your Input". **zl:qsend** predates Converse and is retained for compatibility.

print-sends &optional (*stream* **zl:standard-output**) *Function*

Prints out all messages you have received (but not messages you have sent), in forward chronological order, to *stream*. Converse is more useful for looking at your messages, but this function predates Converse and is retained for compatibility.

zl:qreply &optional *text* *Function*

Sends a reply to the Converse message received most recently. You can supply a string as the text of the message or omit it and let **zl:qreply** prompt for it. It returns a string of the form "*user@host*", indicating the recipient of the message. This function predates Converse and is retained for compatibility.

Customizing Converse

The following variables allow you to customize Converse's behavior. You can set them in your init file.

zwei:*converse-mode* *Variable*

Controls what happens when an interactive message arrives. It should have one of the following values:

:pop-up (This is the default.) A message window pops up at the top of the screen, displaying the message. You are asked to type R (for Reply), N (for Nothing), or C (for Converse). If you type R, you can type a reply to the message inside the message window. When you press END, this reply is sent to whomever sent the original message to you, and the pop-up message window window disappears. If you type N, the message window disappears immediately. If you type C, the Converse window is selected. The input editor is used while you reply to a message in the pop-up message window, so you get some editing power, although not as much as with full Converse.

:auto The Converse window is selected. This is the window that shows you all of your conversations, letting you see everything that has happened, and letting you edit your replies with the full power of the editor. With this window selected, you can reply to the message that was sent, send new messages, participate in other conversations, or edit and write out messages or conversations. You can exit with c-END or ABORT (c-END sends a message and exits; ABORT just exits), or you can select a new window by any of the usual means (such as the FUNCTION or SELECT keys).

:notify A notification is printed, telling you that a message arrived and from whom. If you want to see the message, enter Converse by pressing SELECT C. There you can read the message and reply if you want to.

:notify-with-message
A notification is printed, which includes the entire contents of the message and the name of the sender. If you want to reply, you can enter Converse.

zwei:*converse-append-p* *Variable*

If the value is **nil** (the default), a new message is prepended to its conversation. If the value is **t**, a new message is appended to its conversation.

zwei:*converse-beep-count* *Variable*

The value is the number of times to beep or flash the screen when a message arrives. The default value is two. Beeping or flashing occurs only if the Converse window is exposed or if the value of **zwei:*converse-mode*** is **:pop-up** or **:auto**. (Otherwise, notification tells you about the message and includes the usual beeping or flashing.)

zwei:*converse-end-exits* *Variable*

Controls the behavior of END and c-END. If **zwei:*converse-end-exits*** is set to **nil**, the default, END sends the message and you remain in Converse. c-END sends the message and exits Converse. Setting **zwei:*converse-end-exits*** to **t** reverses this, so that c-END sends the message and remains in Converse and END sends and exits.