Z-Mail Reference Manual

Version 3.2

Z-Code Software
A Division of NCD

101 Rowland Way, Suite 300
Novato, CA 94945
+1 415–898–8649

Important Notice

Printed in the United States of America.

CONTENTS

Z-Mail Reference Manual

VERSION 3.2

Chapter 1	***Introducing Z-Mail***	***5***
	Using Z-Mail Documentation	7
	Conventions Used in this Manual	8
	Where You Should Start	10
Chapter 2	***Z-Mail Basics***	***13***
	E-Mail Terminology	14
	Z-Mail's User Interfaces	20
	Using Help	22
	Specifying Message Lists	24
	Specifying Text Patterns	26
	Specifying Filename Patterns	27
	Specifying E-Mail Addresses	29
Chapter 3	***Starting and Initializing Z-Mail***	***33***
	Starting Z-Mail	34
	Z-Mail Options Summary	36
	Z-Mail Options Reference	40
	Initializing Z-Mail	63
	Managing Initialization Files	66

Chapter 4 *Using Z-Mail's Command-Line Interface* *71*

Starting Z-Mail's Command-Line Interface 72
Command-Line Interface Basics 73
Tilde-Escape Editing Commands 91
Using Macros 102

Chapter 5 *Z-Script Commands* *109*

?	117	filter	143	page	186	source	210
about	117	flags	144	pick	186	stop	211
alias	117	folder	145	pinup	187	stty	211
alternates	119	folders	152	pipe	187	task_meter	211
ask	120	foreach	153	preserve	189	thwart	212
attach	122	from	154	previous	189	top	212
await	123	function	155	print	190	trap	213
builtin	124	headers	157	printenv	190	type	213
button	124	help	160	pwd	190	unalias	213
chroot	129	hide	160	quit	190	unbutton	214
cd	129	history	160	read	191	uncmd	214
close	130	iconify	161	redraw	191	undelete	214
cmd	131	if...else...endif	161	remove	192	undigest	215
compcmd	131	interpose	166	rename	193	unfilter	216
copy	133	ignore	168	reply	193	unfunction	216
debug	133	jobs	169	replyall	194	un_hdr	216
delete	134	lpr	170	resume	195	unhide	217
detach	135	ls	171	retain	195	uniconify	217
dialog	136	mail	171	return	196	unignore	217
disable	136	map	175	rmfolder	196	unmark	218
display	137	map!	175	save	196	unmenu	218
each	138	mark	175	saveopts	198	unpreserve	219
echo	139	merge	176	search	199	unretain	219
edit	140	msg_list	177	set	202	unset	219
enable	140	menu	178	setenv	204	unsetenv	220
error	141	my_hdr	184	sh	205	update	220
eval	141	next	185	shift	206	version	221
exit	142	open	186	sort	208	write	221
expand	142						

Chapter 6 *Z-Mail Variables* *223*

Types of Variables. 224
Setting Variables. 225
Storing Variable Settings 227
Using Variables 228

Variable Summary 233
Variable Reference 248

address_book 248
address_cache 249
address_check 249
address_filter 250
address_sort 250
alternates 250
alwaysexpand 251
alwaysignore 251
always_sort 252
ask 252
askcc 253
attach_types 253
autoclear 253
autodismiss 254
autodisplay 254
autoedit 254
autoformat 255
autoiconify 255
autoinclude 255
autoprint 256
auto_route 256
autotyper 258
autosave_count 259
autosign 259
autosign2 260
baud 263
cdpath 264
child 264
cmd_help 264
colors_db 264
comp_attach_label 265
complete 265
compose_lines 267
compose_panes 267
compose_state 267
crt 268
cwd 269
date_received 269
dead 269
deletesave 269
detach_dir 270
display_headers 270
domain_route 270
dot 271
dot_lock 272
edit_hdrs 273
editor 274
escape 274
expire_cache 275
fetch_time_out 275
fignore 275
filelist_fmt 276
folder 277
folder_title 277
folder_type 277
fonts_db 278
fortune 278
fortunates 279
gui_help 279
hangup 280
hdrs_only 280
hidden 281
history 281
hold 281
home 281
hostname 282
i18n_read_mailbox 282
i18n_write_mailbox 282
i18n_mail_transport 283
ignore_bang 283
ignoreeof 283
indent_str 283
index_dir 284
index_size 284
input 285
in_reply_to 285
intr_level 286
is_gui 286
is_lite 287
is_sending 287
is_shell 288
ispeller 288
keepsave 288
known_hosts 288
logfile 289
lookup_max 290
lookup_sep 290
mail_icon 290
mailbox_name 290
main_folder_title 290
main_panes 291
main_state 291
mbox 292
message_lines 292
message_panes 292
message_state 293
metoo 294
mil_time 294
msg_attach_label 294
msg_win_hdr_fmt 295
newline 295
newmail_icon 296
no_expand 296
no_hdrs 297
no_menubars 297
nonobang 297
nosave 297
openfolders 298
output 298
pager 298
picky_mta 299
pop_options 299
pop_timeout 299
post_indent_str 300
pre_indent_str 300
presign 301
printer 301
printer_opt 302
print_cmd 302
prompt 303
quiet 303
realname 304
record 304
record_users 305
recursive 306
redirect 306
reply_to_hdr 306
save_empty 307
screen 307
sendmail 307
show_deleted 307
sort 308
speller 308
squeeze 308
status 309
summary_lines 309
summary_fmt 309
templates 310
textpart_charset 310
thisfolder 311
timeout 311
tmpdir 311
toplines 312
unix 312
user 313
verbose 313
verify 313
version 314
visual 314
warning 314
window_shell 315
wineditor 315
winterm 316
wrap 316
wrapcolumn 316
$$ 317
$* 317
$# 318
$0 318
$1, $2, . . . 318

Chapter 7 ***Integrating Z-Mail with Other Products*** ***321***

Importing Text Files into Messages. 323
Launching Applications from Z-Mail 325
Defining Attachment Types 328

Chapter 8 ***Using Z-Mail X Resource Variables*** ***335***

CHAPTER 1

Introducing Z-Mail

In the rapidly growing computing environments of networked servers and workstations, electronic mail (e-mail) is the most popular method of personal communication. In large organizations, it is common for some people to receive a hundred or more e-mail messages per day. Z-Mail is a comprehensive electronic mail system that not only enables you to send and receive e-mail messages, but also helps you manage your e-mail efficiently.

Multiple interfaces

Z-Mail gives you a choice of user interfaces that make you more productive regardless of what UNIX computer or terminal you use.

If you have a graphical workstation with the X Window System or an X Window System terminal, you can use Z-Mail's point-and-click graphical user interface (GUI).

If you have a character terminal, you can use the Z-Mail Lite character interface, which is designed to mimic the GUI interface in most respects. Lite provides dialog boxes, pulldown menus, and a highly configurable user interface, just like the GUI interface.

If you're an experienced user, you may prefer Z-Mail's command-line interface. In the command-line interface, you have access to all of Z-Mail's features and capabilities in a text-only form.

The “fullscreen” interface that was available in earlier versions of Z-Mail has been superseded by Z-Mail Lite. If you wish, you can configure Z-Mail Lite to behave like the older fullscreen interface.

On-line help

If you need on-line information about a feature or function, Z-Mail provides context-sensitive help from any part of the program. In the Motif and Lite interfaces, each dialog box includes a *Help* button. The command-line interface provides help via various **help** commands.

Z-Script

For more experienced mail users, Z-Mail includes Z-Script,™ a scripting language for creating message filters, functions, GUI and Lite buttons, and dialog boxes. You can use Z-Script to customize your Z-Mail installation so it works exactly the way you want.

Using Z-Mail Documentation

Z-Mail User Guide

The *Z-Mail User Guide* helps you understand the basic functions of Z-Mail and how to use them. The User Guide is available in Motif, Open Look, and Lite versions.

Z-Mail QuickStart Guide

The *QuickStart Guide* is an abbreviated set of instructions that introduces new users to the basics of Z-Mail.

Z-Mail Reference Manual

The *Z-Mail Reference Manual* — which you're reading now — is intended for experienced users, system administrators, and intermediate users who wish to become experts. It is organized to help you understand basic Z-Mail concepts, administer your Z-Mail installation, configure Z-Mail to meet your specific needs, get the most out of each Z-Mail interface, and use the Z-Script language.

Z-Mail Installation Guide

The *Z-Mail Installation Guide* provides step-by-step instructions for successfully installing Z-Mail. The guide assumes you are familiar with your equipment and the UNIX operating environment.

You can obtain any of the above documentation by calling Z-Code Software Corp. in San Rafael, California at +1 415-499-8649.

Conventions Used in this Manual

Throughout this manual, the following typographical conventions are used.

Text in *italics* is used for filenames, Z-Mail terms when they are introduced, cross-references, and emphasis. Examples:

Your configuration is stored in *~/.zmailrc* by default.

Messages are delivered to your *system folder*.

The **write** command does *not* create folders, since headers are not included in the written messages.

Text in **narrow bold** is used for variable names, commands, and Z-Script statements. Optional parts of Z-Script statements are enclosed in braces. Examples:

The **autoformat** variable...

The **mail** command...

If you start Z-Mail with the command **zmail –lite**...

The following is a Z-Script function:

```
function checkit() {
        if $1 == "done"
             builtin update
        endif
}
```

The command **button [–n]** creates a button; if **–n** is specified, the button does not require a message list as input.

Text in ***narrow italic bold*** is used for argument placeholders. Optional arguments are enclosed in braces. Examples:

The command **page** ***filename*** displays the contents of ***filename*** in a pager.

The command **source [*filename*]** reads the *~/.zmailrc* initialization file, or ***filename*** if specified.

Text in narrow font is used for Z-Mail output and the value of variables. Examples:

The **verify** variable may contain the quit value.

If no mail has arrived, Z-Mail will display No new mail...

Text in **bold** is used in keystroke names. Example:

Type **Enter** to accept the default.

Text in *ITALIC CAPITALS* is used for environment variable names. Examples:

If *TERM* is unset, Z-Mail Lite cannot run properly.

The **shell** Z-Script variable takes its initial value from the *SHELL* environment variable.

The percent (%) character is used at the beginning of a line to introduce a command that is issued at the UNIX shell prompt. Example:

% **zmail –motif**

Words enclosed in curly braces (**{ }**) and separated by vertical bars (**|**) are sets of choices. One word of the set of choices may be used. In the following example, **–default yes**, **–default no**, and
–default cancel are all valid options:

If you use **ask –default { yes | no | cancel }**, the default you specify will be highlighted when the dialog appears.

Where You Should Start

Motif users

If you use the Motif version of Z-Mail, you should first read the *Z-Mail User Guide*. It discusses how you can make the most of Z-Mail's powerful mail management capabilities using its point-and-click interfaces. After that, if you intend to write scripts or customize your Z-Mail installation, you may want to read various sections of the *Z-Mail Reference Manual* as the need arises.

Z-Mail Lite users

Z-Mail Lite is a highly configurable character-terminal version of Z-Mail that is designed to mimic the Motif GUI as closely as possible. Lite users can take the same path through the documentation as GUI users due to this similarity, substituting the *Z-Mail Lite User Guide* for the *Z-Mail User Guide*. Since Z-Mail Lite is a relatively new product, you may find that in some cases the on-line help is more up-to-date than the information contained in this Manual.

Fullscreen users

The fullscreen interface, supported in versions 2.1 and earlier, has been replaced by Z-Mail Lite, which is more flexible and easier to use. If you are a fullscreen interface user, contact Z-Code's Sales Department at 415-499-8649 or info@z-code.com for information on upgrading to Lite.

Command-line users

If you are an experienced mail user and intend to use the command-line version of Z-Mail, you should read the beginning of this manual first. After you understand basic Z-Mail concepts, you should read *Using Z-Mail's Command-Line Interface* on page 71, and then *Z-Script Commands* on page 109. You can use almost any Z-Script command at Z-Mail's command-line prompt. For information about command-line macros, see *Using Macros* on page 102.

Alternative to *mail* command

You can also use Z-Mail as a UNIX command to send mail directly from your UNIX shell. You do this by specifying options — command-line modifiers to the **zmail** command — that contain all the information Z-Mail needs to send your message. When you use Z-Mail in this send-only mode, it does not start one of its interfaces. Instead, Z-Mail returns control back to the UNIX shell or the program that called Z-Mail. For more information on Z-Mail's send-only capabilities, see *Z-Mail Options Summary* beginning on page 36, and *Z-Mail Options Reference* beginning on page 40.

CHAPTER 2

Z-Mail Basics

Electronic mail, or *e-mail*, is a form of personal and group communication used extensively by computing communities. Instead of sending paper letters through the mail, users send electronic messages via a network that interconnects users' computers. You create messages on your workstation or terminal and send them to other users.

As e-mail users have increased in sophistication, they have demanded more from their e-mail systems. Today, users not only send messages, they also routinely send programs and documents as attachments to mail messages, just as you would enclose a brochure or floppy disk in a business letter.

Today's electronic networks allow users all over the world to communicate and exchange electronic documents. As the number of users and messages skyrockets, so does the need to manage this electronic information.

Z-Mail is designed to handle these demanding needs and more. To properly use all of Z-Mail's capabilities, it is helpful to understand some basic terminology and conceptual building blocks that are used throughout Z-Mail.

E-Mail Terminology

E-Mail Addresses

All e-mail users have a unique *e-mail address* that identifies their name and the computer on which they read mail. This address is all you need to send a message to another user. A typical e-mail address for John Smith at the Justin Case Company is *jsmith@justin.com.* This address is in *Internet domain* or *RFC822* format.

The string *justin.com* identifies a computer at the Justin Case Company. The *.com* suffix indicates that Justin Case is a commercial organization. Other suffixes are described in *About address suffixes* on page 31.

Your own e-mail address is assigned to you by your system administrator. It is usually your userid followed by @ and the address of the computer where you receive e-mail.

Mail Transport Agent

The operating system of your UNIX computer has a *mail transport agent* (MTA) that functions like your local post office in that it picks up mail you send and delivers mail that others have sent to you. In addition, your computer may be connected to a worldwide network of computers such as the *Internet* that functions like a postal service.

To send a message to a user on another computer, you simply address the message and your computer delivers it to the proper remote computer over the network. The remote computer then delivers it to the addressee.

Mail User Agent

The program you use to read and compose e-mail is called a *mail user agent* (MUA). Z-Mail is a mail user agent that not only allows you to read and send messages, but also provides you with extensive capabilities to manage your mail. Z-Mail is *not* an MTA; it does not transport mail. When you compose and send a message in Z-Mail, Z-Mail hands it off to your computer's MTA for delivery.

E-mail messages

The basic unit of e-mail communication is the *message*. The format of e-mail messages that Z-Mail and nearly every UNIX MTA uses is a worldwide standard known as *RFC822*, with enhancements for multimedia *attachments* known as *MIME* (the Multimedia Internet Mail Extensions). Nonstandard e-mail systems may require a special *gateway* that can translate between RFC822 and the proprietary format to be usable with UNIX systems and Z-Mail. If you need a gateway to exchange e-mail with a non-RFC822 compliant system, contact the system's vendor.

RFC822 format

An RFC822 e-mail message is made up of several *headers*, a *body* and an optional *signature*. Z-Mail can also include *attachments* in a message. Here is a sample RFC822 compliant message that contains two attachments; normally, Z-Mail hides the attachment information to make the message easier to read. Text in **bold** is what you actually see in the Z-Mail message reading window. The various parts of the message are described below.

Headers

```
Received: from mickey.philo.com by zen via SMTP (920330.SGI/920502.SGI)
    for steve id AA20197; Tue, 23 Mar 93 10:16:09 -0800
Received: from zmips by philo.com (4.1/NBN-16/ZC-7)
    id AA18829; Tue, 23 Mar 93 10:21:31 PST
Received: from zen (zen.ARPA) by zmips (4.0/1.34)
    id AA04798; Tue, 23 Mar 93 10:16:01 PST
Received: by zen (920330.SGI/920502.SGI)
    for jane@zen.philo.com id AA20194; Tue, 23 Mar 93 10:16:01 -0800
From: steve (Steve Webster)
Message-Id: <9303231016.ZM20189@zen.philo.com>
Date: Tue, 23 Mar 1993 10:15:58 -0800
Reply-To: steve@philo.com
X-Mailer: Z-Mail_Lite (2.1b.1516 16mar93)
To: jane@zmips
Subject: An example RFC822-compliant message
Encoding: 1 TEXT BOUNDARY, 21 MESSAGE, 1 TEXT BOUNDARY, 570 MESSAGE
Content-Type: X-Zm-multipart
Content-Lines: 593
X-Zm-Content-Type: multipart
X-Zm-Content-Length: 44783
Status: OR
```

Body

```
----------
Encoding: 14 TEXT
X-Sun-Content-Lines: 14
X-Sun-Data-Type: Text
X-Zm-Content-Type: Text
X-Zm-Content-Length: 234
X-Zm-Data-Type: Text

This is the beginning of the message body. Note that the body is
```

```
separated from the headers by a blank line. Also note that the message
body is itself an attachment.

Here's another attachment; this one is a compressed, encoded PostScript
file:

----------
Encoding: 560 X-Zm-compress
X-Zm-Content-Name: SampleDesc.ps
X-Zm-Content-Type: text
X-Zm-Content-Length: 44124
X-Sun-Content-Lines: 560
X-Sun-Data-Type: X-Zm-postscript
X-Zm-Data-Type: postscript
X-Zm-Encoding-Algorithm: compress
X-Zm-Decoding-Hint: atob|uncompress

xbtoa Begin
+.\K>69pse+paSq@`l'Fi<"ac&g`:R.2.H])`]rETLe's:)Q'..%4QFK&[[a.3:s+""0/
"1hY1^abX
C39:%>DU)9'q7#W*G@[-6S0k?Z66VjS4B]\9E1iM/[XPmUm#_
...data continues for several hundred lines.

and here's the signature:

-steve webster
 Philomotion CEL Services
 Baton Rouge, LA
```

Signature

Headers

The *headers* are formatted fields with special names such as *To*, *From*, and *Subject*. Headers contain information about who the message is from, who it is for, what it is about, how it was created, how it is to be delivered, and more. *Headers* always appear at the beginning of a message, and are separated from the *body* by a blank line. Z-Mail allows you create and use your own custom headers.

Body and signature

The *body* contains the text of the message. The *signature* is an optional piece of text that a mail user agent can place on the end of the message, in effect "signing" it with the author's name, title, or other information.

Attachments

When you include an optional *attachment* with a Z-Mail message, the attachment is embedded in the message body, surrounded by special codes that the Z-Mail program notices and omits from the recipient's message display. When the recipient reads the message, the recipient is notified by Z-Mail that the message they're reading has one or more attachments, which the recipient can detach or display.

If you send an attachment to a recipient who does not use Z-Mail, the *X-Zm-Decoding-Hint:* line of the attachment data will help the recipient extract the original attachment file from the message. See the above message for an example.

MIME

MIME stands for Multipurpose Internet Mail Extensions, and is a commonly-used mechanism for sending and receiving multimedia Internet mail. Z-Mail is a MIME-compliant mail program, and uses MIME mechanisms for sending attachments, which may include images, sounds or non-ASCII text. Because Z-Mail is MIME compliant, you are able to exchange attachments with users of MIME-compliant mail programs at other sites.

Mail Folders

UNIX mail messages are stored in specially-formatted files called *folders*. Typically, each message in a folder is appended to the previous message, separated by a specially-formatted *From* line. UNIX delivers messages to your e-mail address by writing them to your *system folder* where they stay until you start Z-Mail.

See the *Folder Manager* dialog in the *Z-Mail User Guide* for instructions on creating and deleting folders in the GUI and Lite interfaces.

Compatible formats

Note that some UNIX MUAs use different folder formats. Z-Mail 2.1 is compatible with the following kinds of folders:

- *V7* format, described above, which is used natively by Z-Mail, *mail*, *Mail*, *mailx*, *xmail*, and other MUAs based on BSD and System V mail.
- *MMDF* format, which is similar to V7 format with the inclusion of four `^A` (control-A, ASCII 1) characters and a newline before and after each message in the folder.

Z-Mail can be configured to create new folders in MMDF format by appropriately setting the **folder_type** variable.

- *MH* format, in which each message is a separate file whose name is the same as the message's number in the folder. Z-Mail can read, but not write, this type of folder.

Incompatible formats

Z-Mail 2.1 is not directly compatible with these kinds of folders:

- Custom *MMDF* formats that don't use four ^A characters around each message in the folder. These can be made compatible by replacing the separators with four ^As.
- *AMS*, or *Andrew Message System*, folders. AMS is similar to MH in that messages are stored in separate files, except the files are named with a unique string. These can be made to look like MH messages by changing the filenames to numbers.
- *Rmail* format, which is similar to MMDF format with the addition of message-state information between messages.
- *Babyl* format, which is used by some species of *Emacs*-based mailers. Generally, these mailers can convert from their format to V7 format for use by Z-Mail.

If you encounter an incompatible type of folder not listed here that you want to read with Z-Mail, contact Z-Code Technical Support at **support@z-code.com** for assistance.

System folder

When you start Z-Mail, it automatically opens your *system folder* (which is where your mail is delivered by the MTA) and shows you the summaries of the messages it contains. When you are finished reading your messages, Z-Mail stores them in either your system folder or your *main mailbox* depending on how you've configured Z-Mail (see the **hold** variable on page 236). You can think of the system folder as the mailbox outside your house and the main mailbox as the in-box on the corner of your desk.

The messages stay in your system folder or your main mailbox until you **delete** them or **save** them to another folder you create. To see the messages in a particular folder, you **open** the folder.

Z-Mail permits you to have more than one folder open at a time. You can switch among opened folders to more easily manage your mail. Only one folder is *active* at a time. Most Z-Mail commands affect messages in the active folder. The messages in each folder are numbered so you can specify particular messages when you issue Z-Mail commands.

Z-Mail's User Interfaces

GUI

Z-Mail offersa Motif-based graphical user interfaces (GUI) for X Window System users. The GUI uses the point-and-click simplicity of a mouse-driven interface to improve your productivity. Z-Mail's GUI is started with one of these commands at the UNIX shell prompt:

zmail –motif
zmail –gui

There are two versions of the Z-Mail GUI program: Z-Mail Basic, and Z-Mail Pro. Z-Mail Pro has several additional features that are not available in Z-Mail Basic. To find out which version you are running, choose Product Info from the Help menu, or run the **version** command.

Lite

Z-Mail's Lite interface is designed to mimic the Motif interface for character terminal users; you use keyboard commands to move the terminal's cursor around the screen. To start Z-Mail Lite, enter this command at the UNIX shell prompt:

zmail –lite

Z-Mail Lite is not built into the Z-Mail GUI program; it is a separate product.

Command Line

Z-Mail also offers a command-line interface for managing your mail. Z-Mail supports mail commands you may already know from using the *mail* program supplied with most UNIX systems or *BSD Mail*. In addition, Z-Mail has many commands that go far beyond standard UNIX *mail* capabilities; see *Z-Script Commands* beginning on page 109. To start the command-line interface, enter this command at the UNIX prompt:

zmail

Send-Only Mode

You can also use Z-Mail as a mail utility in its send-only mode. This mode does not have an interactive user interface at all; you instead issue the **zmail** command at the shell prompt and specify a variety of options to send messages to other users. For more information, see *Z-Mail Options Summary* beginning on page 36. The following command would send a message with subject "*example*" to user *fgreene*, taking the message body from the file *text-file*.

```
zmail –subject "example" fgreene < text-file
```

Using Help

Z-Mail's help facility provides information in both a context-sensitive fashion and an index of specific help categories that you can browse.

Command Line

In the command-line interface, type **–?** after a command to receive help about the command. For example,

help –?

displays help about using the **help** command itself.

If you type **?** as a command, Z-Mail displays a list of valid commands.

GUI and Lite

In GUI and Lite interfaces, Z-Mail provides context-sensitive help. Menubars have a *Help* menu in the upper right corner. Motif and Lite dialog boxes have a *Help* button in the lower right corner. Selecting any Help item opens a new *Help* window that you can keep on your screen (in the GUI interface) as you work.

In the GUI interfaces, you can also select the *Click for Helpt* option from the *Help* menu. Z-Mail changes the mouse pointer from an arrow to a hand. Click on the item about which you want help and Z-Mail opens a *Help* window with information about that item.

Customizing the help files

You can customize Z-Mail's help files to suit the needs of your site. For example, you may wish to add entries that describe buttons or functions that you've added to the systemwide initialization files, or mention the telephone number of your Help Desk. The help files that Z-Mail uses by default are located in the Z-Mail library directory (usually /*usr/lib/Zmail*). The format of each file is described in the file itself.

- *command.hlp*, which contains the text displayed by the **?** command and in the *Help* dialog when you set the *Z-Script Commands* toggle. If you set the **cmd_help** variable to a different file, Z-Mail will use that file instead.
- *motif.hlp*, which contains the text displayed in the Help

dialog in the Motif interface. If you set the **gui_help** variable to a different file, Z-Mail will use that file instead.

- *openlook.hlp*, which contains the text displayed in the Help dialog in the Open Look interface. If you set the **gui_help** variable to a different file, Z-Mail will use that file instead.
- *lite.hlp*, which contains the text displayed in the Help dialog in the Lite interface. If you set the **gui_help** variable to a different file, Z-Mail will use that file instead.
- *variables*, which contains both the text displayed in the Variables dialog, and the default settings for many system variables. You may create new system variables by adding them to this file. For more information, see the *Note for System Administrators* on page 226.

Specifying Message Lists

Often, you perform an action on the current message(s) by entering a command. You tell Z-Mail explicitly which messages to process by specifying a *message list*. For example, one place where you can enter a message list is the *Messages* field of the Main Window in the GUI and Lite interfaces.

Z-Mail supports a special set of *metacharacters* that are shorthand notations for specifying complex lists of message numbers. When processing message lists, Z-Mail recognizes the following metacharacters:

*****	All messages
^	First message
$	Last message
. (period)	Current message
n–m	Range beginning with message number *n* and ending with *m*. *n* and *m* can be metacharacters.
{*messagelist*}	Do not include messages in *messagelist*

Z-Mail examines message lists from left to right. Follow these rules when specifying lists:

- You can use spaces or commas to separate arguments.
- You cannot begin a list with a negated, i.e. **{*messagelist*}**, argument. For example, **{10–20} 1–25** is invalid.
- You can reassert (un-negate) messages that were negated earlier in argument lists. For example, **^–90 {40–60} 53,55** is valid.
- Ranges must be ascending. For example, **1–7** is a valid range, but **5–2** is invalid.

EXAMPLES

The following table illustrates specifying message lists:

Message List:	***Refers to Messages:***
4–7	4 5 6 7
1, 3, 7–10 12	1 3 7 8 9 10 12
4–$	4 through last message
^–.	First message through current message
^,$	First and last message
*** {4–8}**	All messages except 4 through 8

1-20 {2–6} 5	1 5 7-20
∗ { . }	All messages except current one

Specifying Text Patterns

When you search the text of messages, you specify *text patterns*. These patterns are strings that contain literal characters and *metacharacters* that Z-Mail interprets. You can use the standard UNIX regular expression metacharacters as described in the *ed(1)* UNIX man page. These metacharacters include, but are not limited to:

.	Any single character except *newline***.**
*	Any number of repetitions of the previous character (including none).
^string	**Matches string at the beginning of lines.**
string$	**Matches string at the end of lines.**
\	Matches following character literally (overrides metacharacter meaning of following character).
[xyz]	Matches any one of the characters enclosed.
[a–z]	Matches any one of the characters in the ASCII range a–z, inclusive.

EXAMPLES

The following table illustrates the use of metacharacters in matching text patterns.

String:	***Matches:***	***But not:***
mark..	market, mark12	mark, 2mark
mark.*	market, mark, mark12	marek
.*mark$	unmark, remark	smarks, market
^mark	market	unmark, 2mark
mark[a–c]	marka, markb, markc	market, markd
mark[ac]	marka, markc	markb
abc*	ab, abc, abcc, abccc	abcd, abcabc
.abc	1abc, aabc	abc, abcd
g[abc]*	gabc, gabcabc	gdbc, ab

Specifying Filename Patterns

Directory Notations

Z-Mail recognizes the **~** (tilde) and **+** metacharacters at the beginning of filenames in all interfaces:

~	Expands to value of **home** variable.
~*user*	Expands to ***user***'s home directory.
+	Expands to value of **folder** variable.

By default, the **home** variable contains the pathname of your home directory, and the **folder** variable contains *~/Mail*.

You can specify another user's home directory with *~user*. Files in your home directory must be referenced as *~/filename*. However, the forward slash (**/**) character is optional when you refer to folders with **+**. That is, **+***filename* and **+***/filename* refer to the same *filename* in your **folder** directory.

EXAMPLES

The following table illustrates the use of **~** and **+** in filenames. Assuming that your home directory is */usr/acct/gnomon* and your folder directory is *~/Mail*,

if you type	it means
~	/usr/acct/gnomon
~/tract	/usr/acct/gnomon/tract
~tract/gyre	/usr/acct/tract/gyre
~/gyre~	/usr/acct/gnomon/gyre~
+	/usr/acct/gnomon/Mail
+/moon	/usr/acct/gnomon/Mail/moon
+sun+moon	/usr/acct/gnomon/Mail/sun+moon

Filename Completion

If you set the **complete** variable, Z-Mail expands metacharacters to all the files matching the pattern you specify when you press the completion character (**Esc** by default) or listing character (**Ctrl-D** by default). See the **complete** variable on page 265 for capabilities and limitations of this facility. These metacharacters are also recognized by the **foreach** command; see page 114.

Metacharacters that the completion listing facility and **foreach** recognize are:

+, ~	See above
?	Any single character except *newline*.

*	Any number of any characters including none.
\	Makes the following character literal.
[xyz]	Matches any one of the characters enclosed.
[!xyz]	Matches any one character not enclosed.
{abc,[g–r],...}	Matches any one of the patterns enclosed.

EXAMPLES

The following table illustrates the use of metacharacters in matching filename patterns.

String:	***Matches:***	***But not:***
pro??	proga, pro12	pro123, pro
pro?*	prog, prog12, protest	pro, apro
pro*	pro, prog, protest	apro
pro[a–c]	proa, prob, proc	pro, protest
pro[ac]	proa, proc	prob
pro	unpro, 2pro, protest	pre, april
?pro	1pro, 2pro, apro	unpro, prog
pro{ab,g,123}	proab, prog, pro123	pro, prob, apro

Specifying E-Mail Addresses

To send an e-mail message, you must properly address the message with its destination. This section describes a few basic ways to address messages you send to other users.

Note: If you use mail gateways at your site to allow otherwise incompatible mail systems to communicate, you may have more options for addressing your messages than are described here. Ask your system administrator for details.

Local Addresses

To send a message to users on the local machine, i.e., the machine on which Z-Mail is running, you only need to provide their usernames:

john pat kfong rgomez

If you use space characters to separate user addresses, Z-Mail automatically insets commas between the addresses:

john, pat, kfong, rgomez

You can prevent Z-Mail from inserting commas in addresses sent to the MTA by setting the **picky_mta** variable to omit_commas. See the **picky_mta** variable on page 299 for more information.

Domain Addresses

The most common form of e-mail addresses are *domain addresses* — also known as *Arpanet, Internet domain*, or *RFC822* addresses.

SYNTAX

The general form of a domain address is:

user@host.domain

EXAMPLE

To send a message to Kyle Dwyer at Ukiah Fiber Company, you might enter the address *kdwyer@ukiahfib.com*. The @ character separates the username, *kdwyer*, from the domain name, *ukiahfib.com*.

Note: You may also encounter addresses containing % characters which separate user and domain names. Some, but not all, hosts can understand this notation. Most hosts, however, recognize the @ notation, so it is safer to use instead of the % character.

UUCP Addresses

When sending e-mail to a user whose computer is connected to your computer via UUCP, you may need to route the message through a series of host computers. To route your message properly, the mail transport agents on those computers look for its *UUCP address* (also known as a "bang path").

A UUCP address describes the various hosts through which your message will be routed to reach its destination.

SYNTAX

A UUCP address may indicate as many hosts as necessary to route a message to the correct user.

host1!host2! ... !hostn!user

EXAMPLE

The following UUCP address sends your message from your host computer to the host named *utech*, which sends it to *amcorp*, which in turn sends your message to *eastsoft*. The recipient, *psteele*, reads the message on the *eastsoft* host computer.

utech!amcorp!eastsoft!psteele

Address Conventions

Z-Mail understands e-mail addresses that contain a *comment field*. Comment fields do not affect the destination address of mail. These fields are purely for legibility and may be used under the following constraints:

- Anything inside parentheses is a comment:
  dpowers@ukiahfib.com (Dan Powers)
- Anything within angle brackets (**< >**) is an address; whatever is outside of the address is considered a comment. Some MTAs will complain about anything that follows the **>** character in the address unless what follows is enclosed in parentheses.
  Dan Powers <utech!ukiahfib!dpowers> (El Dan)
  Dan Powers <dpowers@ukiahfib.com>
- Text inside double quotes (**" "**) is treated as a comment by some MTAs; whatever is inside the double quotes is not used as part of the address.
  "Dan Powers" utech!ukiahfib!dpowers

"Dan Powers" dpowers@ukiahfib.com

Most MTAs, however, will consider the above syntax an error. If you need to specify a double-quoted element of an address, place the entire address, including the quoted portion, inside angle brackets. For example
(Dan Powers) <"Dan Powers"@ukiahfib.com>

- If the comment contains unbalanced double quotes, you may get unpredictable results, and your MTA may not deliver the mail.
- If the comment contains a comma, you must use either parentheses or double quotes, or Z-Mail will interpret the comma as the start of a new address. In the following two examples, the first is correct usage, while in the second there are three addresses separated by a comma and a space respectively:
  mary@finance.justin.com (Mary Edwards, VP Finance) CORRECT
  <mary@finance.justin.com> Mary Edwards, VP Finance WRONG
- Angle brackets have the highest precedence and may be used in conjunction with double quotes and parentheses:
  The Boss (Tom Justin) <tom>
  Frank "Frankie" Jones <fjones@justin.com>
- You may enter multiple addresses on a line:
  mary, fjones@justin.com, dpowers@ukiahfib.com

Note: Since mail transport agents are case-insensitive, you may specify e-mail addresses in any mixture of upper- and lower-case letters. For example, *mary@justin.com* is equivalent to *Mary@Justin.COM* for addressing purposes.

About address suffixes

Educational institutions have domain addresses that end in *.edu*, such as *cmu.edu*. Non-profit organizations typically end in *.org*, U.S. government sites end in *.gov*, U.S. military sites end in *.mil*, commercial organizations end in *.com*, and non-U.S. sites end in a country code, such as *.ca* for Canada.

For more information on network addressing, see your system manager or O'Reilly & Associates' text, *!%@:: A Directory of Electronic Mail Addressing & Networks*.

Specifying Alias Addresses

An *alias* is an abbreviation of an e-mail address or group of addresses. You can define mail aliases using the **alias** command on page 112. For example, the following **alias** commands define aliases:

alias john John Talbott <john@justin.com>
alias mary Mary Edwards <mary@justin.com>
alias sales john, mary

The aliases john and mary refer to single users. The alias sales refers to the aliases john and mary. Once you have defined these three aliases, you can address mail to john, mary, or sales, as well as by entering their real e-mail addresses. Note, however that you cannot specify an alias and a comment on the same address line:

To: "Sales Department" <all>

Z-Mail does not retain the comment *Sales Department* when it expands the alias *all*. Z-Mail replaces the entire address, including comment, with the expansion of the alias. To prevent alias expansion in addresses, see the **no_expand** variable on page 296.

See the **alias** command on page 112 for more information about defining aliases.

CHAPTER 3

Starting and Initializing Z-Mail

You start Z-Mail with the **zmail** command at a shell prompt. The **zmail** command has many options that allow you to configure your Z-Mail session and choose which of Z-Mail's interfaces you wish to use.

Most users who prefer the Motif or Lite interfaces use Z-Mail menus to perform the various functions offered by the numerous **zmail** command options.

Starting Z-Mail

You start Z-Mail by typing the **zmail** command at your UNIX shell prompt. You specify the interface you wish to use with a command option.

SYNTAX

The general syntax of the **zmail** command is a series of options followed by arguments:

zmail [*option1* [*arg_list1*]] [*option2* [*arg_list2*]] *. . .*
[*script–args*]

The **zmail** command options are described in *Z-Mail Options Summary* beginning on page 36 and *Z-Mail Options Reference* beginning on page 40.

Arguments ***script–args*** to the **zmail** command that follow the last valid **zmail** option are passed into the Z-Script initialization scripts as arguments. They are available in initialization scripts via the function variables **$*** (see page 317) and **$1, $2...** (see page 318).

GUI

To start a Z-Mail graphical user interface, enter one of the following commands:

zmail –motif [*options*] [*arguments*]
zmail –gui [*options*] [*arguments*]

Lite

To start the Z-Mail Lite character interface:

zmail –lite [*options*] [*arguments*]

Command Line

To start the Z-Mail command-line interface:

zmail –shell [*options*] [*arguments*]

Your system administrator may also have installed a "stripped-down" command-line-only version of the Z-Mail program that does not include the GUI interface, so it loads somewhat faster. To run the command-line only version of Z-Mail, use:

zmail.small –shell [*options*] [*arguments*]

Send-Only

You can also use the **zmail** command at the UNIX shell prompt to send mail just like the UNIX *mail* command. Using the **zmail.small** program, rather than **zmail**, is recommended due to **zmail.small**'s reduced size and faster loading. If **zmail.small** is available on your system, replace **zmail** in the examples below with **zmail.small**.

SYNTAX

The general syntax for sending messages from the UNIX shell command line is:

zmail [–subject "*subject*"] [–attach [*program*:]*filename*] [–cc *cc_list*] [–bcc *bcc_list*] [to_list] [< *textfile*]

The syntax for editing and sending an existing message file from the UNIX shell command line is:

zmail [–send[!]] –draft *draftfile*

Z-Mail Options Summary

This section summarizes the numerous options to the **zmail** command by category:

- User Interface Options
- Initialization and Startup Options
- Options for Sending Mail
- Options for Managing Folders
- Miscellaneous Options

For details about a specific option to the **zmail** command, see *Z-Mail Options Reference* beginning on page 40.

User Interface Options

-gui Starts the Motif GUI, depending upon which is present in the **zmail** program.

–lite Starts Z-Mail's Lite character interface, if installed.

–motif Starts Z-Mail's Motif GUI, if installed.

–shell or ***–shell!*** Forces Z-Mail to start the command-line interface even if you have no mail in the initial folder.

–interact Forces Z-Mail to start an interface even if input is redirected.

Initialization and Startup Options

–acceptFolders Tells Z-Mail to be a folder server, enabling other Z-Mail processes to hand off "open" commands to it.

–echo Tells Z-Mail to process input only after you type **Enter** in the command-line interface.

–eval After the initial folder is loaded, executes a Z-Script command you specify.

–headers Previews the contents of a folder by displaying header summaries of its messages.

–iconGeometry Specifies the size and location of your Z-Mail Main Window icon. For Motif only.

–init Loads an initialization file you specify before other initialization files.

–library Specifies the location of the Z-Mail library.

–mailbox Specifies the location of your system folder (spool file).

–noheaders Suppresses header summary information display at startup.

–noinit Suppresses loading of initialization files.

–offerFolders Hand off folders to a folder server.

–picky Prevents Z-Mail from supplying *From:* and *Date:* headers to the mail transport agent.

–refuseFolders Do not become a folder server. This option overrides the MediaMail default and any set X resources.

–source *Sources* (reads and processes) commands from a file you specify.

–timeout Specifies how often, in seconds, you want Z-Mail to check for new mail in the GUI interface.

Options for Sending Mail

–attach Attaches a file to an outgoing message.

–blind Sends a blind carbon copy of the message to a list of users you specify.

–carbon Sends a carbon copy of the message to a list of users you specify.

–direct Passes all redirected input to the MTA.

–draft Uses the specified text file as the body of the message.

–send Immediately sends a message with the included draft file.

–subject Sets the *Subject* header of the message.

–template Uses the specified text file as a template for the message.

–verbose Passes the **verbose** option to the mail transport agent.

Options for Managing Folders

–filter Tells Z-Mail to run message filters at startup.

–folder Opens the specified folder at startup.

–nofilter Tells Z-Mail to ignore **–filter** options that appear earlier on the command line.

–readonly Prevents modifications to the initial folder.

–user Sets the user name for this Z-Mail session to a name you specify.

Miscellaneous Options

–debug	Activates verbose debugging output.
–help	Displays help text on the various options to the **zmail** command.
–register	Registers the Z-Mail activation key and licensed users.
–version	Displays Z-Mail version information and exits.
–zcnlserv	Specifies a Network License Server (NLS) from which Z-Mail will obtain a license.

Z-Mail Options Reference

This section explains the various options to the **zmail** command. Options are listed in alphabetical order.

–acceptFolders

zmail command option *GUI ONLY*

SYNTAX **–acceptFolders**

The **–acceptFolders** option tells Z-Mail to be a folder server, enabling other Z-Mail processes to hand off "open" commands to it. This can only be used in combination with the **-gui** option. Only one folder server can be active per display. If there is already a server on the current display, the **–acceptFolders** option is ignored.

–attach

zmail command option *SEND-ONLY MODE*

SYNTAX **–attach [*type*:]*filename***

SYNONYM **–A**

The **–attach** option adds an attachment to the outgoing message. Z-Mail will prompt you for information about the attachment, including its ***type*** (if not supplied on the command line), a short description, and the encoding method.

You may optionally precede ***filename*** with its ***type*** and a colon, in which case Z-Mail will not prompt you for ***type***.

If ***type*** is not specified, Z-Mail will examine the first few thousand bytes of ***filename*** for 8-bit characters to determine whether ***filename*** is text or binary, and set the default for ***type*** to text or binary accordingly.

If input is redirected, Z-Mail will not prompt for ***type*** or a description.

If you are unsure whether a file is binary or text, specify its ***type*** as binary, since you must encode binary files, and it never hurts to encode text files. Note, however, that encoded text files are slightly larger than unencoded text files.

To attach multiple files to the outgoing message, you must use a different command (all on one line):

zmail –noheaders –shell
–eval! 'mail –attach *file1* –attach *file2* –attach *file3*...'

For additional information on sending attachments, see *Defining Attachment Types* beginning on page 328 and the **detach** command on page 135. Also see the *Z-Mail User Guide*.

You cannot specify **–attach** if you also specify **–gui**, **–motif**, or **–lite**.

EXAMPLE

The following command instructs Z-Mail to send a message with an attachment called *reserve.form* to *robert*. Z-Mail prompts you for the additional information it needs to send the attachment.

```
% zmail –A reserve.form robert
Attachment type (? for list): text
Attachment description: Reservation form
Encoding (? for list) : ?
compress btoa uuencode
Encoding (? for list) : none
To: robert
. . . etc.
```

–blind

zmail command option *SEND-ONLY MODE*

SYNTAX **–blind *bcc_list***

SYNONYMS **–blindcarbon, –bcc, –b**

To send a “blind carbon copy” of a message to a list of users without having their names appear in the message’s headers, specify the **–blind** option. If you specify more than one e-mail address or an address containing spaces in ***bcc_list***, you must enclose the entire list in quotation marks.

You cannot specify **–blind** if you specify the **–gui**, **–motif**, or **–lite** option.

EXAMPLE

The following command starts a message addressed to *robert* with blind carbon copies to *keri* and *marc*. Z-Mail then prompts for a subject and waits for you to enter the text of the message.

% **zmail –bcc "keri marc" robert**
To: robert
Subject: **Design Meeting at 10:30**
Bcc: keri, marc
. . . etc.

When *robert* receives this message, it will contain no indication that *keri* and *marc* also received a copy.

–carbon

zmail command option *SEND-ONLY MODE*

SYNTAX **–carbon *cc_list***

SYNONYMS **–copy, –cc, –c**

To send a "carbon copy" of a message to a list of users, specify the **–carbon** option. The list of names in ***cc_list*** appears in the *Cc:* header of the message. If you specify more than one address or an address containing spaces, you must enclose the entire list in quotation marks.

You cannot specify **–carbon** if you specify the **–gui**, **–motif**, **-lite** option.

EXAMPLE The following command starts a message addressed to *danielle* with carbon copies for *jon* and *terry,* and a subject of *Architectural Design Notes*. Z-Mail then waits for you to enter the text of the message.

% **zmail –carbon "jon terry" danielle**
To: danielle
Subject: **Architectural Design Notes**
Cc: jon, terry
. . . etc.

–debug

zmail command option

SYNTAX **–debug**

SYNONYM **–d**

RELATED COMMAND **debug**

RELATED VARIABLE *DBFILE* (environment variable)

If the **–debug** option is specified, Z-Mail displays a level 2 debugging trace. The **–debug** option is equivalent to the Z-Script command:

debug 2

See the **debug** command on page 133 for more information.

–direct

zmail command option *SEND-ONLY MODE*

SYNTAX **–direct**

SYNONYMS **–notempfile, –D**

If you are sending a file as a message and want Z-Mail to pass its standard input directly to the mail transport agent (MTA), specify the **–direct** option. Z-Mail passes the data on its standard input without buffering it in a temporary file. Z-Mail ignores this option unless you redirect input from a file or a UNIX shell pipeline. The **–direct** option is intended for sending large text files when storage space in the file system is at a premium.

You must supply addressing information on the command line because **–direct** is noninteractive.

You cannot specify **–direct** if you specify the **–gui**, **–motif**, **–lite** option.

EXAMPLE The following command sends the file *rollup.data* to *joe* with a subject of *Consolidated data*.

```
% zmail –subject "Consolidated data" –direct joe < rollup.data
```

When *joe* receives this message, the data that was in the file *rollup.data* will appear as the body of the message.

–draft

zmail command option *SEND-ONLY MODE*

SYNTAX **–draft *draft_file***

SYNONYMS **–headerfile, –h, –message**

To send a previously prepared message file (a *draft*), specify the **–draft** option. The ***draft_file*** may be an unsent message generated by an earlier Z-Mail session.

The ***draft_file*** must contain all message headers needed to send the message correctly. Z-Mail adds only a new *Date:* and a *From:* header if there is none present. The file must contain at least a *To:* header for Z-Mail to accept it as a message draft.

The headers in ***draft_file*** override headers specified on the command line.

If you specify the **–** character as the ***draft_file*** name, Z-Mail reads the standard input for the draft text.

If you specify **–draft** and **–direct**, the **–direct** option is ignored.

You can only specify the **–draft** option in send-only mode. You cannot use **–draft** if you specify the **–gui**, **–motif**, or **–lite** option.

EXAMPLE

The following command sends the file *directions* to *linda* after waiting for you to finish editing the text of the message.

```
% zmail –draft directions linda
27 lines
(continue editing or ^D to send)
```

The next command uses the **–send** option to send the draft file without waiting for further input from you.

```
% zmail –draft directions –send linda
27 lines
```

–eval

zmail command option

SYNTAX

–eval *zcommand*

SYNONYMS

–execute, –e, –command

To have Z-Mail execute a command just after it loads the initial folder, specify the **–eval** option.

After loading all initialization files, including any you specify with the **–source** option, Z-Mail executes ***zcommand***. If ***zcommand*** contains spaces, you must enclose the entire command in quotation marks. If ***zcommand*** contains quotation marks, you may have to escape them by preceding them with a backslash (****).

If you specify several **–eval** options, Z-Mail executes each in the order in which they appear in your **zmail** command.

Z-Mail ignores the **–eval** option when you use the send-only mode of Z-Mail, unless you specify the **–shell** option.

EXAMPLE

The following command starts the Motif interface and opens the *Compose* window so you can send a mail message.

% **zmail –gui –eval "dialog compose"**

–echo

zmail command option *COMMAND-LINE ONLY*

SYNTAX **–echo**

SYNONYM **–E**

If you specify the **–echo** option, Z-Mail sets local echo so it remains on. This tells Z-Mail to only process input after you type **Enter**. Normally, Z-Mail runs with local echo off and each character you type is processed individually. This allows Z-Mail to handle macros, keyboard mappings, and filename completion. Typically, keyboard management is transparent to you, and the use of this option is usually not necessary. If you specify the **–echo** option, key sequences you set with the **map** or **map!** commands do not take effect, nor will filename completion work.

Z-Mail ignores this option in the Lite and GUI interfaces.

–filter

zmail command option

SYNTAX **–filter**

To tell Z-Mail to run message filters as it loads the initial folder, specify the **–filter** option. This option also allows Z-Mail to filter incoming mail before it is delivered to your system folder (see the **–folder** option below). You define a filter by using the **filter** command in an initialization file. If your initialization files do not define any filters, the **–filter** option has no effect.

During the loading of the initial folder, Z-Mail does not execute filters defined with the command **filter –n**, even if you specify the **–filter** option.

This option is used most frequently in conjunction with **–folder**, such as when you automatically run Z-Mail from a

.forward or *.maildelivery* file. See the **–receive** option on page 54 for more information.

If you specify both the **–filter** and **–nofilter** options, Z-Mail only recognizes the last one present on the command line.

–folder

zmail command option

SYNTAX

–folder [*foldername*]

SYNONYM

–f

To make a folder ***foldername*** the initial folder instead of your system folder, specify the **–folder** option. If you do not specify ***foldername***, you must place the **–folder** option as the last argument on the **zmail** command line. When you do, Z-Mail opens the folder specified by the **mbox** variable, *~/mbox* by default.

If ***foldername*** is a single hyphen ("–"), Z-Mail receives messages on its standard input. You can take advantage of this to run Z-Mail automatically from delivery files such as *.forward* and *.maildelivery*. Instead of starting one of its interfaces, Z-Mail performs the actions you specify with the **–filter**, **–source**, and **–eval** options, and then appends all messages not marked for deletion to your system folder.

Note that **zmail –filter –folder –** is equivalent to **zmail –receive**. See the **–receive** option on page 54 for more information.

WARNING: When you specify ***foldername*** as a single hyphen ("–"), any action that causes Z-Mail to exit prematurely — whether in initialization files, in a file read by the **–source** option, or in commands you execute with the **–eval** option — prevents Z-Mail from appending messages to your system folder. In this case, Z-Mail leaves your messages in a temporary file, subject to automatic deletion by the operating system at reboot or after a certain period of time. For this reason, novice users should avoid the **–source** and **–eval** options when using
–folder –. Instead, they should use the **–filter** option, which always places incoming messages in the system folder if they are not diverted to another folder.

EXAMPLES

The following command starts the Motif GUI, opens the folder *+surveys*, and makes it the active folder.

% **zmail –gui –folder +surveys**

–gui

zmail command option *GUI ONLY*

SYNTAX **–gui**

SYNONYMS **–motif**

To start Z-Mail's Motif interface, use the **–motif** or **–gui** option. This option should come before any other X Window System or Z-Mail options.

The **–gui** option is equivalent to **–motif**.

EXAMPLE The following command starts Z-Mail's graphical user interface:

% **zmail –gui**

–headers

zmail command option

SYNTAX **–headers[:*c*]**

SYNONYM **–H**

To preview the contents of a folder without starting a Z-Mail interface, specify the **–headers** option. Z-Mail displays the message headers for all or a subset of the messages in your system folder, or another folder you specify with the **–folder** option. To identify the messages for which you want to display headers, specify one of the following header modifiers as the ***:c*** argument:

a	All messages
d	Deleted messages
f	Forwarded messages
n	New messages
o	Old messages
p	Preserved messages
r	Replied to messages
s	Saved messages
u	Unread messages

If you do not specify ***:c***, Z-Mail displays a list of headers for all the messages in the active folder (as if you specified **H:a**).

The format of the information Z-Mail displays in the list of headers depends on the value of the **hdr_format** variable. For more information, see the **summary_fmt** variable on page 309.

Z-Mail ignores the **–headers** option if you specify the **–gui**, **–motif**, or **–lite** option. Otherwise, the **–headers** option prevents Z-Mail from starting an interface, overriding the **–shell** option.

EXAMPLE

Assume your system folder has three new messages in it and your initialization file *~/.zmailrc* sets the value of **hdr_format** with this command:

```
set hdr_format = "%15n %M %–2N %7T %20s"
```

If you enter the following command, Z-Mail lists message headers for the three new messages in your system folder:

```
% zmail –H:n
51   N   Sean O'Brien    Jul 15   8:49am    Meeting at 11:30
52   N   Jon Yost        Jul 16   12:10pm   Porting standards
53   N   Ian Trainor     Jul 16   1:05pm    Maintenance
```

–help

zmail command option *SEND-ONLY MODE*

SYNTAX **–help**

SYNONYM **–?**

If you specify the **–help** option, Z-Mail displays the general syntax of the **zmail** command as well as a list of all options to the **zmail** command. It displays a subset of the information covered in this section of this manual.

–iconGeometry

zmail command option *GUI ONLY*

SYNTAX **–iconGeometry *location***

Specify the location of Z-Mail icon when the Main Window is iconified. ***location*** is specified according to the geometric specification:

{+-}*<xoffset>*{+-}*<yoffset>*

<xoffset> and ***<yoffset>*** are integers representing number of pixels. The sign of ***<xoffset>*** and ***<yoffset>*** indicate:

+*<xoffset>*	Measuring starts from *top* of display
-*<xoffset>*	Measuring starts from *bottom* of display
+*<yoffset>*	Measuring starts from *left side* of display
-*<yoffset>*	Measuring starts from *right side* of display

For example, the following:

-iconGeometry +10+10

indicates to place the icon ten pixels from the top of the display and ten pixels from the left.

–init

zmail command option

SYNTAX **–init[!] *filename***

SYNONYM **–I** (upper-case i)

To force Z-Mail to process a specific initialization file before it performs any other initialization, specify the **–init** option. Z-Mail reads and processes the file you specify as ***filename*** before the default system initialization file.

Only one **–init** option and ***filename*** may be specified per **zmail** command.

The optional exclamation point (**!**) modifier prevents Z-Mail from processing *system.zmailrc*, the default system initialization file, so you can use the **–init!** option to specify a substitute default file. When you specify **–init!**, Z-Mail still processes your personal initialization file normally. For more information, see *Initializing Z-Mail* on page 63.

EXAMPLES The following command starts Z-Mail's command-line interface, processing the initialization file called *new.zmailrc* before processing Z-Mail's default initialization files.

% **zmail –init new.zmailrc**

The following command starts Z-Mail's Lite interface, processing the initialization file called *test.zmailrc* instead of the default file, *system.zmailrc*.

% **zmail –lite –init! test.zmailrc**

–interact

zmail command option

SYNTAX **–interact**

SYNONYM **–i**

To force Z-Mail to start an interface even if input is redirected, specify the **–interact** option. This option can be used in combination with **–gui**, **–motif**, **–lite**, or **–shell**. If you don't specify an interface, the command-line interface is started.

EXAMPLE The **–interact** option forces Z-Mail to start an interactive interface when you are running the **zmail** command on another ***host*** from a remote-shell utility, such as *rsh*. For example, the command

```
% rsh host zmail –shell
You can't redirect input unless you're sending mail.
If you want to run a shell with redirection, use "–i"
```

will fail with the indicated message because Z-Mail's input is redirected by the *rsh* command. Instead, add the **–interact** option:

```
% rsh host zmail –shell –interact
Z-Mail (2.1.3 26jan93): Type '?' for help.
(20:16) [no folder]: #0 of 0>
```

Z-Mail automatically assumes **–echo** if you specify **–interact**.

It also allows you to redirect input from a script of Z-Script commands. For information on writing scripts, see *Initializing Z-Mail* on page 63 and *Z-Script Commands* beginning on page 109. Note that this option differs from the **–i** (ignore) option of BSD *Mail*.

–library

zmail command option

SYNTAX **–library *librarydir***

SYNONYM **–L, –lib**

If you specify the **–library** option, Z-Mail searches the directory ***librarydir*** for the Z-Mail program library. If Z-Mail is unable to find the directory you specify, it searches the directory specified in the value of the *ZMLIB* variable, as well as the default directory */usr/lib/Zmail*. If Z-Mail is unable to find the library in any of these directories, Z-Mail cannot run.

–lite

zmail command option *LITE ONLY*

SYNTAX **–lite**

To start Z-Mail's Lite interface, specify the **–lite** option. You should place this option first on the command line.

EXAMPLE The following command starts Z-Mail's Lite interface:

% **zmail –lite**

–mailbox

zmail command option

SYNTAX **–mailbox *mailboxpath***

SYNONYM **–m**

To force Z-Mail to use a specific file as your system folder, specify the **–mailbox** option. Z-Mail treats ***mailboxpath*** as your system folder instead of the default */usr/spool/mail/$USER* or */usr/mail/$USER* mailbox.

The **–mailbox** option is helpful if you have configured your mail transport agent to deposit your mail in various files in your home directory. By specifying this option, you tell Z-Mail to treat these files the same way it normally treats your system folder. This option does not affect the **mbox** variable (your *main mailbox*).

–motif

zmail command option *MOTIF ONLY*

SYNTAX **–motif**

To start Z-Mail's Motif interface running under the X Window System, specify the **–motif** option. You should place this option first on the command line, before any other X Window System or Z-Mail options.

EXAMPLE The following command starts Z-Mail's Motif interface:

% **zmail –motif**

–nofilter

zmail command option

SYNTAX **–nofilter**

To prevent Z-Mail from running message filters as it opens your active folder, specify the **–nofilter** option.

You define filters with the **filter** command in an initialization file. If you do not specify any filters, this option has no effect. If you specify both the **–nofilter** and **–filter** options, Z-Mail recognizes the last one on the command line.

–noheaders

zmail command option *COMMAND-LINE ONLY*

SYNTAX **–noheaders**

SYNONYM **–N**

To prevent Z-Mail from displaying message headers in the command-line interface, specify the **–noheaders** option. Z-Mail ignores this option if you specify the **–gui**, **–motif**, or **–lite** option.

–noinit

zmail command option

SYNTAX **–noinit[!]**

SYNONYM **–n**

To prevent Z-Mail from processing its system initialization file *$ZMLIB/system.zmailrc* at startup, specify the **–noinit** option. If you specify the optional exclamation point

(**–noinit!**), you prevent Z-Mail from processing your personal *.zmailrc* file as well. For more information about initialization files, see *Initializing Z-Mail* on page 63.

–offerFolders

zmail command option *GUI ONLY*

SYNTAX **–offerFolders**

Hand off folders to a folder server. This option is ignored except when used in combination with both **-gui** and **-folder** options, and only one folder (specified by the **-folder** option) can be handed off.

–picky

zmail command option

SYNTAX **–picky**

SYNONYM **–P**

Some mail transport agents do not accept *From:* and *Date:* headers generated by a user agent. To prevent Z-Mail from passing *From:* and *Date:* headers to your MTA, specify the **–picky** option, Specifying **–picky** is equivalent to setting the **picky_mta** variable to **omit_from**. For more information, see the **picky_mta** variable on page 299.

–readonly

zmail command option

SYNTAX **–readonly**

SYNONYM **–r**

If you specify the –**readonly** option, Z-Mail opens the initial folder "read-only." A read-only folder cannot be modified.

See also the **–r** option of the **folder** command on page 145.

–receive

zmail command option

SYNTAX

–receive

SYNONYM

–filter –folder –

The **–receive** option is a shorthand for **–filter –folder –**, which applies filters to messages that arrive on the standard input.

As messages arrive, Z-Mail places them in a temporary folder, applying filters that are defined in the initialization file(s). When the standard input closes, Z-Mail saves all messages from the temporary folder to the system mailbox, except those messages that are marked as saved or deleted.

You must be careful, therefore, to ensure that all messages which are marked as saved or deleted by your filters have been processed properly, and are stored somewhere or intentionally discarded.

EXAMPLE

The **–receive** option is particularly useful in standard automatic message filtering applications, such as *.forward* files. For example, the *sendmail* MTA will notice and use a *.forward* file in your home directory when it tries to deliver mail to you. To have Z-Mail process your mail automatically while you are on vacation, you might use the following in your *.forward* file (all on one line):

|"/usr/local/bin/zmail.small –init /usr/lib/Zmail/vacation
–mailbox /usr/spool/mail/*username*
–zcnlserv *nlshost*:2722 –receive"

Each time the *sendmail* MTA runs this command when mail arrives, it will execute the initialization file */usr/lib/Zmail/vacation*, which generates an automatic response to each message. The **–mailbox** option tells Z-Mail where to find your mailbox, and the **–zcnlserv** option tells Z-Mail where to obtain a license. You may not have to specify **–mailbox** or **–zcnlserv** depending on your system's configuration; see below.

The command(s) in a *.forward* (or *.maildelivery* for *MMDF*) file are normally run by the MTA under a special userid (usually *mail* or *root*), not under your userid. This means that your login and shell environment initialization files (*~/.login*, *~/.profile*, *~/.cshrc*) are not read. In particular, note that if you use the Z-Code Network License Server for Z-Mail licensing, you may need to include the **–zcnlserv *host*:*port*** option in your **zmail** command to obtain a license.

You system administrator can tell you whether your system uses *.forward* or *.maildelivery* files for automatic mail processing.

–refuseFolders

zmail command option *GUI ONLY*

SYNTAX **–refuseFolders**

Do not become a folder server. This option overrides the MediaMail default and any set X resources.

–register

zmail command option

SYNTAX **–register**

SYNONYM **–R**

To enter new local license information for your Z-Mail installation, specify the **–register** option. Z-Mail prompts you for your Z-Mail activation key. If you enter a valid key, Z-Mail updates your license file with the new key and other pertinent information. Then Z-Mail displays the new information about your Z-Mail license.

To register users for a local license, you must have permission to write to the */usr/lib/Zmail/license* directory. If you are using the NLS (Network License Server) for Z-Mail licensing, check with your system administrator to determine where the license information is located.

For additional information on recording your license information and registering users, see the *Z-Mail Installation Guide*.

–send

zmail command option *SEND-ONLY MODE*

SYNTAX **–send[!]**

SYNONYM **–U**

To immediately send a draft file without editing it further, specify the **–send** option. If you append the optional exclamation point (**–send!**), Z-Mail does not append signatures (see the **autosign** variable on page 259) or fortunes (see the **fortune** variable on page 278) to your outgoing message.

You may only use **–send** in conjunction with the **–draft** option. For more information see the **mail** command on page 171.

EXAMPLE

The following command immediately sends the draft file called *directions* to *linda* without waiting for further input:

```
% zmail –draft directions –send linda
27 lines
```

–shell

zmail command option

SYNTAX

–shell[!]

SYNONYM

–S

If you do not specify an interface, and your system folder or the folder you specify with the **–folder** option is empty or does not exist, then the **zmail** command will display

No mail for *$USER*

and exit. To force Z-Mail to start the command-line interface regardless, specify the **–shell** option.

EXAMPLE

The next command uses the **–shell** option to start Z-Mail even though the folder *meetings* does not exist:

```
% zmail –folder meetings –shell
```

You can also use the **–shell** option to force Z-Mail to start its command-line interface after you compose a message. Normally, when you specify options or addresses in your **zmail** command for composing a message, Z-Mail assumes you want to be in send-only mode, and returns you to your UNIX shell when you are finished composing your message. If you specify the **–shell** option, Z-Mail starts its command-line interface after you are finished composing the message.

The **–shell!** option works like the **–shell** option, above, but additionally causes all command-line arguments that follow **–shell!** to be passed into Z-Mail for Z-Script access, instead of being treated as e-mail addresses. The arguments that are passed into Z-Mail are accessible via **$*** (and other argument variables described starting on page 317) in scripts and initialization files.

EXAMPLE If you issue the command:

zmail –motif –shell! /usr/home/jerry/alternate reply

then the arguments /usr/home/jerry/alternate and reply are passed into Z-Mail and assigned to **$***. Also, the argument /usr/home/jerry/alternate will be assigned to **$1**, and reply will be assigned to **$2**.

–source

zmail command option

SYNTAX **–source[!]** ***filename***

SYNONYMS **–file, –F**

To force Z-Mail to read commands from a file *after* it opens your system folder, specify the **–source** option. Z-Mail reads commands from ***filename*** after it opens your system folder, but before it accepts input from you. The Z-Script commands in ***filename*** may include interactive commands, and commands that manipulate messages. Normally, interactive and message commands may not appear in initialization files because Z-Mail processes the initialization file before it opens the folder.

The optional exclamation point (**–source!**) prevents Z-Mail from starting one of its interfaces after sourcing ***filename***. Otherwise, Z-Mail continues and starts the interface you specify.

EXAMPLE The following command opens the folder *results* then processes the commands in the file *~/scripts/sortmsg*:

% **zmail –folder results –source ~/scripts/sortmsg**

–subject

zmail command option *SEND-ONLY MODE*

SYNTAX **–subject** ***text***

SYNONYM **–s**

To specify a subject for an outgoing message, specify the **–subject** option. Instead of prompting you for the subject of your message, Z-Mail uses ***text*** as its subject. If ***text*** has any spaces, you must enclose it in quotes.

You cannot specify **–subject** if you specify the **–gui**, **–motif**, or **–lite** option.

EXAMPLE

The following command starts a message addressed to *sales* with a subject of *New Vending Machines* and then waits for you to enter the text of the message.

```
% zmail –subject "New Vending Machines" sales
To: sales
Subject: New Vending Machines
. . . etc.
```

–template

zmail command option *SEND-ONLY MODE*

SYNTAX **–template *templatename***

SYNONYM **–p**

To load a message template as the starting point for a new message, specify the **–template** option. Z-Mail loads a text file named ***templatename*** as a new message for you to send. You then fill in the various fields of the template and send it.

You cannot use the **–template** option when you start Z-Mail's GUI or Lite interface. Instead, start **zmail** with the **–lite**, **–gui**, or **–motif**, option and select the *Templates* option from the *Compose* menu.

EXAMPLE

The following command starts a phone message addressed to *tfoley* by loading a prepared template:

```
% zmail –template phonemsg tfoley
To: tfoley
Subject: Phone Message

Message from: Ira Ajerbai
Phone Number: 800-555-2041
. . . etc.
```

–timeout

zmail command option *GUI ONLY*

SYNTAX **–timeout** ***nseconds***

SYNONYM **–T**

To specify how often you want Z-Mail to check for new mail, specify the **–timeout** option. Z-Mail then checks for new mail every ***nseconds***. For performance reasons, you cannot check for new mail more frequently than every thirty (30) seconds. If you specify this option, Z-Mail enters the GUI interface, even if you do not specify the **–gui** or **–motif** option.

EXAMPLE The following message starts Z-Mail's Motif interface and tells Z-Mail to check for new mail every two minutes.

```
% zmail –motif –timeout 120
```

–user

zmail command option

SYNTAX **–user [*username*]**

SYNONYM **–u**

If you specify the **–user** option, Z-Mail acts as if another user (specified by ***username***) is running Z-Mail. You are not prompted for a password. If you do not specify ***username***, Z-Mail uses *root*.

Note: Z-Mail does not change your userid, system folder, or permissions to that of the other user, but Z-Mail does set your home directory, real name, and login name to that of the user named ***username***. Z-Mail does not pass this information to the mail transport agent, preventing malicious or accidental mail forgery.

Use the **–mailbox** command option (see page 51) to change your system folder to that of another user. You cannot change to another system folder after you start Z-Mail.

The **–user** option is useful for system administrators who need to read mail directed to *postmaster* or other pseudo-users.

–verbose

zmail command option *SEND-ONLY MODE*

SYNTAX **–verbose**

SYNONYM **–v**

To force Z-Mail to pass the ***verbose*** option to your mail transport agent when Z-Mail sends messages to it, specify the **–verbose** option. Some MTAs do not have a **verbose** option, so this feature may not apply to your system. This option applies only when you send mail and cannot be used if you specify the
–gui, **–motif** or **–lite** option.

Setting the **verbose** variable in any user interface also causes Z-Mail to display additional information about the *send* process. For additional information, see the **verbose** variable on page 313.

–version

zmail command option

SYNTAX **–version**

RELATED COMMAND **version**

RELATED VARIABLE **version**

The **–version** option causes Z-Mail to display its version information and immediately exit.

–zcnlserv

zmail command option

SYNTAX **–zcnlserv** ***host*** **[:*port*]** **[*host*** **[:*port*]** ***host*** **[:*port*] . . .]**

SYNONYM **–Z**

RELATED VARIABLE *ZCNLSERV* (environment variable)

When Z-Mail starts, it requires a *license* before it will open folders or send mail. Your site has two options for Z-Mail licensing: local or network. Local licenses are tied to a particular host, while network licenses can be spread among users on several hosts. Local licenses are typically used by small Z-Mail installations, while larger installations use the NLS

(Network License Server) for its flexible license administration.

By default, Z-Mail will attempt to obtain a local license by looking in the file */usr/lib/Zmail/license/license.data* for a valid activation key. If you use local licensing, you do not need to use the
–zcnlserv option.

If you specify the **–zcnlserv** option, Z-Mail will attempt to obtain a network license from the NLS at ***host*:*port***, where ***host*** is the name (or network address) of the machine that is running the NLS software, and ***port*** is the "port number" on which the NLS is listening. ***port*** is 2722 by default, but may have been changed to a different number by the system administrator when the NLS was installed. You may omit **:*port*** if your NLS uses port number 2722. On most systems, you can find out what ***port*** the NLS is using by examining the zcnls entry in the ***host*** computer's */etc/services* file.

Optional backup NLS hosts may be specified in the order that they should be contacted. If you specify more than one NLS host, separate the ***host*:*port*** pairs by spaces and place the entire list of ***host*:*port*** pairs in quotes. Your system administrator can tell you what sequence of backup NLS servers to use.

You may instead set the *ZCNLSERV* environment variable to one or more ***host*:*port*** pairs; this has the same effect as using the
–zcnlserv option.

See the file *$ZMLIB/license/bin/README.NLS* for more information about installing, configuring, and tuning the NLS. This file may not be present on your system if the system administrator has not installed the NLS software.

EXAMPLES

If there are three NLS server machines at your site, named alpha, beta, and gamma, and alpha is the primary server, then you might use the following command to start Z-Mail:

```
% zmail –motif –zcnlserv "alpha beta gamma"
```

If your NLS doesn't use the default port number 2722, then you must supply the port number in the ***port*** argument:

```
% zmail –motif –zcnlserv "alpha:2226 beta:2226 gamma:2226"
```

If your NLS is in a different domain, you may need to specify the entire domain name:

% **zmail –motif –zcnlserv "alpha.is.justin.com:2226"**

Also see the **–receive** option on page 54 for an example of using **–zcnlserv** as part of a *.forward* automatic mail processor.

Initializing Z-Mail

When Z-Mail starts, it processes one or more initialization files that run Z-Script commands which, among other things, define variable values, filters, address aliases, and command aliases. You can customize your Z-Mail installation on a system-wide or individual basis by modifying the initialization files.

When Z-Mail starts, by default it:

- interprets the arguments supplied with the **zmail** command,
- loads a series of initialization files, and
- opens your system folder.

Loading Initialization Files

Automatic Initialization

After interpreting the **zmail** command options, Z-Mail configures itself to your specifications by loading Z-Mail initialization files. Z-Mail loads the following files, in order:

1. Z-Mail first looks for a **–init** or **–I** option to the **zmail** command, and loads the specified file.
2. Next, Z-Mail loads the default system initialization file, *$ZMLIB/system.zmailrc*. *ZMLIB* is typically */usr/lib/Zmail*. This file is usually created by the system administrator and contains Z-Mail commands and settings for all Z-Mail sessions.
3. Z-Mail then loads your personal initialization file, which is the first file Z-Mail finds from the following list:
   - The file named in the *ZMAILRC* environment variable.
   - The file named in the *MAILRC* environment variable.
   - The file *.zmailrc* in your *home* directory (*~/.zmailrc*).
   - The file *.mushrc* in your *home* directory (*~/.mushrc*).

Initialization options

By specifying any of the following options to the **zmail** command, you affect the initialization process:

–init *filename* If you specify the **–init** or **–I** option, Z-Mail reads ***filename*** before any other initialization files, then reads other initialization files as specified above.

–init! *filename* If you specify the **–init!** or **–I!** option, Z-Mail reads ***filename*** before any other initialization files, then reads your personal initialization file, and does not read the system initialization file *system.zmailrc*.

–noinit If you specify the **–noinit** or **–n** option, Z-Mail reads only *~/.zmailrc* and a file optionally specified by **–init**.

–noinit! If you specify the **–noinit!** or **–n!** option, Z-Mail reads no initialization files whatsoever, except for a file optionally specified by **–init**.

–source *filename* If you specify the **–source** option, Z-Mail reads the specified file of Z-Script commands after the initial folder is opened.

Opening an Active Folder

Loading Messages

After Z-Mail reads your initialization files, it loads all the messages in your system folder. You may tell Z-Mail to read a different folder at startup by specifying the **–folder** option to the **zmail** command:

zmail –folder *fname*

Z-Mail will load the messages in the ***fname*** folder and make it the active folder after initialization.

Sorting Messages

If you set the **sort** variable, then after Z-Mail loads the initial folder's messages, it sorts the messages according to the value of the **sort** variable. For more information, see the **sort** variable on page 308.

Sourcing Other Initialization Files

After Z-Mail loads the messages in your active folder, it checks if you specified a **–source** option in your **zmail** command. If so, it sources the file you specified in the option. In effect, this file is another initialization file, except that files you **source** are read *after* the system folder is opened. See the **–source** option on page 57 for more information.

Manual Initialization

Once you start Z-Mail, you may load initialization files manually using **source**. See the **source** command on page 210 for more information. You may also use the **–eval** command option to tell Z-Mail to execute a Z-Script command after loading the initial folder. See the **–eval** option on page 44 for more information.

Starting the User Interface

After loading all your initialization files and opening your active folder, Z-Mail starts the user interface you selected with one of the following options. Note that these options must appear before any other command options. See *Z-Mail's User Interfaces* on page 20 for more information.

–motif If you specify the **–motif** option, Z-Mail starts its Motif interface.

–lite If you specify the **–lite** option, Z-Mail starts its Lite character interface.

-shell If you specify **–shell** without **–motif** or **–lite**, Z-Mail starts its command-line character interface.

(none) If you don't specify a user interface, Z-Mail checks your system folder and enters the command-line interface only if you have new mail. Otherwise, Z-Mail displays No mail for *$USER* and exits.

Managing Initialization Files

An initialization file is a script of Z-Mail commands and settings stored in a text file.

Creating Personal Initialization Files

Saving Z-Mail Settings

The easiest way to customize your initialization file is to start with one with settings you like. Your Z-Mail settings are usually stored in the file *~/.zmailrc*.

When Z-Mail writes the *~/.zmailrc* file, it only includes those settings and functions that differ from those specified in the global *system.zmailrc* initialization file.

GUI or Lite

To create an initialization file in the GUI or Lite interface:

1. Set variables to values you want using the *Variables* dialog box (Choose the *Variables* item from the *Options* menu).
2. Click the *Save* button in the Variables dialog box to create the initialization file *~/.zmailrc*.

The *Save* button asks if you want to overwrite your *.zmailrc* file. If you have made no manual changes to the file, you should confirm the overwrite.

WARNING: If you have added commands and comments or made other manual changes to the file, the save procedure overwrites those changes and they are lost.

Command-line

To create an initialization file in the command-line interface:

1. Set variables to values you want.

   ```
   set askcc
   set autosign = "Robert Trimble"
   unset hold
   set hdr_format = "%22n %M %–2N %5T (%1) %25s"
   ```
   etc. . . .

2. Use the **saveopts** command to save all current variable settings, aliases, and other Z-Mail settable attributes into an

initialization file.

The **saveopts** command prompts you for a name for the new file and asks you if you want to overwrite an existing file, if a file by the name you specify already exists.

WARNING: If you have added commands and comments or made other manual changes to the file, the save procedure overwrites those changes and they are lost.

Editing Initialization Files

You can edit existing initialization files to:

- Change variable settings,
- Add or modify aliases,
- Define functions and buttons,
- Add conditional expressions, and
- Add comments.

Adding Commands

Since Z-Mail treats initialization files as scripts, you can use almost any Z-Script command in an initialization file.

exit

The **exit** command has special meaning in a Z-Mail initialization file. If Z-Mail encounters an **exit** command, it immediately stops reading from that file and proceeds with the remainder of initialization. The **quit** and **x** commands cause Z-Mail to exit completely.

Comments

When Z-Mail reads the initialization file, it recognizes the pound (**#**) character as a comment character. It may be anywhere on a line in the file. When this character is encountered, Z-Mail treats the remainder of the line as a comment and does not process it. If you enclose the **#** in single or double quotes, Z-Mail does not consider it as the beginning of a comment.

EXAMPLE

For example:

```
# this entire line is a comment
set shell = /bin/csh  # only the end of this line is a
     comment.
# the # sign on the next line is inside quotes.
set prompt = "Message #%m: "
```

Commands to Avoid

You should avoid using certain kinds of commands in Z-Mail initialization files.

Interactive Commands

You should never use interactive Z-Script or UNIX commands in a Z-Mail initialization file. Z-Mail cannot prevent you from using these commands because it is impossible to trace which UNIX commands are interactive. For example, the **ask** Z-Script command presents an interactive dialog box to the user, but during initialization Z-Mail has not yet started a user interface that is capable of presenting the dialog. Noninteractive commands that present information to the user, such as **echo**, are permitted in initialization files.

Commands that Affect Messages

Since Z-Mail reads initialization files before it loads any messages, you should not use commands that operate on messages in an initialization file unless the file will only be run using the **source** command.

Conditional Expressions

You may use **if** conditional expressions in initialization files. For additional information on writing conditional expressions, see the **if...else...endif** command on page 161. This section summarizes the use of the **if** command.

Z-Mail Internal Variables

Z-Mail provides several internal variables that help you set up conditional processing in initialization files so your initialization can take different actions, depending on the interface you've started.

The following chart summarizes Z-Mail's internal variables. For details see each variable's entry in *Z-Mail Variables* beginning on page 223.

Variable	***Evaluates to true if:***
is_gui	You started Z-Mail with the **–motif** or **–gui** option.
is_lite	You started Z-Mail with the **–lite** option.
is_shell	You are in the Z-Mail program, and not just calling Z-Mail from a UNIX command shell. When **is_shell** is true, Z-Mail's user interface has been fully started and interactive commands are permitted.

is_sending	You are sending a message from the UNIX command line; no interface will be started.
hdrs_only	You started Z-Mail with the **–headers** option, telling Z-Mail to only process message headers and exit.
redirect	Input to Z-Mail has been redirected.

EXAMPLES

The following script sets the **autoprint** variable if the GUI interface is in use:

```
if  is_gui
        set  autoprint
else
        exit        # exits the script, not Z-Mail
endif
```

The following script tests to see if your terminal type is *adm3a* by checking the value of the *TERM* environment variable. If the terminal type is *adm3a*, the script sets the **pager** variable to the *more* program. Otherwise, it sets the **pager** variable to *less*.

```
if  $TERM  ==  adm3a
      set  pager  =  more
else
      set  pager  =  less
endif
```

Using Z-Mail in an NFS Environment

This section contains information for system administrators regarding the use of Z-Mail in an environment where NFS-mounted volumes are used as mail delivery and folder-storage areas.

NFS file locking

In general, NFS-mounted filesystems that are acting as mail delivery areas do not support file locking. If you use NFS-mounted filesystems for mail delivery, and your MTA supports dot-locking, you should put the line

```
set dot_lock
```

into your site's *system.zmailrc* file to tell Z-Mail to use *.lock* lock files, as well as file locking, for synchronization with the MTA. Note that this will require you to ensure that the

Z-Mail program has the ability to create and remove lock files in the mail delivery area. If your mail delivery area's permissions are already *world-write* and *owner-delete* (world permission **t**) then you're set. Otherwise, you'll have to give the Z-Mail executable the *set-group-id* permission and *chgrp* it into the group that owns the mail delivery area.

If your MTA does *not* support dot-locking and you are using NFS-mounted filesystems for mail delivery, you may be in for trouble. We don't recommend that you set your mail delivery system up this way, since file locking on NFS servers is, in general, non-robust.

See also the **dot_lock** variable in the *Z-Mail 2.1 Reference Manual* for more information.

NFS caching

You should ensure that the NFS volume where your mail delivery directory is located has attribute caching turned *off*. This means that you should mount the volume with the *noac* option. Cached attributes can cause file information like file size and modification date to be misreported by NFS, which confuses Z-Mail and other MTAs.

Hard NFS mounts

NFS-mounted mail delivery areas should always be *hard mounted*, not soft mounted.

CHAPTER 4

Using Z-Mail's Command-Line Interface

Z-Mail's command-line interface is designed for the more experienced user who prefers a command-driven mail system. The character interface can be used anywhere you can use a shell, including workstations, standard ASCII terminals, teletypewriters, and terminal emulators. Since the output of the command-line interface is relatively terse, it is a good choice if you are running Z-Mail over a slow connection (such as a low speed modem dialup) to a host computer.

The command-line interface uses a standard tty-command mode similar to other command-driven programs such as *csh.* The Z-Mail command-line interface has many commands in common with other mailers, such as the BSD *Mail* and UNIX System V *mailx* programs. However, Z-Mail is far more flexible and powerful than these older programs.

You interact with the command-line interface by typing one or more Z-Mail commands and pressing **Enter**. If you don't specify all the information necessary to complete your request, Z-Mail prompts you for additional information.

Starting Z-Mail's Command-Line Interface

To start Z-Mail's command-line interface, do not specify an interface option (**–lite**, **–gui**, or **–motif**) in your **zmail** command:

zmail

If you have no messages in your system folder, Z-Mail will notify you with **No mail for** *$USER* and exit. To start the command-line interface regardless of whether you have mail, use the **–shell** option:

zmail –shell

zmail.small

If your system administrator has installed the command-line-only version of Z-Mail, called **zmail.small**, you may use it instead. The **zmail.small** program loads faster and takes fewer system resources than the **zmail** program, since the GUI and Lite interfaces are not present in **zmail.small**. The **zmail.small** and **zmail** programs are identical in all other respects.

Command-Line Interface Basics

In the command-line interface, you can issue nearly all of the many Z-Script commands described in *Z-Script Commands* beginning on page 109. To perform the everyday tasks of managing your mail, you'll need to know only a small number of the available commands.

Startup Information

When you start Z-Mail's command-line interface, Z-Mail displays:

- the name of your active folder,
- the number of messages in the folder,
- a screenful of message summaries for the messages in the folder, and
- a customizable prompt indicating the current message number or other meaningful information.

EXAMPLE

```
[0]+ "/usr/spool/mail/psteele": Type '?' for help.
>1   r   Sean O'Brien     Jul 15    8:49  Meeting at 11:30
 2       Shawna Wilson    Jul 16   11:58  Tuesday's schedule
 3   p   Jon Yost         Jul 16   12:10  Porting standards
 4   N   Ian Trainor      Jul 16   13:05  Maintenance program
 5   N   Shawna Wilson    Jul 18    8:31  Reports Due Now
 6   N   Terri Johnson    Jul 18   10:02  Version 3.1 is ready
psteele: (#1 of 6) >
```

The actual output you see could be different because the format of both the headers and the command prompt are customizable. For details, see the **summary_fmt** variable on page 309 and the **prompt** variable on page 303.

Current Message

The > character next to the first message indicates that it is the *current message*, as does the string #1 of 6 in the command prompt at the bottom of the example.

Message Lists

When you enter commands in Z-Mail, you can optionally include a message list. If you do not specify a message list, Z-Mail performs the command you enter on the *current message* (or messages) as indicated by the value of the **output** variable. Every time you execute a command, Z-Mail resets the value of **output** to the messages that were affected by the command. For more information, see *Specifying Message Lists* on page 24 and the **output** variable on page 298.

The rest of this section explains how you perform basic mail management functions using Z-Mail's command-line interface. For a complete discussion of each command in detail, see *Z-Script Commands* beginning on page 109.

Displaying Messages

To display the current message, enter **display**. You can also enter **type**, **t**, **print** or **p** to display a message. You can abbreviate **display** as **.** (period). You cannot abbreviate **display** as **d**; as in the UNIX *mail* program, **d** is an abbreviation for the **delete** command.

EXAMPLES

```
...
>5   !   Shawna Wilson    Jul 18    8:31  Reports Due Now
 6   r   Terri Johnson    Jul 18   10:02  Version 3.1 is ready
psteele: (#5 of 6) >  display
Message  #5  (15  lines)
From:  swilson@utech.com  (Shawna Wilson)
Date  Mon,  18 Jul 1991  8:31:19 PDT
To:  research
Subject:  Reports Due Now

The deadline for product status reports is today at noon. If you have
    yet
to submit your report, please e-mail it to me by the end of the day.
```

You can also display a specific message by entering its number, or by entering **display** and its number:

```
psteele: (#5 of 6) > 3
psteele: (#5 of 6) > display  3
```

To display the next undeleted and unsaved message in your active folder, enter **next**. To display the previous undeleted and unsaved message, enter **previous**.

Displaying Header Summaries

To display the header summaries for the messages in your active folder, enter **headers** or **h**.

EXAMPLE

```
psteele: (#8 of 14) > headers
```

```
>1   !   Sean O'Brien      Jul 15    8:49   Meeting at 11:30
 2   !   Shawna Wilson     Jul 16   11:58   Tuesday's schedule
 3   N   Jon Yost          Jul 16   12:10   Porting standards
. . .
14   p   Terri Johnson     Jul 27   14:45   Tech support issues
psteele: (#8 of 14) >
```

For information on how to customize the content and format of the information Z-Mail displays in your header summaries, see the **summary_fmt** variable on page 309.

Sending Messages

To begin composing a new message, enter the **mail** command. Z-Mail then prompts you with header names (*To*, *Subject*) for you to fill in. You must enter at least one address in the *To:* header. All other headers are optional; to omit them from your outgoing message, simply press **Enter** to not specify a value. When Z-Mail finishes prompting you for headers, it skips a line and puts the cursor at the left side of the screen. You can then enter the body (the main text) of your message.

EXAMPLE

```
>5   !   Shawna Wilson     Jul 18    8:31   Reports Due Now
 6   r   Terri Johnson     Jul 18   10:02   Version 3.1 is ready
psteele: (#5 of 6) >  mail
To:  jyost, tjohnson
Subject:  Trip to London

I'll be out of the office for the next 10 days on a trip to London. If
you need to reach me, Lori has the itinerary for my trip. I'll be
reading my mail every few days as well. See you in two weeks.
```

When you are finished typing your message, type **Enter** to move the cursor to the beginning of a new line. Then type your terminal's end-of-file character (usually **Ctrl-D**) to send your message. If the **askcc** variable is set, Z-Mail will prompt you with **Cc:** for a list of users to whom you want to send carbon copies of your message. See the **askcc** variable on page 253 for more information.

Replying to Messages

When you reply to a message, Z-Mail automatically addresses the reply message and supplies a Subject: header. To send a reply, you can enter one of these commands:

reply	Reply to sender of the current message.
replyall	Reply to sender and all recipients of the current message.

You can abbreviate **reply** as **r** and **replyall** as **R**. You can also specify the number of the message to which you want to reply.

EXAMPLE

This command replies to the sender of message number 6:

```
>5   !   Shawna Wilson   Jul 18    8:31  Reports Due Now
 6   !   Terri Johnson   Jul 18   10:02  Version 3.1 is ready
psteele: (#5 of 6) > reply 6
To: tjohnson@utech  (Terri Johnson)
Subject:  Re: Version 3.1 is ready

Sounds great.  How can I get a copy of Version 3.1 to test?
<Ctrl-D>
```

Forwarding Messages

To forward a copy of a message to other users, enter **mail –forward**, and Z-Mail prompts you for the addresses of the recipients:

EXAMPLE

```
mail –forward 3
To:  itrainor  tjohnson
forwarding message 3 . . . 25 lines

Please review the above report and send me your comments . . .
```

Z-Mail then lets you enter text that it appends to the end of the forwarded message. When you are finished entering text, type **Ctrl-D** to send the message.

To forward a message without editing it, use **mail –resend**:

EXAMPLE

```
mail –resend 3
To:  itrainor  tjohnson
forwarding message 3 . . . 25 lines
```

The message is forwarded immediately.

Signing Messages

To append a personal 'signature' to the end of your outgoing messages, see the **autosign** variable on page 259, and the **~S** command on page 95.

Canceling a Message

To cancel a message you are composing without saving it, type your terminal's interrupt character (usually **Ctrl-C**). Z-Mail will respond:

```
** interrupt – – one more to kill letter **
```

Type the interrupt character again to cancel the message.

Use **~q** to cancel a composition in progress and save it to the folder specified by the **dead** variable (usually *~/dead.letter*). See the **~q** tilde-escape command on page 99 for more information.

Use **~x** to cancel a composition in progress without saving it to the **dead** folder. See the **~x** tilde-escape command on page 100 for more information.

Use **~z** to suspend a composition in progress. See the **~z** tilde-escape command on page 99 for more information. You can return to the suspended composition with the **resume** command (see page 195), and list suspended messages with the **jobs** command (see page 169).

Adding Messages, Files, and Attachments

Z-Mail gives you several ways to add text and/or data to your mail messages. You may:

- include the contents of another message in your message,
- copy an external text file into your message,
- use the contents of an external file as a template, and
- attach an external file to your message.

Including an Existing Message

When you send a new message, you may want to add your comments into the existing text of a message that you include. To start a new composition that includes the current message, enter:

mail –i

You can also include the text of a specific message by supplying its message number to the **mail** command:

mail –i 4

Z-Mail will include the text of message 4 in your composition, clearly mark it as an included message, and allow you to edit its contents. To invoke an external text editor, enter **~e** and Z-Mail starts the editor indicated by the value of the **editor** variable (see page 274) or **visual** variable (see page 314) if **editor** is undefined.

Frequently, when you reply to a message, you'll want to include a copy of the original message to make your reply meaningful to its recipients. To include a copy of the original message in your reply, add the **–i** option to your **reply** or **replyall** command:

reply –i
replyall –i

Z-Mail includes a copy of the original message in your reply, identifying it so the recipients of your reply can distinguish your message from the original message. For details on how Z-Mail formats the included message, see the variables **indent_str** on page 283, **pre_indent_str** on page 300, and **post_indent_str** on page 300.

EXAMPLE

psteele: (#2 of 10) > **mail –i**
To: **rgomez**
Subject: **Feedback from our meeting**

including message 2 **. . .** (37 lines)
I have assembled a few comments that I heard about our last meeting and included them for your review.
<**Ctrl-D**>

Reading Text Files

Z-Mail can read the entire contents of a text file into the body of a composition. To read the file, enter **mail –H** followed by the pathname of the file. If the text file contains the headers for your outgoing message, enter **mail –h** and Z-Mail looks for the message's subject, carbon copy list, etc., at the beginning of the text of the file as Z-Mail reads it.

Z-Mail loads the exact text of the file as if you had typed all its contents yourself, then lets you edit the text of the file using an external text editor. To invoke a text editor, enter **~e** and Z-Mail starts the editor indicated by the value of the **editor** variable. If the file is a template, you can fill in its various fields and add comments. When you are done, type **Return** to move the cursor to the beginning of a line, then **Ctrl-D** to send the message.

EXAMPLES

```
psteele: (#5 of 6) > mail –H results.txt
To: sales
Subject: Survey results

105 lines
(continue editing or ^D to send)
If you have any questions, please call me or send me a message.
<Ctrl-D>
```

Note: If you want to precede the included file with your comments, type the comments and use the **~r** tilde-escape command to read in the text of the file. See page 96 for more information about **~r**.

If you use the **–h** option to **mail**, Z-Mail assumes it can find all the header information it needs in the text file and does not prompt you for header information.

```
psteele: (#5 of 6) > mail –h results.txt

105 lines
(continue editing or ^D to send)
Here's a summary of the results from the customer satisfaction
survey.
<Ctrl–D>
```

You can also attach files to a message by entering **~r** at the beginning of a text line. For more information, see the **editor** variable on page 274 and *Tilde-Escape Editing Commands* on page 91.

Using Templates

A message *template* is a text file that you can load as a starting point for composing a new message. The **templates** variable holds the name of a directory where you keep your message templates; **templates** is set to *$ZMLIB/forms* by default.

To use a template, specify the template name (which is the filename of the template) in the **mail –p** ***template-name*** command. Z-Mail will look in the directory specified by the **templates** variable for the ***template-name*** you specified.

The following command starts a new message for you to compose based on the *phone* template.

```
triley (#4 of 17) > mail –p phone

18 lines
(continue editing or ^D to send)
```

~e

Since you'll probably want to edit the message you've started with a template, it is easiest to do so in an external editor. See the **~e** tilde-escape command on page 93 for instructions on starting an external editor.

For more information, see the **templates** variable on page 310, the **visual** variable on page 314, the **editor** variable on page 274, and *Tilde-Escape Editing Commands* beginning on page 91.

Sending Attachments

To send an external file such as a text file or an application's document (such as a FrameMaker or Wingz file) along with your message, you can *attach* the file to your composition. When you attach a file, Z-Mail does not make it a visible part of the message body, but instead sends it like an enclosure in a business letter. By sending the attachment in this manner, recipients who use Z-Mail can easily view, detach, and save the file in its native format. For general information about attachments, see the *Z-Mail User Guide*.

The mechanisms used in attachment creation, display, encoding for mailing, and decoding after mailing are defined in the *$ZMLIB/attach.types* file. You may add new attachment types to Z-Mail, or modify existing ones. For more information, see *The Attachment Types File* on page 328 and the **attach** command on page 122.

Non Z-Mail recipients

Recipients who are *not* using Z-Mail to read a message containing an attachment can use a text editor to extract the attachment data from the message body. Instructions embedded in the message (the *X-Zm-Decoding-Hint*) tell the recipient how to re-create your original file using the information in the *attach.types* file.

To attach a file to a composition when you initiate it, enter **mail –A** followed by the pathname of the file you wish to attach to the message. The following Z-Mail command attaches a file called *~/survey.doc* to the outgoing mail message.

EXAMPLE

```
psteele  (#11 of 52) > mail  –A  ~/survey.doc
Attachment type (? for list):  framemaker
Attachment description:  Corporate Benefits Survey
Encoding (? for list):  uuencode
```

To: allmgrs
Subject: Draft Copy of Benefits Survey

Z-Mail then lets you enter the text of your message. For technical details on specifying attachment types, see *Defining Attachment Types* on page 328.

You can also attach files to a composition in progress by entering **~A** at the beginning of a line. For more information, see the **~A** command on page 97.

Deleting Messages

To delete a message or list of messages, use the **delete** command:

delete [*messagelist*]

If you do not specify ***messagelist***, Z-Mail deletes the current message. Deleting a message does not actually remove it from the folder until the folder is *updated*. You may undelete a message that you have marked for deletion with the **undelete** command. Once a deleted message has been removed from the folder by an update, the message cannot be undeleted.

For more information, see the **delete** command on page 134.

For more information about message lists, see *Specifying Message Lists* on page 24.

To suppress the display of deleted messages in header listings and elsewhere, unset the **show_deleted** variable. See the **show_deleted** variable on page 307 for more information.

EXAMPLES

For example:

delete	Deletes the current message.
delete 2 4-6	Deletes messages 2 and 4 through 6.
d 6-$	Deletes messages 6 through the last message.

Saving Messages

You can *save* messages in *folders* for future reference or storage. To save a message or list of messages to a folder, use the **save** command. The general syntax of the command is:

save [*messagelist*] [*foldername*]

If you do not specify ***messagelist***, Z-Mail saves the current message. If you do not specify ***foldername***, Z-Mail saves the message in a special folder called your *main mailbox*. The location of your main mailbox is defined by the **mbox** variable (see page 291).

For more information about saving messages, see the **save** command on page 196. See also the **copy** command on page 133 (which saves without deleting), and the **write** command on page 221 (which saves without including headers).

EXAMPLES The following commands perform as indicated:

save	Saves current message to your main mailbox.
save 5 7 meetings	Saves messages 5 and 7 to the *meetings* folder.
save 1–. +norm	Saves message 1 through the current message to the folder **$folder**/*norm*.

By default, Z-Mail does not delete messages from your active folder after saving them to another folder. For details on changing this behavior, see the **hold** variable on page 281 and the **keepsave** variable on page 288.

Printing Messages

To print a message or list of messages, use the **lpr** command. The general syntax of the command is:

lpr [*messagelist*]

If you do not specify ***messagelist***, Z-Mail prints the current message(s). The behavior of the **lpr** command can be tuned with the **print_cmd**, **printer**, and **printer_opt** variables.

EXAMPLES The following commands perform as indicated:

lpr	Prints the current message.

lpr 1 3 4	Prints messages 1, 3 and 4.
lpr .–$	Prints the current message through the last message.

For more information on message lists, see *Specifying Message Lists* on page 24.

Using Aliases and Distribution Lists

The **alias** command allows you to define a name that stands for a list of mail addresses. For example, if you send mail regularly to two field sales managers, you may want to define the alias *salesmgr* as a shorthand for their e-mail addresses:

alias salesmgr kthomas@east ariley@west

For details about aliases, see the **alias** command on page 117.

Sorting Messages

The **sort** command orders message by author, date, status, priority or subject. For example, to sort your active folder by date and within date, by message length, enter:

sort –d –l

For more information about sorting, see the **sort** command on page 208.

Searching Messages

The **search** command allows you to select messages that match criteria you specify. For example, the following command finds all the messages in the active folder that are dated prior to January 1, 1991.

search –d –1/1/91

The following command finds all messages in the active folder with *kthomas* in the *From* header.

search –f kthomas

Using pipelines, you can feed the output of one **search** command into another, executing complex searches for specific sets of messages. For more information about searching messages, see the **search** command on page 199.

Special Command-Line Interface Features

Z-Mail supports several features that you may want to take advantage of as an experienced mail user:

- customizable command prompt,
- command history,
- command-line aliasing, and
- pipelines that redirect command input and output.

Customizable Command Prompt

Z-Mail lets you customize the prompt that Z-Mail displays when it is waiting for you to enter a command. When Z-Mail displays its command prompt, it evaluates the **prompt** variable, which by default is set to the string:

Msg %m of %t:

So if you have 10 messages and your current message is 5, then your Z-Mail prompt looks like:

```
Msg 5 of 10:
```

For more information on customizing your command-line prompt, see the **prompt** variable on page 303.

Command History

You can refer to the history of commands you entered earlier in your Z-Mail session, rather than retyping them at the command-line to re-execute them.

Z-Mail supports a history mechanism similar to the one supplied by *csh*. Z-Mail also supports selected *csh* history modifiers so you can refer to commands you previously issued and to extract arguments you specified.

The Z-Mail history mechanism stores your most recent *n* commands, where *n* is the value of the **history** variable. If you do not set this variable, Z-Mail only remembers your most recent command.

EXAMPLES

The following command sets the value of the **history** variable to 25, causing Z-Mail to record your most recent 25 commands.

set history = 25

The following command displays the most recent 4 commands.

psteele: (#5 of 6) > **history 4**
42 display
43 mail –i
44 delete
45 history 4

For details, see the **history** command on page 160 and the **history** variable on page 281.

Referring to Previous Commands

To reference commands you previously entered, use the exclamation point (**!**). You can select some or all of the arguments in previous commands, as well as print the commands using these special characters. The arguments in every command are numbered beginning with argument 1. The command name itself is argument 0. The general form of a history reference is ***command:action*** where:

Command	***Refers to***
!*n*	command number *n*.
!–*n*	command that was *n* commands ago.
!!	the last command (same as **!-1**).
!*pattern*	the most recent command beginning with *pattern.*
!?*pattern*?	the most recent command containing *pattern.*

Action	***Refers to***
n	argument number *n* (first argument is 0).
n-m	arguments numbered *n* through *m*.
$	the last argument.
*	all arguments.
p	prints (displays) the command without executing it.
s/*pattern1*/*pattern2*	Replaces *pattern1* with *pattern2*.

EXAMPLES

Using these special characters, you can refer to commands you typed earlier.

!–2	Re-execute the command from two commands ago.
!!	Re-execute the most recent command.

mail !!:1-3 Send a message using arguments 1, 2, and 3 from
the most recent command.
!4:p Show the fourth command and its arguments.
!sort Re-execute the most recent *sort* command.
!h Re-execute the most recent command beginning with the letter h.
!4 ∗ Execute the fourth command, appending ∗ to it.
!?mail?:p Show the most recent command containing *mail*.
!?mail?:s/mail/reply/
Execute the most recent command containing *mail*, replacing *mail* with *reply*.

Turning Off History References

If you send messages using UUCP notation (which contains exclamation points, also known as "bang" characters), you may want to disable Z-Mail's history capabilities. To do so, set the **ignore_bang** variable. Z-Mail then treats all **!** characters literally instead of trying to expand them as history references.

Note: When you **source** initialization files (e.g., *.zmailrc*), Z-Mail does not perform any command history processing.

Command Aliases

You use command aliases to create special names for commands. This feature enables command substitution similar to *csh*. To create a command-line alias, use the **cmd** command whose general syntax is:

SYNTAX

cmd [*name* [*command*]]

When you specify ***name*** and ***command***, Z-Mail creates a command-line alias called ***name***. Then when you type ***name*** at the command line, Z-Mail executes the Z-Mail command called ***command***. The following commands define command aliases for the Z-Mail commands that follow them:

cmd d delete
cmd t type
cmd dt 'd; t'
cmd – previous

You may embed command history references in ***command***; they will be expanded each time ***command*** is executed. See *Command History* on page 85 for a list of command history references you can use in the command-line interface.

The following command history references may be embedded in ***command*** in all interfaces.

All interfaces

!!	the entire command line
!$	the last word of the command line
!*	all arguments of the command line (entire line except ***name***)
!:$	the last word of the command line
!:*n*	the ***n***th word of the command line
!:*m–n*	the ***m***th through ***n***th word of the command line.
!:–*n*	word zero through ***n*** of the command line
!:*	Same as **!***, above
!:*n*–	the ***n***th word through the next-to-last word of the command line
!:p	Print, but do not execute, the command line

For example, the following command alias uses the **!*** history reference. Note that **!** characters must be escaped with a backslash (****) to prevent them from being interpreted in the command.

cmd r 'reply \!* ; delete ; next'

Using this example, if you type the command:

r –e 12

it is translated to:

reply –e 12 ; delete ; next

which will start a **reply** composition to message number 12, then **delete** message 12 (since it was selected by the **reply** command), and display the **next** message.

For more information, see the **cmd** command on page 131.

Piping Commands

You can *pipe* the output of one Z-Mail command to another using the vertical bar or *pipe character* (**|**). When two commands are separated with a pipe, Z-Mail uses the output of one command as input to the next command. When you pipe multiple commands together on the same line, they are called a *pipeline*.

The first command in a pipeline uses either the current message or the messages you indicate in the command. Thereafter, each Z-Mail command in the pipeline passes a *message list* of the messages it affected to the next command in the pipeline. Z-Mail suppresses the screen output of all but a few of its commands when you use pipelines.

A common use of pipelines in Z-Mail is to connect the output of commands that search or select messages to the input of commands that act on those messages.

Important Note Remember, Z-Mail pipes message lists, *not* text, to Z-Mail commands. See *Pipelining UNIX Commands*, below, for instructions regarding piping text to UNIX commands.

EXAMPLES

The following command finds all the messages from *fred* and saves them all in the folder named *fred_mail:*

search –f fred | save fred_mail

The next sends messages 4, 5, 6, 7, and 8 to the printer and then deletes them:

lpr 4–8 | delete

The following example deletes all old mail (mail you have already read):

headers :o | delete

The next command prints all the message that are more than two weeks old and from *rthomas*.

search –ago 2 weeks | search –f rthomas | lpr

Pipelining UNIX Commands

Since Z-Mail pipelines pass message lists from command to command, and UNIX commands expect text on standard input, you must precede all UNIX commands with Z-Mail's **pipe** command. The **pipe** command allows you to direct a list of messages to UNIX commands.

You may never specify a UNIX command as the first command in a pipeline, unless you use the **pipe** command explicitly. If you wish to execute UNIX commands that are piped to one another, use Z-Mail's **sh** command.

Specifying Multiple Commands

You can specify more than one Z-Mail command on the same line by separating the commands with a semicolon (**;**). Z-Mail processes semicolon separated commands one-at-a-time as though you typed them on two separate lines.

How Z-Mail Processes Commands

Before executing your commands, Z-Mail preprocesses them in a style similar to the C-shell (*csh*). As you enter characters at the command-line prompt, Z-Mail parses each line and expands history references, command line aliases and variable references. When Z-Mail encounters the end of the line, a pipe (|) or semicolon (;), it assumes it has reached the end of a command and executes the command.

After parsing your input, Z-Mail searches for command names and predefined function names. If Z-Mail finds one of these commands or functions, Z-Mail executes it and passes the remainder of your input as command arguments.

If Z-Mail does not find a command that matches the one you specify, then Z-Mail checks the **unix** variable. If the **unix** variable is set, Z-Mail attempts to run your input as a UNIX command. If the **unix** variable is not set, or if Z-Mail cannot find the UNIX command in the path indicated in the *PATH* environment variable, Z-Mail displays a message that the command was not found.

Tilde-Escape Editing Commands

While composing a message in the command-line interface, if you enter a line that has a **~** character in the first column on your screen, Z-Mail treats the line as a *tilde-escape command.*

Tilde-escape commands are a special set of commands that let you edit and save your outgoing message while you are composing it. Z-Mail's GUI and Lite interfaces provide the capabilities of these tilde-escape commands as menu selections.

The remainder of this section explains the various tilde-escape commands you can use in Z-Mail's command-line interface while you are composing messages.

Modifying Headers

~t [*tolist*]

Use **~t** to modify the list of recipients in the *To*: list. If you specify a list of addresses in ***tolist***, Z-Mail appends this list to the current *To:* list. If you do not specify ***tolist***, Z-Mail displays the current list and places the cursor at the end of the list. You can then backspace over the addresses in the list or append more addresses onto the end of the list as you wish.

EXAMPLE

In the following example, you begin composing a message and then realize you wanted to send your message to an additional address.

swilson (#20 of 22) > **mail**
To: **tjohnson gstewart**
Subject: **Offsite meeting**

The offsite meeting to finalize next quarter's development schedule will be next Wednesday at Provincial Suites starting at 8:30.Terri Johnson will present the status of the A/P system, Kim Fong will
~t
To: tjohnson gstewart **kfong**
(continue editing letter)
bring us up to date on the inventory system, and Guy Stewart . . .

~b [*bcclist*] Use **~b** to modify the list of blind carbon copy recipients in the *Bcc:* header of your outgoing message. If you specify addresses in ***bcclist***, Z-Mail appends the addresses to your existing *Bcc:* list. If you do not specify ***bcclist***, Z-Mail displays your existing *Bcc:* header so you can edit it.

EXAMPLE The following example displays the blind carbon copy list so you can edit it or add to it as you are entering the text of an outgoing message.

> **. . .**
> **without making the necessary changes. We can modify the other**
> **~b**
> Bcc: pvolk **bgreene**
> (continue editing letter)
> **systems later to reflect the new specifications. . .**

~c [*cclist*] Use **~c** to modify the list of carbon copy recipients in the *Cc:* header of your outgoing message. If you specify addresses in ***cclist***, Z-Mail appends the addresses to your existing *Cc:* list. If you do not specify ***cclist***, Z-Mail displays your existing *Cc:* header so you can edit it.

EXAMPLE The following example adds a name to the carbon copy list as you are entering the text of an outgoing message.

> **. . .**
> **the publishing deadline. If you hurry and send in your materials, you may get them to Rick Gomez in time. Otherwise, you should send**
> **~c rgomez**
> (continue editing letter)
> **them via air courier to the conference in Paris. . .**

~s [*subject*] Use **~s** to modify the subject of your outgoing message. If you specify ***subject***, Z-Mail replaces your existing subject with the new one you specified. Note that this differs from the behavior of other tilde-escape commands that display, and allow you to edit, the existing contents of headers. If you do not specify ***subject***, Z-Mail displays your existing *Subject:* header so you can edit it.

> swilson (#20 of 22) > **mail**
> To: **tjohnson gstewart**
> Subject: **Offsite meeting**

The offsite meeting to finalize next quarter's development schedule will be next Wednesday at Provincial Suites starting at 8:30. Please
~s
Subject: Offsite Meeting **next Wednesday at 8:30**
(continue editing letter)
arrange for your own transportation . . .

~h Use **~h** to display and edit the *To*, *Subject*, *Cc*, and *Bcc* headers. Z-Mail displays each header one at a time so you can edit it. If you don't want to change a particular header, press **Return** instead of editing it.

swilson (#20 of 22) > **mail**
To: **tjohnson gstewart**
Subject: **Offsite meeting**

The offsite meeting to finalize next quarter's development schedule will be next Wednesday at Provincial Suites starting at 8:30.Terri Johnson will present the status of the A/P system, Kim Fong will
~h
To: tjohnson gstewart **kfong**
Subject: Offsite meeting **is Wednesday at 8:30**
Cc: **<Return>**
Bcc: **<Return>**
(continue editing letter)
bring us up to date on the inventory system, and Guy Stewart . . .

Viewing and Editing Message Text

~e [*editor*] Use **~e** to start the text editor indicated by ***editor***. If ***editor*** is not supplied, the value of the **editor** variable, environment variable *EDITOR*, or *vi* is used instead. You may enter **~e** at the beginning of any line in the text of an outgoing message.

Z-Mail starts an edit session and loads the text of your message into the editor. If the **edit_hdrs** variable is set (see page 273) then the messages headers are loaded as well. When you are finished editing, save your file and Z-Mail returns you back to the outgoing message screen.

~p [*pager*] Use **~p** to display the headers and text of the outgoing mail message. If the message is longer than one screen, Z-Mail pages the message using the ***pager*** program specified. If you do not specify ***pager***, Z-Mail uses the pager program indicated by the **pager** variable, the *PAGER* environment variable, or the default pager set up by your system administrator. If Z-Mail can't find any of these pagers, it uses its internal pager.

~u As you are composing outgoing messages and you notice a mistake in the previous line, you can edit the line without having to start an external editor by entering **~u** and pressing **Return**. Z-Mail displays the previous line and places the cursor at the end, allowing you edit the line.

Note for UNIX System V users: On some System V based systems, if the new, edited line is shorter than it was before the **~u** command, Z-Mail pads the line with blanks to preserve the previous length of the file.

EXAMPLE The following example shows how you use **~u** to edit the contents of line you most recently entered.

The staff meeting will be Monday at 8:30. <Return>
~u
The staff meeting will be Monday at **9:30.**

~v [*visual-editor*] Use **~v** to start the visual editor indicated by the value of the **visual** variable,*VISUAL* environment variable, or *vi.* You may enter **~v** at the beginning of any line in the text of an outgoing message. Note that you may set **visual** to one editor, and the **editor** variable to another.

Z-Mail starts an edit session and loads the text of your message into the editor. When you are finished editing, save your file and Z-Mail returns you back to the outgoing message screen.

~E Use **~E** to tell Z-Mail to save the contents of the outgoing message immediately in the dead letter folder indicated by the value of the **dead** variable. Z-Mail then starts a new outgoing message using the existing message headers.

~E! Use **~E!** to tell Z-Mail to discard the contents of the outgoing message and start a new message.

Adding Signatures

~S Use **~S** to append a signature to the end of the outgoing message. Z-Mail evaluates the **autosign** and **autosign2** variables to determine the text of the signature to append to the message.

EXAMPLE To add a signature at the end of your message, use **~S**.

```
~S
Adding signature at end of message.
(continue editing letter)
```

~S! Use **~S!** to stop Z-Mail from appending a signature to the end of the outgoing message if you have **autosign** or **autosign2** set to append one by default.

For more information on signing messages, see the **autosign** variable on page 259 and the **autosign2** variable on page 260.

~F Use **~F** to append a fortune to the end of the outgoing message.

~F! Use **~F!** to stop Z-Mail from appending a fortune to the end of the outgoing message if you have the **fortune** variable set to append one by default.

For more information on appending fortunes to messages, see the **fortune** variable on page 278.

Including Other Text Files

~i [*messagelist*] Use **~i** to include the body text, but not headers of other message(s) in an outgoing message. If you specify a ***messagelist***, Z-Mail includes the bodies of all the messages you list. If you do not specify ***messagelist***, Z-Mail includes the current message.

Z-Mail marks the included message by indenting it, as well as preceding and following it with descriptive text. For details on how Z-Mail marks included messages, see the **indent_str** variable on page 283, the **pre_indent_str** variable on page 300, and the **post_indent_str** variable on page 300.

EXAMPLE The following example shows how to include the text of messages 8, 9, 10 and 14 in your outgoing message.

To: **kfong**
Subject: **Survey Responses**

Here are the responses I received from the survey.
~i 8-10 14
Including message 8 . . . (24 lines)
Including message 9 . . . (37 lines)
Including message 10 . . . (18 lines)
Including message 14 . . . (21 lines)
(continue editing letter)

~I [*messagelist*] The **~I** tilde-escape command functions the same as **~i**, but also includes the message headers of included messages in your outgoing message.

~f [*messagelist*] Use **~f** to include entire messages in the body of an outgoing message (also called *forwarding*). If you specify ***messagelist***, Z-Mail includes all the messages you list. If you do not specify ***messagelist***, Z-Mail includes the current message.

Z-Mail marks forwarded messages by preceding and following them with a line saying Forwarded mail. Z-Mail does *not* indent forwarded messages in the body text of the outgoing message.

~r *filename* Use **~r** to read the read a text file called ***filename*** into your outgoing message. Z-Mail includes the text of the file as if you had typed it all on your keyboard, and does not indicate in any way that the text has been loaded from a file. Tilde characters that appear in ***filename*** are not treated as tilde-escape commands.

EXAMPLE The following example reads the file *surveytext* into your outgoing message.

To: **sales**
Subject: **Competition survey**

Please take five minutes to fill out this survey on your latest competitive situations so we can prioritize our product development and marketing plans. Thanks for the help.

~r surveytext

49 lines
(continue editing letter)

Processing Attachments

~A *filename*

Use **~A** to attach files to outgoing messages. The **~A** tilde-escape works similarly to the **–attach** option in the **zmail** command, prompting for additional information such as a short description, encoding and decoding instructions.

EXAMPLE

The following example attaches a FrameMaker file to the outgoing message.

To: **khagen**
Subject: **Chapter 7 draft**

Kevin -
I have attached the FrameMaker file of the second draft of Chapter 7. Please call me with any questions or comments. Thanks.
~A chap7.doc
Attachment type (? for list): **framemaker**
Attachment description: **Chapter 7 second draft**
Encoding (? for list): **btoa**
(continue editing letter)

~A

Use **~A** to list the numbers and descriptions of all attachments for the currently outgoing message. You can use this information to check that you have properly attached documents you are sending, or to check message numbers so you can delete a specific attachment (see **~A!** below to delete attachments).

EXAMPLE

The following example lists all attachments to the outgoing message.

. . . and I've included all the files you need to finish the project.
~A

Part	**Name**	**Type Key**	**Encoding**	**Size**	**Description**
1	**chap7.doc**	**frame**	**btoa**	**(pending)**	**Chapter 7 Second**
2	**hiho.c**	**text**	**(none)**	**(pending)**	**himem module**
3	**filler.c**	**text**	**(none)**	**(pending)**	**buffer module**

(continue editing letter)

~A! *part* Use **~A!** to remove a specific attachment from the currently outgoing message. If you do not specify ***part*** to indicate which attachment to detach, Z-Mail prompts you for a part number or name.

EXAMPLE The following example deletes an attachment from the outgoing message.

```
. . . and I'll send the himem module later.
~A!
Part Name      Type Key  Encoding Size      Description
1    chap7.doc frame         btoa (pending) Chapter 7 Sec-
ond
2    hiho.c    text        (none) (pending) himem module
3    filler.c  text        (none) (pending) buffer module
Remove part (number or name): 2
(continue editing letter)
```

Storing Messages

~w *file* Use **~w** to write the outgoing message to a file named ***file***. If the **edit_hdrs** variable is set or if you included the **–E** option in the **mail** command that started the current composition, Z-Mail includes the headers of the outgoing message in ***file***. Otherwise, Z-Mail writes only the text of the outgoing message to ***file***.

You can use files that include headers as *draft files* — partially composed messages that you can edit and send later. You can read files without headers into outgoing messages later using the **~r** command explained above.

WARNING: The **~w** command overwrites the existing contents of ***file***. To append to ***file*** without destroying its contents, use the **~a** command below.

~a *filename* Use **~a** to append the text of the composition to the file named ***file***. The **~a** command works like the **~w** command, but appends to ***file*** instead of overwriting its contents. You can use the **~a** command to keep a log of selected outgoing messages, or as a clipboard that allows you to retrieve text from messages you composed earlier.

~l Use **~l** (lowercase L) to tell Z-Mail to record the entire message to the folder specified by the **record** variable when the message is sent.

~l! Use **~l!** to tell Z-Mail not to perform message recording for the current message. This does not effect the action of **record_users** (see page 305).

~L Use **~L** to tell Z-Mail to record the headers of the message to the folder specified by the **logfile** variable when the message is sent.

~L! Use **~L!** to tell Z-Mail to not perform message logging for the current message.

Exiting Compose Mode

~q Use **~q** to quit the outgoing message and save it in the dead-letter file specified by the **dead** variable. If the **nosave** variable is set, Z-Mail quits without saving the outgoing message.

~z Use **~z** to suspend the outgoing message and return control to the command-line interface. When you suspend a message, Z-Mail neither sends it nor discards it. To resume the outgoing message later in your Z-Mail session, use the **resume** command. To display a list of suspended messages, use the **jobs** command.

EXAMPLE The following example suspends the outgoing message, sets the value of a variable, then resumes the message.

```
. . . the purpose of the meeting. If we can't come to some conclu-
sion quickly, we should implement the system as it has been
designed.
~z
[1]   Stopped.
```

psteele: #24 of 31 > **set record**
psteele: #24 of 31 > **resume**
(continue editing letter)
We can always add the reporting module later.

For more information on suspending messages, see the **resume** command on page 195 and the **jobs** command on page 169.

~x Use **~x** to immediately exit the outgoing message without saving its contents in the dead letter file.

EXAMPLE The following example terminates the outgoing message, and returns you to the Z-Mail prompt.

. . . the purpose of the meeting. If we can't come to some conclusion ~x
psteele: #24 of 31 >

Running Commands

~:command Runs ***command*** as a Z-Mail command. You cannot run any command that sends mail. (Use **~z** to suspend the current outgoing message if you wish to compose another message.) Be careful changing folders with **~:folder** since it invalidates any current lists of message numbers.

EXAMPLE The following example displays a list of new mail messages, and then resumes editing the outgoing message.

. . . just in time for the quarterly review meeting. I am planning on
~:headers :n

5	!	Shawna Wilson	Jul 18	8:31	Reports Due Now
6	!	Terri Johnson	Jul 18	10:02	Version 3.1 is ready

(continue editing letter)
presenting our latest designs on the Firefox project. If you have . . .

~ | command* (tilde-vertical bar)** Runs the UNIX command specified by ***command. Z-Mail passes the text (not including headers) of the outgoing message to ***command*** as input, and replaces the outgoing message text with the output of ***command***.

Since both the standard input and standard output of ***com-***

mand are connected to Z-Mail, interactive commands are not generally usable in ***command***.

EXAMPLE

The following example passes the text of the outgoing message to a program that checks for spelling mistakes and passes back a marked-up text file indicating any errors. Z-Mail loads the corrected text as the new outgoing message.

. . . and thanks again for all your help.
/ David/
~|spellchk
(continue editing letter)

Note: The program *spellchk* is only an example. Your system may have a different program that checks spelling; ask your system administrator.

Substituting Text Strings

~~

Use the **~~** command to begin a line with a tilde character. When Z-Mail encounters tilde characters in screen columns one and two, it inserts a single **~** into the outgoing message.

~$variable

Insert the value of **variable** into the message. If you specify a boolean-valued variable, Z-Mail inserts nothing.

EXAMPLE

The following example inserts the value of the **hdr_format** variable in the text of the outgoing message, then uses the **~u** tilde-escape command to display the line that Z-Mail inserted in response to the **~$** command.

To: **tjohnson**
Subject: **Formatting your Z-Mail header summaries**

Here's the formatting string I use as the value of the hdr_format variable:
~$hdr_format
(continue editing letter)
~u
%19n %M %–2N %6T %25s

Using Macros

Z-Mail provides two types of macros that help you manage your mail more quickly:

- Command macros created with the **map** command.
- Composition macros created with the **map!** command.

Command macros work on the command line, and composition macros work when you compose messages. Macros created with **map** and **map!** work only in the command-line interface. Command and composition macros don't work when you start Z-Mail with the **–echo** (**–E**) option.

Z-Mail Lite has its own mechanism for macros. See the online help for the **multikey** and **bindkey** commands for more information.

How Macros Work

Key Sequences and Expansions

Macros have two parts: a ***key sequence*** and an ***expansion***. The ***key sequence*** is the character or string of characters that Z-Mail watches for as you type. When you type the ***key sequence***, Z-Mail runs the macro. The ***expansion*** is the string of keystrokes that Z-Mail actually executes when it recognizes the ***key sequence***.

Macros are like an interactive search-and-replace function: if you type a recognized key sequence, Z-Mail substitutes the associated expansion in its place. For example, if you create a macro whose key sequence is **Ctrl-X Ctrl-W** and whose expansion is the string save, then when you hold down the **Ctrl** key and type **x** and **w**, the effect is as if you had actually typed the four characters **s, a, v,** and **e**. This process is called *expanding* the macro.

Key sequences are often made up of control characters or special strings of characters generated by *function keys*, but can be almost any string you desire.

Keys that generate a signal from the keyboard (for example, on BSD systems, **Ctrl-Z** generates a *stop* signal and **Ctrl-C** generates an *interrupt*) cannot appear in key sequences, and macros in line or composition modes cannot begin with a *newline*, **Ctrl-D**, or any of the editing keys (*erase*, *word-erase*, *line-erase*, etc.).

Key sequences for command-line and composition macros should be unusual characters or combinations of characters, and not individual lower-case letters. If you use common characters or strings for key sequences, you could inadvertently execute macros while typing the text of commands and messages.

Keystroke Timeouts

As you type in commands or the text of messages, Z-Mail uses a timeout to determine whether the keys you have pressed are a key sequence. In other words, once you type the first character of a key sequence, you must type the succeeding characters relatively quickly, or Z-Mail assumes you didn't mean to invoke a macro with a key sequence. For this reason, you should keep key sequences short, or you should automatically generate them by pressing a special function key on your keyboard.

Macros Are Recursive

Macros are recursive, so if the key sequence that invokes one macro appears in the expansion of another macro (or even of the same macro), Z-Mail recognizes the second key sequence as it expands the first macro, and then Z-Mail expands the second sequence as well. Be careful that you do not define recursive macro expansions that continue to call each other. If you do so, Z-Mail will freeze and eventually quit.

You can prevent Z-Mail from expanding key sequences by using *literal-next characters* from the keyboard or in expansions. In either case, they prevent the character following them from being interpreted as part of a key sequence.

Note for advanced users: Z-Mail recognizes the literal-next character from the *tty* settings of your terminal or workstation, provided you have a newer BSD-style terminal device driver. Otherwise, Z-Mail recognizes **Ctrl-V** as the *literal-next* character. If you have a *tty literal-next* character, you need to type two of them sequentially to send one to Z-Mail; this is because the *tty* driver consumes the first one. It is not necessary to use two *literal-nexts* in macro expansions unless you wish to cause the second *literal-next* to be literal. In addition, you can use a backslash as a *literal-next* when you type, and sometimes as a *literal-next* in expansions; but use it with caution, because it is also the beginning of some escape sequences.

Errors Abort Macros

A macro always aborts whenever any command called by the macro returns an error. This includes recursive expansions, so no matter how often a macro has recurred, it is terminated completely when it encounters an error.

Specifying Control Characters

Z-Mail supports a special syntax for specifying control characters and other non-printing characters in macro key sequences and expansions. This syntax uses the backslash character and is the same as the syntax for specifying key bindings.

If you want to define a key sequence that includes control characters, specify the **Ctrl** key using the **^** character. For example, to define **Ctrl-K** as a key binding, you specify it as **^K**.

The table below summarizes the keys you must enter to indicate special keys when you are defining macros.

To indicate a:	*Enter:*
Control	**^ or \C**
Return	**\n**
Tab	**\t**
^	**\^**
Esc	**\E**

Thus, to create a command-line macro for **Ctrl-X Ctrl-W**, as in the example above, you enter the command:

```
map '^X^W' save
```

If you have an older terminal that does not have a **^** key, use **\C** instead. Therefore, to indicate **Ctrl-P**, you must enter **\CP**.

Prompting for Input within Macros

Z-Mail also provides a method for executing functions from within macros that request input from the user. Two special functions — **getstr** and **getline** — interrupt expansion of the current macro and wait for the user to enter a string at the keyboard. Z-Mail then inserts this string into the macro expansion.

***getline* function**

The **getline** function retrieves the input string and retains the *newline* character (**Return**) at the end of the input string.

***getstr* function**

The **getstr** function retrieves the input string and strips off the *newline* character (**Return**) at the end of the input string.

You tell Z-Mail to execute these functions by surrounding their names with square brackets. Make sure to include no white space inside the brackets. For example:

```
map '^X^W' save [ getline ]
```

creates a command-line macro that runs when you type **Ctrl-X Ctrl-W**, and which displays **save** followed by a space and then waits for you to type a line of input. Z-Mail then uses the line you type as arguments to the **save** command.

Command-Line Macros

Command-line macros combine the convenience of single-keystroke commands with the versatility of the command-line interface.

EXAMPLES

If you frequently forward messages to the e-mail address *fred*, you can create a macro to do it with the following command.

map '^F' 'mail –f . fred\n'

This macro causes the single keystroke **Ctrl-F** to forward the current message to the e-mail address *fred*. The *newline* character **\n** at the end of the expansion causes the command to be executed immediately, without requiring you to press **Return**. Z-Mail displays the expansion part of a command-line macro on your screen as Z-Mail expands it, so you can see what the macro is doing. You can therefore use parts of the expansion as a prompt.

In the above example, you can prompt for a message list rather than always forwarding the current message by changing the macro to:

map '^F' 'mail –f [getstr] fred\n'

This macro displays *mail –f* and a space, then waits for a you to enter a string at the keyboard. After you press **Return**, the **getstr** function strips the *newline* character and passes the string as the message list to the **mail** command. The macro also passes the address *fred* to the **mail** command, forwarding the messages to *fred*.

Making Changes to Macro Input

If you want to be able to change your mind after starting a command-line macro, you must leave the **\n** out of the expansion. Without the *newline*, Z-Mail waits until you type **Return** before executing the macro, so you have a chance to erase or change the command. You can never backspace past the beginning of a string retrieved by the **getline** function, and you can backspace past the beginning of a string retrieved by **getstr** only after the **getstr** function has completed its work.

Composition Macros

Composition macros are typing-accelerators that work while you compose messages. Composition macros are defined like command-line macros, except you use the **map!** command (note the exclamation point) instead of **map**.

EXAMPLES

To define a macro that inserts the word percent into a message whenever you press the **%** key, enter the following command.

```
map! % percent
```

Another common use of composition macros is to perform regular indenting at the beginning of message lines. The following macro inserts four spaces into your message whenever you type **Ctrl-T**.

```
map! '^T' '    '
```

Invoking Tilde-Escapes with Macros

You can also use composition macros to execute tilde-escape editing commands. For example, you could create a macro to invoke an external text editor:

```
map! '^E' '\n~v\n'
```

When you type **Ctrl-E**, this macro types a *newline* (to ensure that the tilde is the first character on a line), then types **~v** followed by another *newline*, to start the editor. For more information, see *Tilde-Escape Editing Commands* on page 91.

Mixed-Mode Macros

Normally, you create macros to run in either command-line or composition mode. You can, however specify an expansion in a macro that switches between modes. For example, the command:

```
map! '^E' '\n~v emacs\n'
```

defines a composition macro that starts the *emacs* editor when you enter **Ctrl-E** while composing a message. Once you have defined this macro, you can create a command-line macro that calls it:

```
map '#' 'reply –i [ getline ] ~t [getstr]\n^E'
```

Now when press the **#** key in command-line mode, this macro displays reply –i and waits for a message list, then starts composing a reply. The macro displays the To: line (the **~t** escape) and waits for you to add other addresses. Finally, the **Ctrl-E** macro is expanded, to start editing the message with *emacs*.

Note: You should be careful to avoid accidentally expanding recursive (self-referencing) macros, especially when you use macros that change modes. When you test new macros, it is a good idea to start Z-Mail with your active folder in read-only mode (**zmail –r**) to be sure you do not lose or alter messages unexpectedly.

Deleting Macros

You do not have to delete a macro in order to redefine it. When you enter a command that specifies a new expansion for a previously defined key sequence, the new expansion automatically replaces the old one.

To remove the definition of a macro, use the following commands:

Use the command:	***To delete:***
unmap	Command-line macros
unmap!	Composition macros

Remember to use a backslash (\) or other *literal-next* character to prevent the expansion of macros when you use these commands, especially **unmap**.

CHAPTER 5

Z-Script Commands

This section describes Z-Script commands you can type at the Z-Mail command prompt, in the command entry field in the GUI and Lite interfaces, or within functions you create using the Z-Mail function editor or standard text editor. You can use these commands to build scripts that operate on messages and folders. Using **function** and **cmd**, you can build your own functions and commands that add new functionality to Z-Mail and perform repetitive actions.

You can also use scripts and functions to customize Z-Mail so it resembles other e-mail products. Z-Mail is pre-configured to provide many commands that are compatible with other line-oriented e-mail products, including all the commands in basic UNIX mail. This allows you to leverage your prior knowledge while you become familiar with the advanced capabilities of
Z-Mail.

This section provides an alphabetical list of available commands, with syntax summaries and some examples. See the *Z-Mail User Guide* for additional information on writing scripts.

Also see the sample scripts provided in the Z-Mail library (usually */usr/lib/Zmail*), including *sample.lib* and *sample.zmailrc*. These are heavily commented files that demonstrate Z-Script programming concepts and technique.

Command Summary

This section summarizes Z-Script commands in functional categories, in the following order:

- Help Commands
- Message Commands
- Folder Commands
- Composition and Addressing Commands
- User Interface Commands
- Programming Commands
- UNIX and Z-Mail Control Commands
- Configuration Commands

For details on individual Z-Script commands, see the section *Command Reference* beginning on page 117.

Help Commands

?	Displays a list of valid Z-Mail commands.
about	Displays information about Z-Mail.
help	Displays help information on various following topics.
version	Displays the version of Z-Mail.

Message Commands

delete	Marks messages for deletion.
detach	Stores one or more message attachments and optionally displays those attachments.
display, type, print	Displays a message. (**lpr** prints messages on a hardcopy printer.)
edit	Opens messages so you can edit them using the **editor** you specify.
flags	Examines or sets the status flags of messages.

from	Displays information taken from the headers of your message list.
headers	Displays a screenful of message headers from messages in the active folder.
hide	Flags messages so they are not displayed in message summary lists.
lpr	Sends messages to the hardcopy printer.
mark	Tags messages with priorities and temporary marks.
next	Displays the contents of the next message, or the current message if it has not yet been read.
pinup	Displays a message in a new message reading window.
preserve	Saves messages to your system folder rather than your main mailbox unless you explicitly delete them.
previous	Displays the contents of the previous message.
search, pick	Searches for and selects messages according to specific criteria.
sort	Sorts the messages in the active folder according to a variety of criteria.
top	Displays the top lines of messages so you can review them quickly in the command-line interface.
undelete	Reverses the effect of a **delete** command.
undigest	"Bursts" specially-formatted "digest" messages into their constituent messages.
unhide	Allows previously hidden messages to be displayed.
unmark	Removes priorities and temporary marks from messages.
unpreserve	Stops Z-Mail from saving messages to your system folder rather than your main mailbox unless you explicitly delete them.

Folder Commands

copy	Copies messages to another folder without marking them for deletion in the source folder.
close	Closes mail folders.
folder	Changes the active folder, opens a folder, or closes a folder.
folders	Lists the folders that are present in the directory identified by the **folder** variable. In the GUI interface, the **folders** command is the same as **dialog folders**.
merge	Copies messages from a folder you specify into your active folder.
remove	Removes (deletes) a file or folder.
rename	Changes the name of a file or folder.
rmfolder	Removes (deletes) a folder, prompting for confirmation if a non-folder file is specified.
save	Saves messages to a file or folder.
update	Writes all changes that have been made to a folder (usually the active folder) to disk.
write	Saves the body text from messages to a file or folder, omitting message headers.

Composition and Addressing Commands

alias	Creates e-mail address aliases and distribution lists.
alternates	Tells Z-Mail about other e-mail addresses you may have.
compcmd	Perform a command on a composition.
expand	Shows the addresses referenced by a mail alias. Z-Mail fully expands nested or hierarchical aliases.
mail	Begins a new message composition.

reply Replies to a message you have received, addressing the response only to the sender of that message.

replyall Replies to the sender of a message and also to everyone who received a copy of the message.

unalias Deletes aliases and distribution lists.

User Interface Commands

bindkey In the Lite interface, binds a key sequence to initiate an interaction. Documented in the Lite on-line help.

button Creates or lists user-defined interface buttons.

cmd Creates or lists command abbreviations.

dialog Displays any of Z-Mail's standard dialog boxes.

iconify In the GUI interface, turns all Z-Mail windows into icons on the desktop. In Lite and command-line interfaces, stops Z-Mail and returns you to the shell.

map[!] Creates or lists macros for the command-line interface.

multikey In the Lite interface, tells Z-Mail what keystrokes your terminal's function keys generate. Documented in the Lite on-line help.

menu Creates user-defined menus and menu items.

redraw Redraws the screen and optionally selects messages in the GUI and Lite interfaces.

stty Emulates the UNIX *stty* command to set terminal parameters.

unbutton Removes user-defined interface buttons created with **button**.

uncmd Removes command abbreviations created with **cmd**.

unmenu Removes user-defined menus or menu items created with the **menu** command.

Programming Commands

ask	Asks a question and obtains a response from the user.
builtin	Calls a built-in Z-Mail command that has the same name as a user-defined function.
each	Executes a command you specify on each message in a message list.
echo	Re-displays the commands you type, expanding all arguments to show their values.
error	Works just like **echo**, but additionally sets the **status** variable to –1.
eval	Re-parses its arguments, then executes them as a command.
foreach	The **foreach** command iterates through a list of message numbers or filenames you specify, executing a command on each iteration.
function	Creates user-defined functions.
if ... else ... endif	Conditionally executes commands in a function.
interpose	Piggybacks a user-defined function on a Z-mail command or operation.
msg_list	Simplifies and selects a message list in metacharacter form to a compact list of numbers.
return	Terminates a function and resumes the routine which called the function.
shift	Shifts the arguments of a function for parsing.
task_meter	Brings up the Task Meter.
thwart	Prevents an interposed function from executing.
trap	Installs UNIX signal handling functions.
unfunction	Removes user-defined functions created with **function**.
uninterpose	Removes an interposer created with the **interpose** command.

UNIX and Z-Mail Control Commands

await	Instructs Z-Mail to stop functioning interactively and wait for new mail.
cd	Changes the current working directory.
chroot	Change the root directory.
debug	Sets the Z-Mail debugging-trace level to help you debug your scripts.
disable	Restrict the use of a Z-Mail command.
enable	Restore a previously disabled command.
exit	Terminates Z-Mail without updating any folders. In user-defined scripts, terminates the script.
history	Displays recent commands you have entered.
jobs	Displays a list of suspended message compositions.
ls	Lists the files in the current directory.
pipe	Sends the contents of selected messages to a UNIX command on its input.
printenv	Displays the set of environment variables that Z-Mail passes to UNIX processes it creates.
pwd	Displays the name of the current working directory.
quit	Terminates Z-Mail, after updating any open folders that have been modified.
resume	Reactivates a suspended message composition.
setenv	Sets environment variables.
sh	Executes a command using the Bourne shell (*/bin/sh*).
stop	In the command-line and Lite interfaces, suspends the Z-Mail session, returning control to the calling shell. In the GUI interface, **stop** iconifies Z-Mail session on the desktop.
unsetenv	Unsets environment variables.

x Terminates Z-Mail without updating folders.

Configuration Commands

attach Defines attachment-type information.

filter Creates or lists message filters.

ignore Tells Z-Mail which message headers to hide when it displays messages.

my_hdr Creates or lists custom headers that Z-Mail adds to outgoing messages.

retain Specifies the headers that Z-Mail should show when it displays messages.

saveopts Stores a copy of all of the current Z-Mail settings in a file.

set Sets the values of Z-Mail variables.

setenv Sets the values of environment strings that Z-Mail passes to other UNIX processes it creates.

source Executes Z-Script commands from a specified file.

unfilter Removes message filters.

unignore Tells Z-Mail to stop ignoring selected message headers.

un_hdr Removes custom headers that have been created with **my_hdr**.

unmap[!] Removes keyboard macros in the command-line interface.

unretain Stops Z-Mail from displaying certain headers when it displays messages.

unset Unsets Z-Mail variables and their values.

Command Reference

This section lists all of Z-Mail's Z-Script commands in alphabetical order. See the previous section for a summary of Z-Script commands by category.

?

The **?** command displays a list of valid Z-Mail commands, or help information about a ***command-name*** you specify.

SYNTAX **? [*command-name*]**

If you enter the **?** command with no arguments, Z-Mail displays a list of valid commands. If you specify a ***command-name***, Z-Mail displays help for that command.

about

The **about** command displays information about the creators of Z-Mail.

SYNTAX **about**

alias

The **alias** command allows you to define a name that stands for a list of mail addresses.

SYNTAX **alias [*aliasname* [*addresslist* | :*filename*:]]**

RELATED COMMANDS **unalias, expand**

RELATED VARIABLE **no_expand**

If you specify no arguments, **alias** displays all the current aliases. With only an ***aliasname*** argument, it displays the list associated with that alias. If you specify more than one argument, **alias** creates a new alias, with the first argument as the alias name and the following arguments the expansion of the alias.

addresslist may contain:

- Mail addresses, such as marti@forth.redmon.ca or newell!routed!thorious.
- Other aliases, which will be expanded at send time to the ***addresslist*** they contain.
- Filenames, which must begin with **/** or **~** or **+**. For compatibility with X.400 (RFC 1327 compliant) MTAs, entries in ***addresslist*** which start with **/** and contain at least one ' **=** ' character are *not* treated as filenames.
- Program names, which must start with a pipe character (|) and be enclosed in quotation marks. The **<** and **>** characters must not appear in program names (although they may surround the program name), or they will be treated as an RFC 822 literal-address and likely fail.

To specify a file name as the source of the addresses, surround it with colons. If the file name is not a full path, it is taken to be relative to your home directory.

EXAMPLES

You can use an alias instead of entering one or more user mail addresses when you are sending a message. For example, if you send mail regularly to Kim Fong at Genisoft Corporation, you may want to define the alias kim as a shorthand for her e-mail address:

alias kim fongk@genisoft.com

The sales manager at Genisoft may want to define an alias for all his direct reports:

alias direct fongk rtaylor ithacker vgomez ~/storage/reports

The direct alias defined above contains four user addresses and one filename.

The system administrator at Genisoft who has already defined aliases called sales and engineering may want to define an alias for the entire company, including the president fkramer:

alias everyone sales engineering fkramer

Use the **unalias** command to delete previously defined aliases.

alternates

Use the **alternates** command to inform Z-Mail of additional login names you have on various host computers to prevent Z-Mail from sending mail to you on other computers.

SYNTAX

alternates [*hostlist*] [*[*username*]] [*address*] [*!path!user*]

SYNONYM

alts

RELATED VARIABLES

metoo, alternates

When you have user accounts on several computers, other users may send messages to all your e-mail addresses hoping to find you the next time you read mail on any of the computers. If you **reply** to one of those messages, you may accidently send your reply to yourself at all your e-mail addresses. To prevent this, you can use the **alternates** command. When you specify your alternate e-mail addresses, Z-Mail does not send a copy of the reply to your login name at any of the hosts listed on the alternates list.

If you specify an asterisk (*) for ***username***, then Z-Mail attempts to match your login name with all pathnames and local addresses on the specified hosts. If you specify the asterisk followed by a ***username***, Z-Mail looks for that name instead of your login name in the addresses of the remote computers.

If you issue the **alternates** command with no arguments, Z-Mail displays your current set of alternate names. Names you entered in *user@domain* form are displayed in UUCP (*!domain!user*) format.

Note: The effect of an **alternates** command is not cumulative; the arguments you specify in the most recent **alternates** command removes your current list of alternate names and replaces it with the new ones you specify.

EXAMPLES

If you have another login name on the local or remote machine, you may specify the address either as *user@domain* or by preceding the login name (or its UUCP pathname) by an exclamation point (!). For example, you can issue the command:

alts hq1 hq2 jhundley@plan.genisoft.com !john

This command looks for the login name *jhundley* on the hosts *hq1* and *hq2*. The last item, !*john*, blocks mail to the e-mail address *john* only if it is destined for the local machine. Root accounts elsewhere are not considered to be equivalent. The address *jhundley@plan.genisoft.com* indicates that at the machine
plan.genisoft.com, the user *jhundley* is the same person as the current user on the local machine. You could also specify this in UUCP format:

alts !plan.genisoft.com!jhundley

You must specify the exclamation points (!) in the address to distinguish the name of the domain from the username that follows it.

ask

Asks for information from the user.

SYNTAX

ask [-noecho] [–input | –file | –list *variable*] [–default *"def-answer"*] [–must_match] *"question"* [*choices*...]

All options can be abbreviated to one letter. When the user answers the question, **ask** sets the variable **status** to:

- 0 (zero) if the user makes a choice or selects Yes
- 1 if the user does not make a choice or selects No
- –1 if the user selects Cancel

GUI and Lite

In the GUI and Lite interfaces, **ask** presents a dialog box where the user answers the ***question***. The contents of the dialog depend on the options specified.

ask "*question*" presents a Yes/No/Cancel dialog with the ***question***. If **–default { yes | no | cancel }** is specified, the corresponding button is highlighted as the default button. **Yes**, **no**, and **cancel** in ***def-answer*** may be abbreviated to one letter. The variable **status** returns the user's choice as described above.

ask –input *variable* "*question*" presents an OK/Cancel dialog with the question and a single-line text entry area. If **–default *def-answer*** is specified, then ***def-answer*** will be placed in the text entry area. If a list of ***choices*** is specified, the dialog will also include a scrolled list with the choices, one per line, from which the user may choose. If **–must_match** is also specified, the user must choose one of the ***choices*** or Cancel. The string the user enters will be placed into ***variable***, and **status** will be set as described above.

ask –file *variable* "*question*" works like **–input**, except the dialog contains a File Finder instead of a list of choices. The initial directory is the current working directory, which you may change with the **cd** command before performing the **ask** command.

ask –list *variable* "*question*" presents the message summaries of the currently selected messages in a scrolled list, from which the user may choose one or Cancel. The **–default**, **–must_match**, and ***choices*** arguments are ignored. The message number of the message summary the user selects will be placed into ***variable***.

If the **-noecho** option is specified, the user's answer is not echoed back to the screen.

Command-line

The user's response to the **ask** prompt must be a single argument; responses that contain spaces must be enclosed in quotes by the user.

ask "*question*" presents a Yes/No/Cancel prompt with the ***question***. You may set the default answer as described for the GUI interface, above. Only the first character of the user's response is examined. Any character other than Y, y, N, or n is considered a Cancel response. The interrupt character (usually **Ctrl-C**) is also considered a Cancel response.

ask –input *variable* "*question*" displays the ***question*** and waits for a response. The user is not permitted to backspace over the prompt. If **–default *def-string*** is specified, then ***def-string*** will be placed after the prompt. If a list of ***choices*** is specified, the choices will be presented as a numbered list, one per line, from which the user may choose. The first ***choice*** is used as the default if one is not specified with **–default**. The user may make a selection by typing the number of their choice, or type a different response. If **–must_match** is also specified, the user must choose one of the choices or Cancel. The string the user enters will be placed into ***variable***, and **status** will be set as described above.

Questions with numeric answers must be answered by selecting from the list of choices; the user cannot type their numeric response directly.

ask –file *variable* "*question*" works like **ask –input** in the command-line interface.

ask –list *variable* "*question*" works like **ask –input** in the command-line interface.

EXAMPLES

To prompt for a text string and assign it to the **folder** variable:

ask –i folder "Enter the name of your folder directory:"

To ask for text input and assign the input to the **realname** variable:

ask –i realname "What is your real name?"

To ask a question and specify yes as the default answer:

ask –d y Continue?

To specify a list of choices:

ask –i m_pri "View which message priority?" A B C D E

attach

Attach adds or modifies attachment processing information in a running Z-Mail. Note that **attach** does not create attachments.

SYNTAX

attach [–rehash]

attach [–load *file* | –save *file* | –merge *file*]

attach –path *directory*[:*directory*[:*directory*]...]

attach –type *keyword* "*viewer-program*" "*editor-program*"

attach –code *keyword* "*encoder-program*" "*decoder-program*"

attach –encode *type-key code-key*

attach –default *code-key*

RELATED FILE

See the *$ZMLIB/attach.types* file, which defines attachment processing information at Z-Mail startup.

RELATED VARIABLE

attach_types

The **attach** command with no options displays all of the attachment information Z-Mail currently is using, in *attach.types* format. See *Defining Attachment Types* beginning on page 328 for more information.

attach –rehash reads the *$ZMLIB/attach.types* file and all files specified in the **attach_types** variable, replacing Z-Mail's current attachment information.

attach –load reads the specified *attach.types*-formatted ***file***, replacing Z-Mail's current attachment information.

attach –save writes the current attachment information to the specified ***file*** in *attach.types* format.

attach –merge reads the specified *attach.types*-formatted ***file***, augmenting Z-Mail's current attachment information.

attach –path adds a list of directories to the attachment encoder, decoder, display, and view program search path.

attach –type defines ***viewer*** and ***editor*** programs for the specified ***keyword***.

attach –code defines ***encoder*** and ***decoder*** programs for the specified ***keyword***.

attach –encode defines which encoder (***code-key***) to use by default for a particular attachment type (***type-key***).

attach –default defines the default encoding to use for files that contain binary (8-bit) data.

await

The **await** command makes Z-Mail wait for new mail to arrive.

SYNTAX

await [–T *timeout*]

Command-line

The **await** command does not return until new mail arrives or until you type a keyboard interrupt (usually **Ctrl-C**). Z-Mail looks for new mail approximately every 30 seconds, or every ***timeout*** seconds as specified by the **–T** option.

GUI and Lite

In the GUI and Lite interfaces, **await** returns immediately if new mail has arrived or if no ***timeout*** is specified. Otherwise, Z-Mail waits ***timeout*** seconds before returning control to the screen.

builtin

The **builtin** command is useful when you have defined a function with the same name as an internal (built-in) Z-Mail command. By preceding a command name with **builtin**, you tell Z-Mail to execute the internal command, and not a user-defined function with that name that may exist.

SYNTAX

builtin *command* [*arguments*...]

RELATED COMMANDS

cmd, function

RELATED VARIABLE

recursive

If you do not specify **builtin** before ***command***, Z-Mail will preferentially execute the user-defined function called ***command*** if it exists.

EXAMPLE

cmd write 'echo "Not saving headers." ; builtin write'

Note: If you execute a Z-Mail command from within a user-defined function or command alias of the same name, make sure to precede it with **builtin**. If you do not use **builtin**, the function terminates with an error. Another way to avoid this is to set the **recursive** variable to true; the function may then call itself recursively. This applies only to functions, not to commands.

button

GUI and LITE ONLY

The **button** command creates or lists *buttons* defined in the GUI and Lite interfaces. Buttons created with **button** execute their associated Z-Script function when pressed.

SYNTAX

button [*location*] [*options*] [*button-name*] [*command*]

RELATED COMMANDS

unbutton, function

In the GUI and Lite interfaces, you use this command to create a user-defined button labeled ***button-name*** in the button panel area of the Main, Message, or Compose window. You may create *action buttons* which execute the Z-Script ***command*** or function when pressed, or *toggle buttons* which alter the state of Z-Mail variables or options.

You may also create and edit buttons using the *Buttons* dialog. See the *Extending Z-Mail* section of the *Z-Mail User Guide* for more information about this dialog.

If you specify no arguments, **button** displays all the current buttons and their associated function names in the order in which they appear in the Main Window, or the current context if you have specified one using the **-buttonlist-context** or **-window-context** options described below. If you specify only ***button-name***, Z-Mail displays the function name associated with that button.

Groups of buttons that appear together on a button panel are specified in *button lists*. There are three default button lists, one for each Z-Mail window.

Window	Button list name
main	MainActions
message	MessageActions
compose	ComposeActions

Using the **button** command, you may:

- add, remove, or change buttons on existing button lists
- create new button lists
- change the button lists associated with windows

Adding a new action button

To add a new action button, use this form of the **button** command:

button [*location*] [*options*] *button-name command*

location may be specified as either the name of the window in which the button appears, or as the name of the button list to which it should be added. The following are valid location options:

–window *windowname*

–window-context *windowname*

–buttonlist ***buttonlistname***

–buttonlist-context ***buttonlistname***

windowname may be either **main, compose,** or **message.** **buttonlistname** may be the name of any default button list (**MainActions, ComposeActions, MessageAction**s) or any user-defined button list.

When you specify a button list, the button will appear in all places where that button list is displayed. If you specify a window, the button becomes part of the button list currently in that window, so it appears in that window and in any other window containing that button list.

The **–window-context** and **–button-context** options are useful when you are specifying several buttons that all have the same location. When a **button** command containing one of these options is executed, all buttons created subsequently will appear in the same location, until another context is specified.

If no location is specified, buttons are added to the current context. (If you have never specified context, the default is the Main Window button panel).

options may be any of the following:

–no-msgs	if messages do not need to be selected in order for the button to execute
–position ***number***	specifies the position of the button; buttons are numbered left to right starting from 1.
–final	to indicate that this button should always have the highest position number. If more than one button in a button panel has this option, the most recently created one becomes the final button.
–sensitivity ***expr***	if the specified Boolean expression evaluates to false, the button is insensitive, or grayed out. If it evaluates to true, the button is sensitive.
–focus-condition ***expr***	

If the specified Boolean expression changes from false to true, the button gains focus. Focus refers to the selection rectangle that appears around an interface object, indicating that Motif keyboard shortcuts will act upon it. (You can change focus between objects using the TAB key.)

Focus and sensitivity ***expressions*** may take the following forms:

expr : : = expr1 && expr2	(expr1 AND expr2)
expr1 || expr2	(expr1 OR expr2)
(expr)	(parentheses may be used for grouping)
!expr	(expr1 is not true)
expr1 == expr2	(expr1 equals expr2
expr1 != expr2	(expr1 does not t equal expr2)
$variable, text-string, number	(expressions may also include Z-Mail variables, strings, and numbers.)

Expressions should be surrounded by single quotes.

button-name specifies the text that appears on the button. It may also be specified as **–label *button-name***. In either case, the button name should be surrounded by double quotes.

command specifies the Z-Script function or command that gets executed when the button is pressed.

Adding a new toggle button

To add a new toggle button, use the **button** command in the following form:

button [*location*] [*options*] –value *expr button-name* [*command*]

location, options, and ***button-name*** may be specified as described in the previous section.

–value *expr* indicates that the button is a toggle button, rather than an action button. When the expression evaluates to true, the button is toggled on; when the expression evaluates to false, the button is toggled off.

Value ***expression*** is restricted to one of the following forms, and should be surrounded by single quotes:

```
expr : : = A
         !A
A : := $variable              (returns contents of variable)
          $?variable           (retruns true only if variable is set)
          $variable:(value)    (returns value if variable is set to
                                value; returns null otherwise)
          $?variable:(value)   (returns true only if variable is set
to
                                value; returns false otherwise)
```

command is optional for toggle buttons; if it is specified, it will be executed when the button is pressed, after the state of the variable or option described by the value expression is altered.

Creating a new button list

To create action buttons in a new button list, use this form of the **button** command:

button –buttonlist *buttonlistname* [options] *button-name command*

To create toggle buttons in a new button list, use this form of the button command:

button –buttonlist *buttonlistname* [*options*] –value *expr button-name* [*command*]

You may use the **–buttonlist-context** option in place of the **-buttonlist** option in these commands.

Using a button list in a window

To install a button list in a window, use the button command in the following form:

button –window *windowname* –buttonlist *buttonlistname*

windowname may be **main, message,** or **compose.**

Use the **unbutton** command to remove buttons from any window or button list.

The **button** command has no effect in the command line interface.

EXAMPLES

The following command creates a button in the main window labeled "Quit" which executes the Z-Script **quit** function when pressed:

button –no-msgs –window main "Quit" quit

The following command creates a toggle button in the main window that toggles the display of the Output area in the Main window:

button –no-msgs –value '$?main_panes:(output)' "Output Area"

The following commands create a new button list called MyButtons, which contains two buttons:

button –buttonlist-context MyButtons "Quit" quit

button "Sign" compcmd –insert-file ~/.signature

The following command installs all the buttons in the MyButtons button list in the Compose window, replacing the buttons specified by the default ComposeActions button list:

button –window compose –buttonlist MyButtons

chroot

The **chroot** command changes the root directory to the specified directory.

SYNTAX

chroot [*directory*]

RELATED COMMANDS

cd

If you specify ***directory***, you will only be able to access files which have this directory as a prefix via file finder dialogs. If you do not specify ***directory***, the root directory (/) is used.

cd

The **cd** command changes the current working directory.

SYNTAX

cd [*directory*]

RELATED COMMAND

pwd

If you specify ***directory***, Z-Mail changes the current working directory to ***directory***. If you specify no arguments, Z-Mail changes the current working directory to your home directory.

If the ***directory*** argument is not a full pathname (i.e., it doesn't begin with **/**) and the pathname specified is not found in the current working directory, then Z-Mail checks the **cdpath** variable. If **cdpath** is set to a list of directories, then Z-Mail searches the **cdpath** directories in order, looking for the requested ***directory***.

To display the pathname of the current working directory, use the **pwd** command.

close

The **close** command closes an open folder.

SYNTAX

close [–n] [–x | –X] [%[*user*] | #[*n*] | *foldername*]

RELATED COMMANDS

folder, open, update

Options to **close** are:

- **–n** close without updating (implies **–X**).
- **–x** force the creation of an index when closing.
- **–X** do not create an index when closing.

The folder to **close** can be specified by:

- **%** means your system folder
- **%*user*** means the system folder belonging to the specified ***user***.
- **#** means the most recently accessed folder that is not active.
- **#*n*** means folder number ***n***. Folders are numbered in the order they are opened, starting with 0.
- ***foldername*** closes the folder named ***foldername***.

The **close** command is equivalent to the **folder –d** command.

To open a folder, use the **open** command, the *Folder Manager* dialog, or the *Add Folder* dialog.

cmd

The **cmd** command defines aliases for Z-Mail command names in all Z-Mail interfaces.

SYNTAX

cmd [*name* [*command*]]

RELATED COMMAND

uncmd

When you specify ***name*** and ***command***, Z-Mail creates a command-line alias called ***name***. When you type ***name*** at the command line, Z-Mail executes the Z-Script command called ***command***. You should enclose ***command*** in quotation marks if it contains semicolons or pipe symbols (**|**) to prevent them from being interpreted at the time you issue the **cmd** command.

If you specify no arguments, **cmd** lists the current command-line aliases. If you specify only ***name***, **cmd** shows the command associated with that alias.

See *Command Aliases* on page 87 for more information about **cmd** and using *csh*-style command history references in ***command***.

EXAMPLE

This command defines a command alias check which displays the message Checking for mail... and waits for mail.

cmd check 'echo Checking for mail...; await'

compcmd

The **compcmd** command performs a command on a composition.

SYNTAX

compcmd [%*job*] *cmd* [:off] [*arguments*]

cmd may be one of the following:

append-line	Append text and a newline to message body.
append-text	Append text to the message body.
attach-file	Attach a file of arbitrary type
blind-carbon	Add Bcc recipients
cancel	Cancel the message

carbon-copy	Add Cc recipients
directory-check	Pass all addresses through the program defined by the **address_book** variable.
edit-message	Call up the external editor defined by the **editor** variable or the **EDITOR** environment variable.
erase	Erase the body of the message, leaving the headers intact.
forward-message	Append a message from the active folder to the message body.
get-header	Set the header_value variable to the current value of the indicated header
include-message	Append (indented) a message from the active folder into the message body.
insert-file	Append a file to the message body
insert-header	Add any header
kill	Cancel the message, after prompting for confirmation
log	Activate outgoing header log
pipe	Pass the message body through a shell command.
preview	Preview an attachment.
record	Activate the outgoing message record
return-receipt-to	Request a return-receipt
save-to-file	Append the message body to a file
send-message	Send the message
sort-addresses	Alphabetize each address header.
subject	Set the subject of the message
to	Add To recipients
transport-via	Set the transport agent.
use-signature	Activate autosign.
write-to-file	Write the message to a file, overwriting an existing file of the same name.

job is the job number of a suspended composition, as displayed by the **jobs** command. If you do not specify a job number, the most recently suspended composition is acted upon.

Use **:off** in conjunction with a **cmd** value to disable that feature or remove a header.

EXAMPLE

For example, the command

compcmd %1 append-line "P.S. See you soon!"

appends the line

P.S. See you soon!

to the first suspended composition.

copy

The **copy** command functions like the **save** command except that it does not mark messages for deletion, regardless of the value of the **keepsave** variable.

SYNTAX **copy [–f | –s | –S | –a | –A] [*messagelist*] [*filename* | *directory*]**

SYNONYM **co**

RELATED COMMANDS **save, write**

RELATED VARIABLE **keepsave**

See the **save** command for more information on using the **copy** command.

debug

The **debug** command toggles a Z-Mail debugging trace off or on, or sets the debugging level.

SYNTAX **debug [*level*]**

RELATED VARIABLE *DBFILE* environment variable

RELATED OPTION **–debug**

Each **debug** command with no arguments toggles the Z-Mail debugging trace on (to level 1) or off (to level 0).

If the ***level*** specified is 0 (zero), then debugging is turned off. At ***level*** 1, Z-Mail generates a debugging trace that is useful for debugging Z-Script functions.

A ***level*** greater than 2 generates a great deal of additional output, most of which relates to the inner workings of Z-Mail and is generally incomprehensible except to Z-Code engineers. Therefore, using a ***level*** greater than 2 is not recommended.

If the *DBFILE* environment variable is set to a filename, Z-Mail will append the debugging trace to that file instead of displaying the trace on the screen.

EXAMPLE

debug 1 generates a trace similar to following when a function or script is executed. Each item in parentheses is one argument in the Z-Script execution engine:

```
executing: (my_hdr) (Reply-To:) (smythe@febrile.com)
executing: (my_hdr) (X-Security-level:) ([low|medium|high|top secret])
executing: (alias) (archie) (archie@archie.umn.edu)
executing: (ignore) (Encoding-Algorithm)
executing: (ignore) (Errors-To)
executing: (filter) (z-fax-filter) (save +zfax-autonotify) (-i) (-s) (Z-Fax
    Send Program)
executing: (function) (new-show()) ({)
function: loading "new-show"
executing: (button) (-n) (Update) (update)
```

delete

The **delete** command marks a message or group of messages for deletion.

SYNTAX

delete [*messagelist*]

SYNONYM

d

RELATED COMMAND

undelete

When you **update** or **close** a folder, messages marked as *deleted* are purged. Deleted messages are marked with an asterisk in the GUI and Lite message summaries, and the command-line header list. Deleted messages are unavailable for processing by some commands.

If a message is marked for deletion, but the folder containing the message has not yet been updated, you can recover the message with the **undelete** command.

detach

The **detach** command extracts an attachment from a message and stores it in a file you specify.

SYNTAX

detach [–part *num*] [–name *attachname*] [–encode *key*] [–display | –use *type*] [*messagelist*] [*filename*]

detach [–all | –list | –rehash]

RELATED COMMANDS

mail, attach

RELATED VARIABLE

attach_types

If you specify ***messagelist***, **detach** uses only the first message in that list. If you do not specify ***messagelist***, **detach** uses the current message. If you specify no arguments at all, **detach** lists the attachments of the current message.

You may specify attachments by name using the **–name** option, or by ordinal position in the attachment list using the **–part** option. If you specify both an attachment name and a part number, Z-Mail detaches the part numbered ***num*** and uses ***attachname*** as the output file name.

If you specify ***filename***, Z-Mail uses ***attachname*** only to identify the part to detach. If you specify ***num***, ***attachname*** and ***filename***, Z-Mail ignores ***attachname*** and detaches the part numbered ***num*** and writes it to a file called ***filename***.

Z-Mail automatically attempts to decode attachments which are encoded. You may also use the **–encode** option to tell Z-Mail to use a specific decoding ***key***. See *The Attachment Types File* beginning on page 328 and *Z-Mail User Guide* for more information about decoding keys.

You may optionally display or additionally process the output file by specifying the **–display** option. The ***filetype*** of the attachment determines how it is processed or displayed. You may force Z-Mail to display the attachment as a specific ***filetype*** by specifying the **–use** option. Use caution when you do this since it may have unexpected results.

To detach all attachments for a specific message, use the **–all** option. To list all the attachments for a particular message, use the **–list** option.

You or your system administrator should define your attachment types and encoding keys in the file *$ZMLIB/attach.types* in or the file(s) specified in the value of the **attach_types** variable. See *The Attachment Types File* beginning on page 328 for more information.

For details on sending attachments, see the **mail** command.

dialog

GUI and LITE ONLY

The **dialog** command causes Z-Mail to display one of its built-in dialog boxes in the GUI and Lite interfaces.

SYNTAX

dialog *dialogname*

RELATED COMMANDS

button, function

The **dialog** command displays the dialog box named ***dialogname***. This command is useful when you are creating user-defined buttons (see the **button** and **function** commands). Dialog box names are the same as the labels under the dialog icons in the
GUI's *Toolbox* dialog:

Aliases	Headers	Dates	Browser†
Fonts*	Menus	Compose*	Help*
Search	Sort	Colors*	Filters†
Envelope	Variables	Printer	Templates
Buttons†	Functions†	Toolbox*	

Dialogs marked with a * in the above list are not available in the Lite interface via the **dialog** command.

Dialogs marked with a † in the above list are not available in Z-Mail Basic.

disable

SYNTAX

disable *commandname1* [*commandname2* ...]

RELATED COMMAND

enable

The **disable** command restricts access to other Z-Script commands. Normally, this command is used only in the $ZMLIB/system.mailrc file by your system administrator.

A disabled command still operates normally when it is executed from system.zmailrc. Once system.zmailrc has been read, any further attempt to execute a disabled command is refused, even if the command appears in a Z-Script function that was defined before the command was disabled.

The disable and enable commands are themselves disabled once system.zmailrc has been read. To use the commands outside the system.zmailrc file, you must enable them explicitly in the system.zmailrc file, with the following commands:

enable disable
enable enable

display

The **display** command displays messages in the active folder.

SYNTAX **display [*messagelist*]**

SYNONYMS **print, p, type, t, read**

RELATED COMMANDS **ignore, retain, lpr, next, pinup, previous**

RELATED VARIABLE **alwaysignore**

The **display** command displays the current message if you do not supply a ***messagelist***.

By default, the **display** command does not display headers you have asked Z-Mail to ignore. If you specify the first letter of the command as a capital letter (e.g., **Display**, **Print** or **Type**) then Z-Mail displays **ignore**'d headers provided you have not set the **alwaysignore** variable to ignore all headers. See the **ignore** and **retain** commands for more information about ignoring message headers.

The **pinup** command (see page 187) works like **display**, but puts each message in its own window in the GUI and Lite interfaces.

You can also use the **+** and the **–** commands to display the next and previous messages, respectively.

each

The **each** command executes a command on a list of messages, one at a time.

SYNTAX

each *messagelist command*

RELATED COMMANDS

cmd, foreach, function

The **command** must be a single argument; it should be enclosed in quotes if it contains spaces. The **each** command gives each of the messages in ***messagelist*** to ***command*** as an argument one at a time, so you should use complex commands containing pipelines or semicolon separators carefully. It is recommended that you use a **cmd** alias or a user-defined function as ***command***.

EXAMPLE

For example, if you enter:

each 1–5 {3} 10 reply

Z-Mail executes the commands:

reply 1
reply 2
reply 4
reply 5
reply 10

If on any iteration of the loop ***command*** returns a nonzero exit status (e.g., sets the **status** variable to –1), the entire loop terminates.

Note: **cmd** aliases and functions may in some cases include commands that return a nonzero status without themselves returning nonzero. As with filters, certain commands cannot be executed in an **each** loop. See the **filter** command for a list of these commands.

echo

The **echo** command evaluates and displays its arguments.

SYNTAX

echo [–n] [–d] [–h | –p] *arguments*

RELATED COMMANDS

from, error, eval

RELATED VARIABLES

hdr_format, prompt

The **echo** command evaluates and echoes its arguments. Expressions that are evaluated by echo include command history references, variables, folder information variables (**–p**), and message information variables (**–h**).

If you specify the **–n** option, **echo** does not append a newline.

If you specify the **–d** option in the GUI or Lite interface, the output of **echo** is displayed in a dialog instead of the output area. You must dismiss the dialog before Z-Script execution continues. This option has no effect in the command-line interface.

If you specify the **–p** option, **echo** expands folder information variables that are embedded in the argument list. See *Folder Information Variables* on page 231 for a list of folder information variables.

If you specify the **–h** option, **echo** expands message information variables that are embedded in the argument list. See *Message Information Variables* on page 229 for a list of message information variables.

Note: You cannot specify both the **–h** and **–p** options in one **echo** command.

EXAMPLES

echo –h This is from %a and is dated %d.

might produce:

```
This is from genisoft!lfong and is dated Dec 14, 5:47pm.
```

and

echo –p There are %n new messages to read in %F.

might produce:

```
There are 5 new messages to read in /usr/spool/mail/mstrohm.
```

edit

The edit command allows you to edit messages in the active folder using a text editor you choose.

SYNTAX

edit [*messagelist*]

RELATED VARIABLE

editor

If you specify no arguments, Z-Mail launches the text editor indicated by the **editor** variable and loads the text of the current message, including headers. If you specify more than one message, Z-Mail allows you to edit one message at a time. As you finish editing each message and the editor exits, Z-Mail asks if you want to edit the next one. If you answer yes, Z-Mail loads the message. This process continues until Z-Mail exhausts the list of messages.

When you edit a message, be careful not to remove any message headers, such as *Date* or *From*. If you remove or change something you shouldn't have, you are notified of the error and Z-Mail does not save the changes you made to the message.

enable

Permit use of a previously diabled command.

SYNTAX

disable *commandname1* [*commandname2* ...]

RELATED COMMAND

disable

The **enable** command reverses the actions of the **disable** command, which restricts access to other Z-Script commands. Normally, this command is used only in the $ZMLIB/system.mailrc file by your system administrator.

The disable and enable commands are themselves disabled once system.zmailrc has been read. To use the commands outside the system.zmailrc file, you must enable them explicitly in the system.zmailrc file, with the following commands:

enable disable
enable enable

error

The **error** command causes Z-Mail to re-display the commands you type, expanding all arguments to show their values.

SYNTAX

error [–n] [–h | –p] ***arguments***

RELATED COMMANDS

from, echo

RELATED VARIABLES

hdr_format, prompt, status

The **error** command functions identically to the **echo** command except that in sets the value of the **status** variable to –1. In the GUI interface, the **error** command always displays its ***arguments*** in a dialog box.

eval

The **eval** command parses its arguments and executes the results as a command.

SYNTAX

eval [–h | –p] ***argument1 argument2*** **. . .**

You can use this command to force Z-Mail to perform multiple levels of variable expansion.

If you specify the **–p** option, **eval** expands folder information variables that are embedded in the argument list. See *Folder Information Variables* on page 231 for a list of folder information variables.

If you specify the **–h** option, **echo** expands message information variables that are embedded in the argument list. Message information variables are expanded using the current message. See *Message Information Variables* on page 229 for a list of message information variables.

Note: You cannot specify both the **–h** and **–p** options in one **eval** command.

Z-Mail expands command history references and Z-Script variables prior to expanding message or folder information variables, and implicitly quotes them so embedded quotes, spaces, tabs, **!** characters, etc. are handled correctly.

The following example finds all messages from the same author as the current message:

EXAMPLE

eval –h search –f %a

If you issue the **search** command above without preceding it with **eval –h**, Z-Mail incorrectly searches for all messages from the author named %a. See the **hdr_format** and **prompt** variables for more information.

exit

If you type **exit** from the command line, Z-Mail exits immediately without updating any folders.

SYNTAX

exit

SYNONYM

x, xit

When Z-Mail encounters an **exit** command in a user-defined function or script, it terminates the processing of the function or script and returns to the calling function (if any). You must use the **x** or **builtin exit** command to terminate the entire Z-Mail program from within a function. Z-Mail asks you for confirmation if you are currently composing any messages.

expand

The **expand** command expands an alias into its constituent addresses.

SYNTAX

expand *aliaslist*

This command takes ***aliaslist***, the alias name or names you specify as arguments, and displays the actual names of the recipients. The **expand** command is useful for identifying circular definitions before you attempt to use an alias name in a message.

If ***aliaslist*** does not exist, it is treated as an alias to itself.

EXAMPLE

If you have previously defined the two aliases x–project and manage:

alias x–project lmp12@merc.umg.com marlin stella@cat
alias manage eliason@fortune.umg.com x–project

then the command

expand manage

will display the expansion of the manage alias as follows:

manage: eliason@fortune.umg.com lmp12@merc.umg.com marlin
stella@cat

filter

The **filter** command allows you to install a command that Z-Mail invokes whenever new mail arrives in your system folder.

SYNTAX

filter [–n] [*filtername* [*command* [*search-options*]]]

RELATED COMMAND

unfilter

The ***filtername*** argument identifies a filter by name so you can modify, replace or delete it later. If you install more than one filter, they are executed in alphabetical order according to ***filtername***.

The ***command*** argument indicates the command you want Z-Mail to run when it executes ***filtername***. The ***command*** argument must be a single text string; it must contain quotation marks if it has any spaces.

You can use any valid option for the **search** command as a ***search-option*** in the **filter** command.

If you use the **–n** option when you define the filter, the filter only affects new mail that arrives while you are using Z-Mail interactively.

Note: It is never necessary for a filter to save or copy a message to the system folder. Z-Mail automatically appends new messages to your system folder unless they are marked for deletion by a filter or by you.

If you give the **filter** command with no arguments, it lists all the filters defined *without* the **–n** option. If you give the command **filter –n** with no additional arguments, it lists all the filters that have been defined with **filter –n**.

You may not use certain Z-Mail commands in filters because they can alter the state of the folder in a way that prevents other filters from functioning correctly. These restricted commands include:

await	update	filter
sort	quit	

To open a new folder, you can use the **folder** command within a filter. However, Z-Mail opens the new folder without updating the folder that is currently being filtered.

You may apply filters to any folder at the time it is opened by using the **–f** option of the **folder** command or the **–filter** option on the Z-Mail command line:

folder –f +mbox
% **zmail –filter –folder +mbox**

You may explicitly prevent filters from being executed at the time a folder is opened by using the **–F** option of the **folder** command or the **–nofilter** option of the zmail command:

folder –F foldername
% **zmail –nofilter**

If you specify the **–nofilter** option to **zmail**, all filters will still operate whenever a new message arrives in the system folder. You can only suppress this filtering by removing the filters.

You can remove previously defined filters using the **unfilter** command. See the **unfilter** command on page 216 for more information.

EXAMPLES

filter expire-old-mail delete –ago 3 months
filter save-announce "save +announce" –i –s announce
filter –n meeting "echo –d You have a meeting" –i –s meeting

flags

The **flags** command modifies the flags set for selected messages.

SYNTAX **flags [[+ | –] [D f N ! O P p R r S U]] [*messagelist*]**

RELATED COMMANDS **hide, unhide**

You may set or modify multiple flags at one time.

The modifiable flags are:

D	Deleted
H	Hidden
f	Forwarded
N	New
!	Newly arrived
O	Old
P	Preserved
p	Printed
R	Read
r	Replied-to
S	Saved
U	Unread
V	Unhidden

If you include a plus sign (**+**) or minus sign (**–**) in the list of flags, the specified flags are turned on or off respectively, without modifying other flags. If you do not specify a plus or minus sign, the list of flags is set explicitly and all other flags are turned off.

You can pipe message lists to or from the **flags** command.

folder

The **folder** command changes the active or current folder.

SYNTAX **folder [–a] [–d] [–f | –F] [–l] [–N] [–n] [–r]**
[%[*user*] | & | #[*n*] | *foldername*]

SYNONYM **fo**

RELATED COMMANDS **close, open, update**

If you do not include any arguments when you issue a **folder** command, Z-Mail displays the name of the active folder in the command-line interface. In the GUI and Lite interfaces, **folder** displays the *Open Folders* dialog.

Options to the **folder** command are:

–a	Open ('add') the named folder. Equivalent to the **open** command.
–d	Delete the named folder from the file system.
–f	Perform filtering when opening the folder.
–F	Do not perform filtering when opening the folder.
–l	(lowercase L) List the folders that are open.
–N	Don't display headers when entering new folder in command-line interface.
–n	Do not update the current folder.
–r	Set read-only mode so you can't alter the folder
–x	Create an index for the folder on update.
–X	Ignore index on load, do not create on update.

Special ways to specify the folder upon which to act include:

none	The active folder.
%	Means /usr/spool/mail/*$user*. (Your system folder.)
%***user***	Means /usr/spool/mail/*user*.
&	Means your main mailbox (Default is **~/mbox**).
#	Switch back to the most recently active folder.
#*n*	Switch to folder number *n*.

File names that do not begin with '**/** ' are interpreted relative to the current directory unless they begin with one of the metacharacters **+** or **~**. As in *csh*, the character **~** expands to your home directory (or to some other user's home directory if used as *~username*). The character **+** expands to the name of your folder directory (defaults to *~/Mail* or the value of the variable **folder**). For compatibility with other mailers, no **/** character needs to appear between the **+** and the name of the folder.

folder can only appear in a pipeline if it is the first command; it returns a list of all the messages in the new folder. You cannot use this command in initialization files before the user interface has started, since no folder has yet been opened.

Read-only folders

Updating a read-only folder will write an external index for that folder rather than rewriting the internal index. To force this behavior on a folder that is not read-only, use **update –external**. External indexes can thus *checkpoint* a writable folder as well as save the state of a read-only folder. Whenever the folder is loaded (except with **open –noindex**), the external index is used to restore the sorting order, saved/deleted/replied status, etc. of the messages.

Note that the index of a folder explicitly opened in read-only mode (with the **–readonly** option when starting Z-Mail or when using the **folder** command) is not normally updated. However, if **index_size** is set and the folder is large enough, an external index file is created when the read-only folder is updated or closed, even if the file already has an index. An index is created automatically if you update a folder when you do not have permission to write to the folder itself.

External index directory

The pathname of the directory where external folder index files are to be kept is specified in the variable **index_dir**. This path is used by **update –x** (and other commands) to create the external index file for a folder. If a complete pathname is not given, the name of the folder directory (abbreviated as **+**; see the **folder** variable) is prepended.

If **index_dir** is not set or is an empty string, then the directory +index.dir is used for external index files. The name of the index file within the index directory is the same as the name of the folder. If the index directory and the directory containing the folder are the same (e.g. **index_dir** is set to **+**) then the index file name is the folder name with *.ix* appended. On systems with file name length limits, the folder name is truncated before appending *.ix*.

Deleting external indices

External index files are removed by **rmfolder –x foldername**, or when the folder itself is removed by **rmfolder foldername**. When a writable folder is updated, the external index is removed when an internal index is written or when all messages have been deleted.

Backup Folders

In some circumstances, described in detail below, Z-Mail will automatically create *backup* folders. Backup folders, when created, are labeled as such in the folder status list in the main window as read-only folders named the same as the original with a string of digits appended.

This section explains when and why backup folders are created, in fairly technical detail. In most cases, users may safely let Z-Mail delete backup folders when Z-Mail offers to do so. In general, backup folders are only created when two different running Z-Mail (or other e-mail) programs are attempting to modify the same folder. This can happen in situations such as:

- several users share a read/write folder,
- a user leaves Z-Mail running at work and logs in from home, starting another Z-Mail which then modifies the folder(s) the Z-Mail at work has open,
- the maintainer of a folder makes changes *other than adding new messages to the folder* while other users have the folder open.

Temporary folders
When a folder is opened for reading and writing, Z-Mail makes a temporary copy of the "real" folder and subsequently works entirely on that copy until:

3. new mail is detected in the real folder, or
4. the user performs an *update* operation on the folder (in any of several ways, including **update**, **close**, **quit**, etc.)

New mail detection
Z-Mail detects new mail in the real folder by noticing that it has increased in size. When new mail is detected, Z-Mail attempts to load the new messages into the temporary folder by:

1. locking the real folder,
2. opening the real folder,
3. moving to the old end-of-folder position in the folder,
4. attempting to start reading messages starting at the old end-of-folder.

Backup folder creation
If Z-Mail cannot start reading messages starting at the old end-of-folder, it assumes that the real folder has been changed by something more drastic than the arrival of new mail. The temporary folder is converted into a backup folder and the real folder is re-scanned from the beginning.

Backup folders are also created if the real folder *shrinks* relative to the temporary folder; Z-Mail will assume that messages have been deleted from the real folder.

Cases when backup folders are NOT created
If the real folder is modified in such as way that it remains exactly the same size, for example, it is re-sorted and then updated, Z-Mail will *not* notice that it differs from the temporary folder.

Backup folders are never created from read-only folders, because it is not necessary for Z-Mail to create a temporary copy of a read-only folder.

Backup folders are also not created if no changes have been made to the temporary folder since it was created, i.e., there has been no deletion, sorting, reading of previously unread messages, etc. In such cases the temporary folder is simply discarded and the real folder is re-read.

What to do with backup folders
In most cases the only thing you need to do with a backup folder is let Z-Mail delete it. When you close or update a folder for which a backup has been created, Z-Mail will offer to save the backups as well; you may wish to do this if you are concerned that messages you still need may not have been in the real folder any longer when it was re-scanned. Alternatively, you can cancel the entire update operation, change to and examine the backup folder(s) and save any messages from there that you feel are important.

Backup folders are saved automatically by Z-Mail to your **folder** directory when Z-Mail receives a *HUP* signal and the **hangup** variable is set. See the *kill(1)* man page for more infomation about signals.

Indexing and Shared (Group) Folders

A shared, or group, folder is simply a folder file to which many users have read access, but to which only one (or a few) have write access. The shared folder acts as a *bulletin board* where users can read messages of common interest. This has some advantages over sending a copy of each message to all the interested users. For example, when an outdated message is deleted by the maintainer of the folder, it disappears from every user's image of the folder as well.

Further, each new message is delivered only to the shared folder, rather than one copy to each user; this saves disk space.

Z-Mail's external folder indices permit each user of a shared folder to maintain their own image of the state of that folder. The shared folder should be writable only by a designated maintainer, so when other users open it, Z-Mail automatically uses read-only mode. If a user replies to, deletes, or otherwise modifies messages, Z-Mail tracks his changes. Then, when the user updates or closes the folder, Z-Mail updates that user's external index file.

The next time the same user opens the shared folder, Z-Mail uses the index file to restore the folder state as nearly as possible. Messages that have been deleted by the folder maintainer are lost. However, almost all other state is restored, including sorting order if the user changed it. New messages that have been placed in the folder since the user last accessed the folder are appended after those that are referenced in the user's index.

Creating a Shared Folder

To set up a shared folder, determine first how new messages should be added to that folder. If messages are to be sent automatically to the group, you may wish to create a *pseudo-user.* A pseudo-user is a user name that exists on the system but does not correspond to any real person. Pseudo-users are typically configured so that they cannot be used as a login userid, so that only the super-user or another specially designated user can act in the name of the pseudo-user. In this case, the folder maintainer, if there is one, is that designated user.

The system mailbox of the pseudo-user becomes the shared folder. A pseudo-user can receive electronic mail even if "he" can never log in to read it. The mailbox should be created and made readable by the group that is to share the messages (or by all if everyone is to have access). It should be both readable and writable by the pseudo-user account only. Messages to the group are sent to the pseudo-user, so they show up in the shared folder (the pseudo-user's mailbox).

For example, suppose that we wish to create a shared folder to be read by Z-Mail users, for discussion of the Z-Mail program. We first create a pseudo-user named *mailGhost*. (The details of user creation vary among different UNIX systems.) Next, we create a mailbox named *mailGhost* in the system mail area (*/usr/mail* or */usr/spool/mail*, usually). Any user should be allowed to read this folder, so we set the permissions for *read and write* by owner only and *read* by all.

Now, users can submit messages to the shared folder by sending mail to mailGhost. Once messages have begun to arrive, Z-Mail users can open the folder %mailGhost to read the messages. The abbreviation % means the system mail area, so %mailGhost refers to the system mailbox of user mailGhost.

Moderated folders

Sometimes, it is desirable to have new messages screened before they are made available for group access. Z-Mail's filtering mechanism can be used to automate this process on behalf of a pseudo-user. However, there may be cases where automated filtering is not sufficient. In these cases, it is necessary to appoint one user (or possibly a few) as maintainer of the shared folder. Messages to the group are sent to the maintainer, who adds them to the folder as appropriate. A folder maintainer who has this responsibility is often called a *moderator* in e-mail lingo.

When a shared folder has a moderator, it is not necessary to establish a pseudo-user and system mailbox. Instead, any folder that is writable by the maintainer only, and that is readable (but not writable) by the interested group, can be used as a shared folder.

Maintaining a Shared Folder

In addition to screening the messages, a maintainer is responsible for deleting outdated (or *expired*) messages from the shared folder. As with delivery, it is possible to automate the expiration of messages from the shared folder with Z-Script functions and/or filters. In this case, a system process such as *cron* periodically executes Z-Mail to run the Z-Script commands that handle message expiration.

If the folder is a pseudo-user's system mailbox, cron can be told to act as the pseudo-user when running the Z-Mail script. Otherwise, cron should be told to run as the maintainer of the shared folder.

The following command deletes messages that are 31 days old or older from the shared folder of the previous example:

su mailGhost –c 'zmail –exec " search –ago –31days | delete; quit " '

Here, the owner of the shared folder is the pseudo-user mailGhost, so we are issuing an *su* command to cause the command following the **–c** to be run by the user mailGhost. The command which mailGhost will execute is

zmail –exec "search –ago –31days | delete; quit"

Examining this command, we see that Z-Mail is started and passed, via the **–exec** option, the Z-Script command

search –ago –31days | delete; quit

which selects the messages in mailGhost's system folder (the shared folder) that are 31 days old or more, pipes the selected message list to the delete command which marks them as deleted, then quits Z-Mail. The messages which have been marked as deleted are purged automatically by the quit command. If a *cron* entry is set up to run this command every day, the shared folder will be automatically cleaned of outdated messages with no further intervention.

folders

The **folders** command lists the names of the folders in your folder directory in the command-line interface. In the GUI and Lite interfaces, **folders** brings up the *Folder Manager* dialog.

SYNTAX **folders**

Your folder directory is usually the directory *~/Mail*. You can set the **folder** variable (see page 277) to specify another folder directory.

foreach

The **foreach** command iterates over message numbers or other strings you specify in a list, executing a ***command*** on each iteration.

SYNTAX

foreach ***loop_var messagelist command***
foreach ***loop_var*** **(*anylist*)** ***command***

RELATED COMMANDS

cmd, each

The ***command*** must be a single argument; it should be enclosed in quotes if it contains spaces. If ***anylist*** is enclosed in parentheses, filename metacharacters that are present in the string are expanded. See *Specifying Filename Patterns* on page 27 for a list of valid filename metacharacters.

If parentheses are not present, then the list is treated as a ***messagelist***, and message list metacharacters in the string are expanded. See *Specifying Message Lists* on page 24 for a list of valid message list metacharacters.

The argument ***loop_var*** is a variable name in the processing of the **foreach** command. On each iteration of the **foreach** loop, the next value of ***messagelist*** or ***anylist*** is assigned to ***loop_var***.

If **command** returns a nonzero exit status (e.g., sets the **status** variable to –1) on any iteration of the loop, the loop terminates. **cmd** aliases and functions may in some cases include commands that return a nonzero status without themselves returning nonzero. After termination of the foreach loop, execution continues with the next Z-Script statement, if there is one.

Note: As with filters, certain commands cannot be executed in a **foreach** loop. See the **filter** command for a list.

EXAMPLE 1

For example, to reply to messages 1, 2, 3, and 10 in order and automatically include their text, enter the command:

foreach num 1–3 10 'reply $num –i'

Z-Mail then executes the commands:

reply 1 –i
reply 2 –i
reply 3 –i

reply 10 –i

EXAMPLE 2

See the **openfolders** variable on page 298 for an example of using **foreach** with an ***anylist*** argument.

EXAMPLE 3

When sending a large file via e-mail, it is common practice to break the file into several smaller files with a utility such as *split*. When you *split* a file, by default it creates a number of smaller files that are named *xaa*, *xab*, *xac*, *xad*, and so forth. You can use the following **foreach** command to mail all the files:

foreach file (~/xa?) 'mail –s "Part: $file:t" –H $file –U marvin@zoo'

Assuming that there are three files named *~/xaa*, *~/xab*, and *~/xac*, this **foreach** command will send three messages to *marvin@zoo* with the **mail** command. Here's the ***command*** that will be executed on the first iteration:

mail –s "Part: xaa" –H /usr/myuser/xaa –U marvin@zoo

See the **mail** command on page 171 for more information about the arguments to the **mail** command used in this example.

from

The **from** command displays information taken from the headers of a message list.

SYNTAX

from [+ | –] [*messagelist*] [*pattern*]

SYNONYM

f

If you do not specify any arguments, the **from** command displays the current message's header summary.

If you specify only ***messagelist*** or if you pipe a list of messages to **from**, the header summaries of the indicated messages are displayed (and selected in GUI and Lite interfaces).

If you specify a ***pattern***, the **from** command displays the headers of those messages whose *From:* headers match ***pattern***. If you specify both a ***pattern*** and a ***messagelist*** (or if you supply a ***messagelist*** via a pipeline), Z-Mail searches for ***pattern*** in only the messages indicated in the ***messagelist***.

You can specify the special arguments – and **+** to move the current message pointer to the previous or next message, respectively, while also displaying that message's header. If you specify a message list in addition to – or **+**, then the current message pointer is set to the first or last message in the message list you specify. If the **wrap** variable is set (see page 316), the next or previous message may be found by "wrapping around" from the end of the folder to the beginning, or vice-versa.

EXAMPLES

from + Kim

displays the headers of all messages that contain "Kim" in the *From* header and sets the message pointer to the last message that meets the criteria specified.

from – 10–30 {16}

displays the header summaries of messages 10 through 30 except for message 16 and sets the current message pointer to 10.

from +

displays the header summary of the message after the current message and increment the current message pointer to the that message.

from $

displays the last message's header summary but does not move the current message pointer.

function

The **function** command creates user-defined functions.

SYNTAX

function [*name*]
functions

function *name* () {
commands ...
}

RELATED COMMAND

unfunction

If you specify the function command without any arguments, it lists the definitions of all known functions. If you specify ***name***, the **function** command lists the definition that function.

The command **functions** displays all the function definitions, without their scripts.

To define a new function, specify a ***name*** followed by a set of empty parentheses with no spaces between them, and the last argument must be an open brace (**{**). Z-Mail reads succeeding lines until it encounters a close brace (**}**) at the beginning of a line. Z-Mail stores the text enclosed in the braces as the function body, and executes the text when the function name is given as a command.

You may use indentation and comments (beginning with **#**) just as you do in initialization files and scripts. When you load function text, Z-Mail converts leading tab characters to 8 spaces, but does not change tabs specified elsewhere. This allows Z-Mail to preserve sensible indentation when function definitions are written to a file by a **saveopts** command.

When you use a function and you specify arguments on the command line, the arguments are available in the function text via a set of special variables. These variables are numbered 1 through the total number of arguments passed, and can be referenced as **$1**, **$2**, ... The complete argument list can also be referenced as **$***. See page 318 for more information about these variables.

EXAMPLE

```
function saveit() {
    if ! $?_filename
        if ! $?mbox
        set mbox = ~/mbox
        endif
    set _filename = $mbox
    endif
    ask –file _filename –d $_filename "Save: $*\nFilename:"
    if $status == 0
        save $* $_filename
    endif
}
```

The **saveit** function defined above is an example of an interactive user-defined function. This function prompts the user for a file name to which messages are saved and remembers the filename you specify so that all the filenames you select during your current session appear in the selection list (created by the **ask** command) the next time you call the function.

The **saveit** function first checks whether a file name has been defined. If not, it makes sure that the **mbox** variable has been initialized, then sets **_filename** to be the same as **mbox**. Next, it uses the **ask** command to prompt for a filename, providing the name stored in **_filename** as the default response. The function stores the actual response in the **_filename** variable. The question passed to ask shows the list of messages (**$***), followed by a newline (represented by **\n**), and finally prompts with the string Filename:. Finally, if **ask** returns successfully, the **save** command saves the argument messages to the file.

Hook Functions

Hook functions are used to have user-defined functions execute at specific times. Z-Mail provides four hook functions:

startup_hook	Called at startup after loading first folder
exit_hook	Called at quit after closing first folder
fetch_mail_hook	Called at the time interval specified by the fetch_time_out variable
receive_mail_hook	Called after fetch_mail_hook completes.

To use a hook function, simply create a function with that name.

headers

The **headers** command displays a screenful of message headers listed in the active folder. **headers** is primarily used in the command-line interface, since the GUI and Lite interfaces maintain a list of the active folder's header summaries in the Main Window.

SYNTAX **headers [[–H] [:*c*]] [+ | –]**

SYNONYM **h, z, :*c***

If you specify a message number as an argument, Z-Mail displays headers for that message first.

You may set the variable **screen** to tell Z-Mail's command-line interface how many headers to display at a time. In the GUI interface, the variable **screen_win** controls the number of headers used in the headers subwindow. In the Lite interface, the number of headers that are displayed is determined by the height of the terminal screen.

A typical command-line interface header may look like:

> 5 +Nr dsmith@justin.case.com Feb 19, (10/278) Sales forecast

The **>** indicates that the current message pointer is on this message and the **5** indicates that it is message number 5. The next three characters are the message flags. The first of these flags is only used if the message has been marked or has been commandname2tagged with a priority. In the sample line above, the **+** character shows that this message has been marked. See the **mark** command for details on marking and message priorities.

The second position can contain any of the following status flags:

!	Newly arrived and unread
N	New and unread
U	Old, but still unread
*	Deleted
S	Saved
P	Preserved
(blank)	Already read

The third position may have an r if the message has been replied to, an f if the message has been forwarded, or a p if it has been printed on a hardcopy printer. Since there are only three positions for flags and more than three flags, some flags override the display of other flags.

The author of the example message header is *dsmith@justin.case.com*, the date is *Feb 19*, the number of lines in the message is *10*, the number of characters is *278* and the subject of the message is *Sales Forecast*. The format of the message header is described by the string variable **hdr_format**. The example given above has a **hdr_format** of:

```
set hdr_format = "%25f %7d (%l/%c) %25s"
```

For more information on specifying **hdr_format** strings, see *Message Information Variables* on page 229.

Header Modifiers

You can display a special subset of message headers by using the **–H:*c*** option, where ***c*** is one of:

a	All messages
d	Deleted messages
f	Forwarded messages
h	Hidden messages
m	Marked messages
n	New messages
o	Old messages
p	Preserved messages
r	Replied-to messages
s	Saved messages
u	Unread messages
v	Visible (unhidden) messages

Note that the **headers** and **–H** are not required; **headers :*c*** or **:*c*** is also valid.

Other options to the **headers** command include **+** and –. These characters display the next or previous screenful of message header summaries. The **z** and **z –** synonyms are identical to the
h + and **h –** commands and provide compatibility with *BSD Mail* and *Mush*.

The **headers** command pipes the message numbers of all the messages it displays to the next command in a pipeline. If you pipe into the **headers** command, it displays the message headers of the messages corresponding to the message numbers that are piped in. Piping into the **from** command has the same result.

help

The help command provides usage information and tips on a variety of Z-Mail topics.

SYNTAX

help [*topic*]

The **help** command with no ***topic*** provides general information as to how to get further help and a list of topics suggested for more specific help. To get help on how to use the help facility, type:

help –?

You can add the **–?** argument to any command to get help on that command. For example, **save –?** displays information about the **save** command.

If Z-Mail can't find the text for the help file, it displays an error message. You can tell Z-Mail the location of your Z-Mail help files by setting the variables **cmd_help** (see page 264) and **gui_help** (see page 279) to the paths of the files.

hide

Turn the hidden flag on for a list of messages.

SYNTAX

hide [*message-list*]

RELATED COMMANDS

flags, unhide

The **hide** command changes the status of messages to hidden. Hidden messages are not displayed in message summaries.

If no list of messages is given, only the current message is hidden.

history

COMMAND-LINE ONLY

The **history** command displays the command history in chronological order.

SYNTAX

history [–h] [–r] [*numlines*]

RELATED VARIABLE

history

Z-Mail displays earlier commands first followed by more recent commands. You can suppress the displaying of history event numbers using the **–h** option. If you specify the **–r** option, Z-Mail displays history events in reverse order.

If you specify ***numlines***, then Z-Mail displays that number of previous commands rather than the number set by the **history** variable. For more information, see the **history** variable on page 281 and *Command History* on page 85.

iconify

The **iconify** command transforms your Z-Mail GUI session into a Z-Mail icon on your desktop.

SYNTAX

iconify

When you issue the iconify command, the status of your Z-Mail session is preserved. All Z-Mail windows and dialog boxes disappear from the screen.

In the command-line and Lite interfaces, the **iconify** command is equivalent to the **stop** command.

if...else...endif

You can use the **if** command to test for specific conditions before executing a set of commands. You cannot use the **if** command from the Z-Mail command line; you can only use **if** in functions and scripts.

An **if** command tests an ***expression*** of the form:

argument1 operator argument2

You must separate all the arguments of an **if** command, including the conditional operator, from each other with blank spaces.

The indentation used in presenting the following syntax and examples is for clarity only; it does not affect the processing or meaning of the commands.

SYNTAX

All conditional statements begin with an **if** expression and end with **endif**. The general form of an **if** command is:

if ***expression***
actions
else
alternate actions
endif

The **else** clause and its associated *alternate actions* are optional. Therefore, the simplest form of an if expression is:

if ***expression***
actions
endif

You can specify more than one action per **if** expression, so you can create conditional statements such as:

```
if  expression
        action1
        action2
        action3
endif
```

You could also perform multiple actions using this syntax:

```
if  expression
        action1;  action 2;  action 3
endif
```

To create complex conditional processing, you can nest **if** statements. The syntax of a more complex conditional statement is:

```
if  expression1
        if  expression2
            if  expression3
                action
            endif
        endif
endif
```

In the above example, Z-Mail performs ***action*** only if all three expressions are true.

Specifying Conditional Expressions

When you specify conditional expressions, follow these rules:

- You must specify exactly one conditional ***expression*** for each **if** line.

- You must separate arguments in expressions with blank spaces, including comparison operators (e.g. **==**, **>=**, etc.)
- You must have an **endif** statement for every **if** statement.
- You must place the **if** statement on a line by itself.
- You may not use parentheses in conditional expressions.
- You may compare strings to strings, numbers to numbers, but not strings to numbers.
- You must prefix a variable name with a **$** to reference the variable's value (except for some internal variables; see *Special Cases*, below). Otherwise, Z-Mail treats the variable name as a character string.

Special Cases

For backward compatibility, the following internal variables should *not* be preceded by a **$** in conditional expressions:

is_gui	is_lite	is_sending	is_shell
hdrs_only	redirect		

Else Clauses

You use an **else** clause to tell Z-Mail to perform an alternative action when the conditional expression in your **if** command evaluates to false. You must specify the **else** keyword on a line by itself with the alternative action commands following it.

EXAMPLE

The following statement tells Z-Mail to sign your outgoing message with the contents of the *.signature* file in the GUI interface and to use the *.alternate* file in other interfaces.

```
if  is_gui
      set  autosign = ~/.signature
else
      set  autosign = ~/.alternate
endif
```

Comparison Operators

When you construct conditional expressions, you may use any of the following operators. Make sure you separate the operators from other arguments by blank spaces.

Operator	*Effect*	*Example*
==	Equivalent	**if "$printer" == lpr**
!=	Not equivalent	**if "$screen" != "$crt"**
<	Less than	**if "$crt" < 20**

<=	Less than or equal	**if "$history" <= 5**
>	Greater than	**if "$screen" > 10**
>=	Greater than or equal	**if "$screen" >= 20**
=~	Equivalent pattern	**if "$TERM" =~ "hp*"**
!~	Not equivalent pattern	**if "$rep" !~ "report[2,3]"**

Z-Script does not currently support **&&** (logical *and*) or **||** (logical *or*) operators to evaluate more than one expression.

Conditional Arguments

The operators in a conditional expression compare two ***arguments*** that can be:

- Character strings (e.g., report or "sales department")
- Numbers
- Variable names preceded by a **$** character (e.g., "**$screen**").

Comparison arguments may not contain:

- Numeric expressions (e.g., 4+5), or
- Parentheses, except within quoted strings.

Variables

When you use a variable name as an argument in a conditional expression, you must precede it with a **$** character or Z-Mail interprets the variable name as a text string. If the value of a variable could contain a blank, enclose the variable name in quotes so Z-Mail treats the value as one argument.

EXAMPLE

In the following example, even if the **report** variable evaluates to a multi-word phrase such as Income Statement or Balance Sheet, the **if** statement still evaluates correctly because the variable, and therefore the string to which it expands, is enclosed in quotes:

```
if "$report" == "Balance Sheet"
     return
endif
```

Notes: When Z-Mail expands an undefined variable, it evaluates as an empty (null) string. Variables that evaluate to an empty string can cause errors. To test whether a variable is undefined, do *not* compare it to an empty string. Instead, use the syntax **$?*varname***. This expression evaluates to true if ***varname*** is defined and false if it is not defined.

EXAMPLE

The following **if** command tests if the **printer** variable is defined. If **printer** is defined, the command prints the currently selected messages.

```
if $?printer
    lpr $*
endif
```

Patterns

You can also use special pattern characters in character strings as the arguments of your conditional expressions. When you specify these arguments, you must use the **=~** operator that compares the two arguments for an equivalent pattern.

Character	***Matches***	***Example***
?	Any single character	**"report3"** matches **"report?"**
*	Any string	**"Prod"** matches **"Pr*"**
[*a,b,c*...]	Any character in list	**"Prod"** matches **"[P,p]rod"**
		"rep2a" matches **"rep[2,3]a"**

EXAMPLES

If you set the values of the three variables listed below as indicated, the following table shows how Z-Mail evaluates conditional expressions.

```
set report = 3
set rname = "Product2"
set TERM = "ncd16e"
```

Command	***Evaluates to***
if "$report" == 3	True
if "report" == 3	False (no **$** prefix)
if "$rname" == "Product2"	True
if "$TERM" != 'ncd16e'	False (arguments are **==**)
if "$TERM" =~ "ncd*"	True
if "$TERM" =~ "ncd16?"	True
if "$rname" == "product"	False (lower-case p)
if "$rname" =~ "[P,p]roduct?"	True
if "$rname" == "[P,p]roduct?"	False (needs **=~**)
if "$rname" =~ "[P,p]roduct[2,3]"	True
if "$TERM" =~ "ncd1[3,5,6]?"	True

File and Folder Operators

When you reference folders or files, you can test to ensure they exist and contain information before you use them. You use the **–e** option to test for the existence of a folder or file, and the **–z** option to test if an existing folder is of zero length (is empty).

Operator	*Effect*	*Example*
–e	File exists	**if –e $logfile**
–z	File is of zero length	**if –z salesforecast**

EXAMPLES

The following **if** command tests for the existence of a folder specified by the **salesforecast** variable and opens that folder if it does exist.

```
if –e $salesforecast
    folder $salesforecast
endif
```

The next example tests to see if the folder called tempfile is empty. If it is, Z-Mail saves the currently selected messages to tempfile.

```
if –z tempfile
    save $* tempfile
endif
```

interpose

NOT AVAILABLE IN Z-MAIL BASIC

Allow a user-defined function to augment a Z-Mail command or operation.

SYNTAX

interpose -before ***command interposing_function***

interpose -after ***command interposing_function***

interpose -operation ***keyword interposing_function***

uninterpose -before ***command interposing_function***

uninterpose -after ***command interposing_function***

uninterpose -operation ***keyword interposing_function***

RELATED COMMAND

thwart

The **interpose** command allows a user-defined function to be piggybacked (or *interposed*) onto a command or operation. Whenever Z-Mail executes the command or operation, the interposed function is also called. Functions can be interposed either before or after commands, but only before operations.

Operation interposers are called whenever the operation occurs, no matter what the reason. Command interposers are called only when the command or one of its synonyms executes.

operation may be one of the following:

display	a message is displayed to the user
delete	a message is marked for deletion
undelete	a deletion mark is removed from a message
save	a message is saved to a file
mail	a composition is initiated
attach	a file is attached to a composition
send	a completed composition is sent

command may be any Z-Mail command.

When interposed before a command, the user-defined function receives exactly the same arguments as the command itself. The interposed function can determine which command caused it to be called by examining the value of positional parameter $0.

Notes: While a function may interpose on any Z-Mail command, you should be aware of several things:

It is more useful to interpose on the send operation, rather than the **mail** command. The **mail** command may return while the composition is still in progress.

The **attach** command is used to define attachment types, not to attach files to messages. The attach operation is more suitable for interposing upon.

The **flags** command can be used to change message status directly, bypassing the interposers for operations such as delete.

Changing folders within an interposer is not prohibited, but can have unpredictable side-effects for message operations. These functions should call **thwart.**

Certain **compcmd** operations automatically call **thwart** when called from an interposer on the send operation.

These include the edit-message and edit-visual operations.

EXAMPLE

The following checks and X-Acknowledge header on any message that is displayed and asks the user to confirm reading the message.

```
function confirm_read() {
 # Get the value of the X-Acknowledge header
 set acks = "$[%?X-Acknowledge?]"
 # Check whether it includes "read", using lower case
 # The X makes sure "if" never receives an empty arg
 if "X$acks:l" =~ *read*
        # Ask whether to acknowledge, not to, or cancel
        ask "Acknowledge that you've read this message?"
        if $status == 0
             # If acknowledged, send the reply
             reply -send -file ~/.acknowledged
        else
             if $status < 0
                    # If canceled, don't continue display
                    thwart
             endif
        endif
 endif
 }
 interpose -operation display confirm_read
```

ignore

The **ignore** command suppresses specific headers in displayed messages.

SYNTAX

ignore [*headerlist*]

RELATED COMMANDS

unignore, retain

RELATED VARIABLE

alwaysignore

When you read messages, all message headers are displayed with the text body of the message. In many cases, these headers are cumbersome and uninteresting. You can filter out unwanted headers by using the **ignore** command. For example, a ***headerlist*** of

ignore Received Message–Id X–Mailer

causes the three specified headers to be suppressed.

Subsequent **ignore** commands are cumulative. Use the **unignore** command (page 217) to reverse the effects of **ignore**.

You may also use the *Headers* dialog of the GUI and Lite interfaces to select which headers to show and ignore.

When a message's headers are matched against the ignored header list, Z-Mail first tries to match case-insensitively. If none of the ignored headers is found, Z-Mail then attempts to match the message's headers case sensitively, this time considering any metacharacters that are embedded in the list of ignored headers.

You can use the file pattern-matching metacharacters in ***headerlist***; see *Specifying Filename Patterns* on page 27 for a list of valid metacharacters. For example,

ignore *	ignores all headers
ignore Received*	ignores headers that start with **Received**
ignore *–Sun*	ignores headers that contain **–Sun**
ignore X–{Sun,Zm}*	ignores headers that start with **X–Sun** or **X–Zm**.

To explicitly specify headers you wish to see, rather than ignore, use the **retain** command.

jobs

The **jobs** command display a list of suspended compositions, which are composed messages currently in progress.

SYNTAX

jobs

Z-Mail identifies each suspended composition with a number. The most recently suspended composition is also marked with a **+** sign. By default, this is the composition that is continued by the **resume** command. The **+** sign is followed by the first part of the *To:* line, and then the first part of the

Subject: line. If the *To:* line is truncated, it ends in an ellipsis (**...**).

EXAMPLE

A typical job listing might look like:

```
[1]     To: kfong,  rteller  Benchmark results
[2]     To: jgomez           Thanks for the extra effort
[3]  +  To: managers         June reports are due Friday
```

Note: Suspending compositions is not true job control in the sense used by some shells, because there is no UNIX "process" associated with a suspended composition. If you are editing headers, the job listing may not accurately reflect the state of the *To:* and *Subject:* lines in the editor file.

In GUI and Lite interfaces, all compositions in progress are considered to be suspended at all times.

lpr

The **lpr** command takes messages from a message list and sends them to the printer, one by one, each separated by page feeds.

SYNTAX

lpr [–h] [–n] [–P *name*] [*messagelist*]

RELATED COMMANDS

ignore, retain

RELATED VARIABLES

alwaysignore, print_cmd, printer, printer_opt

Use the **–h** option to suppress the printing of ignored headers (see the **ignore** and **retain** commands and the variable **alwaysignore**). Specify the **–n** option to suppress all headers. A default printer name is supplied if you do not specify one using the
–P*name* option. If you set the **printer** variable, the **lpr** command uses the value of **printer** as the printer name.

If you set the variable **print_cmd** to a value, Z-Mail uses the command described by that value instead of the default system command for printing. In such cases, Z-Mail ignores the **–P** option and the **printer** variable and the command is simply executed as is.

Z-Mail flags each message it prints with a p status.

Note: If the **lpr** command produces an error message, your system administrator may have configured Z-Mail to require **–d** in place of the **–P** option.

ls

The **ls** command emulates the UNIX command */bin/ls* that lists files.

SYNTAX

ls [*flags*]

By default, the **ls** command always uses the **–C** flag which displays filenames in column format. See the *ls(1)* manual page for a description of valid ***flags***.

mail

The **mail** command sends a message to a list of addresses. If you do not specify a list of recipients on the command line, Z-Mail requests a list from you by displaying a To: prompt in the command-line interface. In the GUI and Lite interfaces, the **mail** command starts a new composition in its own window.

SYNTAX

mail [*options*] [*recipients* ...]

SYNONYM

m

RELATED COMMAND

reply

You must supply a list of recipients before you send the message. Z-Mail allows you to specify files, mail folders and programs as recipients. If you send a message to a file, the filename must be a full pathname; filenames must start with a slash (**/**) so Z-Mail can differentiate filenames from user names.

The file metacharacters **+** and **~** may precede pathnames; see *Specifying Filename Patterns* on page 27. To send a message to a program, precede the program name with the pipe character (**|**). Full pathnames are not required for programs that are in your *PATH* environment variable.

You can specify the following options in the **mail** command:

–A [*type*:]*file*	Attach *file* of indicated *type*
–b *addrlist*	Set list of blind carbon copy recipients.
–c *addrlist*	Set list of carbon copy recipients.
–**C *comment***	Place ***comment*** in X-Remarks header.
–E[!]	[do not] Edit outgoing message headers.
–e[!]	[do not] Immediately enter the editor named by the **visual** variable.
–F[!]	[do not] Add a random fortune to the end of message.
–f [*msglist*]	Forward messages (not indented).
–H *file*	Read file as prepared text (no headers).
–h *file*	Read file as a draft (text and headers).
–I [*msglist*]	Include messages with headers (indented).
–i [*msglist*]	Include messages in letter (indented).
–p *template*	Read file as a prepared message template.
-q [*msgnum*]	Send message immediately to transport agent. Accepts one message number only. If a message number isn't specified, the current message number is used.
-q! [*msgnum*]	Read message into a compose window. Accepts one message number only. If a message number isn't specified, the current message number is used.
-Q *file*	Read file in as message body and send immediately to transport agent.
-R	Request a return-receipt
–s [*subject*]	Prompt for subject.
–U	Send draft unedited (use only with **–h** or **–H**).
–u	Unsigned; no signatures or fortunes added.
–v	Verbose (passed to MTA).

Note: You may not be able to specify the verbose option (**–v**) depending on the MTA on your system. If your MTA cannot accept the **–v** option, you will receive an error when you attempt to send the message.

The **–E** option causes Z-Mail to place the headers of the outgoing message in the editor file so you can change them. See the description of the **edit_hdrs** variable on page 273 for details.

Indenting Included Text

The **–i** option allows you to include the current message in the body of the message you are about to send. By default, the included message is indented with "**>** ". You may change the indent string by setting the **indent_str** variable. If you specify a message list after the **–i** option, then the messages described by that list are included. The **–I** (capital i) option is identical to the **–i** option except that the headers of the message are also included.

Setting Subject

The **–s** option looks at the next argument and sets the message subject to that string. If the string contains spaces, enclose the entire subject in quotes. If there is no following argument, then Z-Mail prompts you for the subject. This option is useful if:

- You do not have the **ask** variable set, or
- You wish to change the subject used with reply.

Forwarding Messages

Use the **–f** option to forward a message. By default, the **–f** option allows you to edit the forwarded message. Z-Mail creates a new set of headers and places the forwarded message into the body of the composition indicating it as forwarded:

```
--- Forwarded message from dsmith@justin.case.com

        forwarded message here

--- End of forwarded message
```

If you specify the **–U** (unedited) option in conjunction with the
–f option, Z-Mail uses the headers of the forwarded message and adds special headers:

Resent-From:
Resent-To:

and perhaps others, depending on your MTA. The *sendmail* MTA, for example, may add a number of other Resent headers.

You can specify an optional message list just as you can with the **–i** option. The subject of the message (if available) is the same as the current message; it is not necessarily the subject of the message being forwarded. The subject of forwarded mail cannot be changed. Z-Mail does not prompt you for the subject and it is ignored completely if you specify the **–f** option. However, using the **–e** option, you can change the subject once you are in editing mode either by using the escape sequence **~s** or by editing the Subject: header.

Overriding Signatures

The **–u** option, meaning *unsigned*, prevents you from appending signatures and fortunes to messages. It overrides the **autosign** and **fortune** variables, and overrides the **–F** option if you specify both **–u** and **–F** on the command line.

Message Templates and Draft Files

The **–h** option indicates that the file you specify is a *message template* — a previously prepared message; possibly a partial message saved with **~w**. This file is called a *draft*. You must specify the ***file*** argument and all message headers must either be present or you must add them manually. At a minimum, there must be a To: header; Z-Mail automatically adds From: and Date: headers if needed before sending a message.

If you specify the **–p** option and indicate a filename, Z-Mail starts composing a new message and loads the text from the file so you can edit it.

Importing Text Files

To read a prepared text file that does not contain headers, use the **–H** option. If you also specify the **–U** option, Z-Mail sends the message immediately without further editing. See also *Importing Text Files into Messages* on page 323 for more information.

Attachments

If you specify the **–A** option, Z-Mail appends the file to the message as an attachment. Attachments may be either text or binary, but binary files must be encoded to 7-bit ASCII before they are sent. See the *Z-Mail User Guide* for more information about attaching files in the GUI and Lite interfaces. See the **~A** command on page 97 and **–attach** option on page 40 for more information about attaching files in the command-line interface.

map

map!

COMMAND-LINE ONLY

The **map** command creates or lists macros for the command-line interface, and the **map!** command creates or lists macros for compose mode.

SYNTAX

map [*key-sequence* [*expansion*]]

map! [*key-sequence* [*expansion*]]

RELATED COMMANDS

unmap, unmap!

The **map** and **map!** commands create *macros* in command and compose modes, respectively. A macro is a binding between a key-sequence and its expansion: when you type the ***key-sequence***, Z-Mail replaces it with ***expansion***.

The ***key-sequence*** is usually one or two control characters, or a series of characters sent by one of the function keys of a particular terminal.

See *Using Macros* on page 102 for the ***key-sequence*** and ***expansion*** syntax, and for comments on the interactions of macros in the same and in different modes. See the *termcap* and/or *terminfo* manual pages of your UNIX system for information about finding the function key definitions for your terminal.

If you do not specify any arguments, **map** and **map!** commands display the list of macros and expansions in the appropriate mode. If you only specify a ***key-sequence***, they display the expansion associated with that string for the appropriate mode. Otherwise, they create a macro, associating the ***expansion*** you indicate with the specified ***key-sequence***.

The **unmap** command removes command macros, and the **unmap!** command removes composition macros.

mark

The **mark** command assigns a priority to messages that you can refer to later when sorting or searching.

SYNTAX **mark [–*priority*] [*messagelist*]** *where priority is A, B, C, D or E.*

RELATED COMMANDS **unmark, headers, flags**

EXAMPLE When you omit ***priority,*** Z-Mail marks the messages in ***messagelist*** with a **+** character, indicating a temporary mark that is not saved when you close the current folder or quit Z-Mail.

When you specify a priority such as **–A**, Z-Mail assigns that priority to the messages in ***messagelist***. For example, the command **mark –C 7** assigns priority **C** to message 7.

You can use the **sort** and **search** commands to order or select messages based on their priority. If you mark a message with both a temporary **+** mark and a letter priority, Z-Mail displays only the **+** mark and it takes precedence over the letter priority when you are sorting and searching.

Use the **unmark** command to remove temporary marks only. To remove the alphabetic priority (A-E) of a message, give the command:

mark – [*messagelist*]

Unless you clear the priority for a message in this way, its priority is saved when Z-Mail updates the active folder.

The command **headers –H:m** displays a list of messages that are marked. See **headers** command on page 157 for more information about manipulating marked messages.

merge

The **merge** command allows you to read messages from a specified folder into the active folder.

SYNTAX **merge [–N] *foldername***

RELATED COMMAND **folder**

The **merge** command copies messages from the folder named ***foldername*** into the active folder. The header summaries of the merged messages are displayed unless you specify the **–N** option. See the **folder** command for details.

The **merge** command can only appear in a pipeline if it is the first command; it returns a list of all the messages selected from ***foldername***. You cannot use the **merge** command in initialization files before Z-Mail has started and opened folders.

WARNING: The **merge** command does not directly support combining two open, non-updated folders. If the folder from which you want to **merge** is open, you should **update** it before performing the **merge**. If you do not do this, the current state of the merged-from folder is not reflected in the merged-in messages.

msg_list

The **msg_list** command parses a message list you specify and sets the **output** variable to that message list.

SYNTAX

msg_list [+ | –] [*messagelist*]

RELATED COMMAND

redraw

RELATED VARIABLE

output

This **msg_list** command is a parsing aid that accepts a message list as an argument. The ***messagelist*** may include metacharacters and negation syntax, as described in *Specifying Message Lists* on page 24. The ***messagelist*** may be piped into **msg_list**.

If ***messagelist*** is syntactically correct, **msg_list** sets the **status** variable to 0 (zero) and sets the **output** variable to the resulting message list, using only ranges and numerals. Metacharacters in messagelist are replaced with their numeric equivalents.

If the list is syntactically correct but produces an empty result, the **output** variable is set to empty string. If the ***messagelist*** is not syntactically correct, Z-Mail sets the **status** variable to –1.

As with the **from** command, the arguments – or + cause **msg_list** to set the current message to the first or last message in the list. Unlike **from**, however, the **msg_list** command never displays text as output.

menu

GUI and LITE ONLY

The **menu** command creates or lists user-defined menus and menu items in the GUI and Lite interfaces. Each menu item created with **menu** is associated with a Z-Script command that gets executed when the item is chosen.

SYNTAX

menu [*location*] [*options*] [*menu-name*] [*command*]

RELATED COMMANDS

unmenu, function

In the GUI and Lite interfaces, you use this command to create or modify menus that appear:

- on the menu bars of the Main, Message, or Compose window
- on pop-up menus in the message summaries area, the output area and the command line area of the Main window
- on the pop-up menu in the body area of the Message window.

Pop-up menus appear when the third mouse button (the far right button if you use your mouse with your right hand) is pressed while the mouse cursor is the appropriate area.

You may create four types of items using **menu:**

- menu titles appear in a menu bar, or are the labels within menus that cause submenus (or pullright menus) to appear.
- action items execute a Z-Script command or function when chosen.
- toggle items change the state of a Z-Mail variable or option between "on" and "off". Items that appear by default on the Options menu in the Compose window are examples of toggle buttons.
- separators are lines that appear on menus to separate items or titles.

You may also create and edit menus using the *Menus* dialog. See the *Extending Z-Mail* section of the *Z-Mail User Guide* for more information about this dialog.

Groups of menus that appear together on a menu bar or on a pop-up menu are specified in *menu lists*. There are seven primary default menu lists:

Window	Menu	Menu list name
main	menu bar	MainMenu
	summaries pop-up	MainSummariesPopupMenu
	output area pop-up	OutputStatictextPopupMenu
	command area pop-up	CommandlineAfPopupMenu
message	menu bar	MessageMenu
	message body pop-up	ReadMessageBodyPopupMenu
compose	menu bar	ComposeMenu

These menu lists are interchangeable with the button lists used with the **button** command. (In other words, button lists and menu lists are actually the same thing.)

Every **menu** command has a *context* associated with it, which specifies what part of the interface will be affected by the command. Contexts may be specified as either windows or menu lists. When you execute **menu** with no arguments, the menu list for the current context is displayed. By default, the context is the MainMenu menu list. You may change the context by specifying the **–menulist-context** or **–window-context** options described below.

Using the **menu** command, you may:

- add, remove, or change items on existing menus
- create new menu lists
- add new menu lists to menu bars or existing menus (as submenus).

Adding a new action item

To add a new action button, use this form of the **menu** command:

menu [*location*] [*options*] *item-name command*

location is the name of the button list to which it should be added. The following are valid location options:

–menulist *menulistname*

–menulist-context *menulistname*

menulistname may be the name of any default menu list or any user-defined menu list. When you specify a menu list, the item will appear in all places where that menu list is displayed.

Determining menu list names

In addition to the seven menu lists shown above, there are menu lists for every default menu and submenu. Before you can add an item to a menu, you need to determine the menu list name for that menu. Menu list names for most menus take the form:

*WindownameMenuname*Menu

For example, the menu list name for the Folder menu in the Main window is MainFolderMenu.

Submenus (such as the Priority menu) have menu list names as well; these are a bit more difficult to figure out. The best way to determine a submenu name is to use the **menu** command to display information about the menu the submenu appears on. For example, to find the menu list name for the Priority menu, use this command:

menu -menulist MainMessageMenu

The output of the command contains the following:

```
Menu:
 Label: (none)
 Name: mm_priority
 Window: MainMessageMenu
 Position: 8
 Menu: PriorityMenu
```

which indicates that the menu list name for the Priority menu is PriorityMenu.

Specifying contexts

The **–window context** and **–menulist-context** options are useful when you are specifying several menu items that all have the same location. When a **menu** command containing one of these options is executed, all items or menus created subsequently will appear in the same location, until another context is specified.

If no location is specified, buttons are added to the current context. (If you have never specified context, the default is the MainMenu menu list.)

Specifying options
options may be any of the following:

–no-msgs	if messages do not need to be selected in order for the item to execute
–position *number*	specifies the position of the item; items are numbered top to bottom starting from 1. (Or left to right, in the case of menu bars).
–final	to indicate that this item should always have the highest position number.
–separator	to indicate that this item is a separator line. (You must still provide an ***item-name*** if you specify this option, but can omit ***command***.)
–sensitivity *expr*	if the specified Boolean expression evaluates to false, the item is insensitive, or grayed out. If it evaluates to true, the item is sensitive.

Sennsitivity ***expressions*** may take the following forms:

expr : : = expr1 && expr2	(expr1 AND expr2)
expr1 || expr2	(expr1 OR expr2)
(expr)	(parentheses may be used for grouping)
!expr	(expr1 is not true)
expr1 == expr2	(expr1 equals expr2
expr1 != expr2	(expr1 does not equal expr2)
$variable, text-string, number	(expressions may also include Z-Mail variables, strings, and numbers.)

Expressions should be surrounded by single quotes.

item-name specifies the text that appears on the item. It may also be specified as **–label *item-name***. In either case, the item name should be surrounded by double quotes.

command specifies the Z-Script function or command that gets executed when the item is selected.

Adding a new toggle item

To add a new toggle item, use the **menu** command in the following form:

menu [*location*] [*options*] –value *expr item-name* [*command*]

location, ***options***, and ***item-name*** may be specified as described in the previous section.

–value *expr* indicates that the item is a toggle item, rather than an action item. When the expression evaluates to true, the item is toggled on; when the expression evaluates to false, the item is toggled off.

Value ***expression*** is restricted to one of the following forms, and should be surrounded by single quotes:

```
expr : : = A
           !A
A : := $variable              (returns contents of variable)
       $?variable             (retruns true only if variable is set)
       $variable:(value)      (returns value if variable is set to
                               value; returns null otherwise)
       $?variable:(value)     (returns true only if variable is set
to
                               value; returns false otherwise)
```

command is optional for toggle items; if it is specified, it will be executed when the item is chosen, after the state of the variable or option described by the value expression is altered.

Creating a new menu list

To create action items in a new menu list, use this form of the **menu** command:

menu –menulist *menulistname* [*options*] *item-name command*

To create toggle buttons in a new button list, use this form of the **menu** command:

menu –menulist *menulistname* [*options*] –value *expr item-*

***name* [*command*]**

You may use the **–menulist-context** option in place of the **–menulist** option in these commands.

Using a menu list in a menu bar

To install a menu list in a the menu bar of a window, use the **menu** command in the following form:

menu –window *windowname* –menu *newmenulistname* [*options*] *menu-name*

windowname may be **main, message,** or **compose.** ***menu-name*** is the title of the menu that appears on the menu bar.

Using a menu list as a submenu

To install a menu list as a submenu on an existing menu, use this form of the **menu** command:

menu -menulist *menulistname* -menu *newmenulistname* [*options*] *menu-name*

menulistname is the name of the menu list that describes the existing menu. ***newmenulistname*** is the menu list that is to become the submenu; ***menu-name*** is the title of the submenu.

Use the **unmenu** command to remove buttons from any window or button list.

The **menu** command has no effect in the command line interface.

EXAMPLES

The following command creates a menu item on Message menu in the Message window labeled “Quit” which executes the Z-Script **quit** function when pressed:

menu –menulist MessageMessageMenu "Quit" quit

The following command creates a toggle item on the Folder menu in the Main window that toggles the display of the Output area in the Main window:

menu –menulist MainFolderMenu –value ’$?main_panes:(output)’ "Output Area"

The following commands create a new menu list called MyMenus, which contains two buttons:

menu –menulist MyMenus "Quit" quit

menu –menulist MyMenus "Sign" compcmd insert-file ~/ .signature

The following command installs the MyMenus menu list in the Compose window menu bar, under a menu titled Other:

menu –window compose –menu MyMenus Other

The following command installs the MyMenus menu list as a submenu titled Other in the Compose window, on the Message menu:

menu –menulist ComposeMessageMenu –menu MyMenus Other

my_hdr

The **my_hdr** command creates custom headers that are included in your outgoing messages.

SYNTAX

my_hdr [*header* [: *text* [| *text* ...]]]

RELATED COMMAND

un_hdr

RELATED VARIABLES

no_hdrs, picky_mta

COMMAND USAGE

my_hdr	Displays all your headers
my_hdr *header*	Displays value of *header*
my_hdr *header: string*	Sets *header* to *string*
my_hdr *header*: [*string1* | *string2* | *string3*]	Prompts you with the listed choices, and sets the value of the header to the one you select.
un_hdr *header*:	Unsets *header*

EXAMPLES

my_hdr X-Phone-Number: +1 415-555-1234

To set a header, you must specify the first argument as a string containing no spaces or tabs and it must end with a colon (**:**). The rest of the command line is the ***text*** to be associated with the header you specify. If you use any quotes in the header and the header itself is not set in quotes, you should precede the quotes with a backslash (\). You should also precede with a backslash any pipe (|) characters, semicolons, or other metacharacters that Z-Mail might interpret as a command line modifier.

If you create a nonstandard ***header*** , or if you are unsure whether the ***header*** you're creating is standard, precede the ***header*** with **X–** to indicate that it is nonstandard.

If you set the **no_hdrs** variable, Z-Mail does not add your custom headers to outgoing messages.

Z-Mail treats the *From:* header specially. If you have set your own *From:* header, Z-Mail performs a simple test to determine whether the address you specify is valid. Any UUCP or domain address that directs mail to your login at the local machine should be acceptable, but certain system configurations may prevent some combinations from being recognized. If the address is valid, your *From:* header is used; otherwise, Z-Mail generates another address known to be valid and uses it instead.

Some mail transport agents are "picky" and do not allow Z-Mail to supply a *From:* header. In these cases, at send time Z-Mail replaces your *From:* header with one generated by the transport agent. See the **picky_mta** variable on page 299 for more information.

Note: You cannot set the *Date:* header. If you attempt to do so, Z-Mail does not generate an error message, but overrides your date with the correct system date and time corresponding to when the message is sent.

See also the *Envelope* dialog in the *Z-Mail User Guide*.

next

The **next** command displays the next undeleted and unsaved message that follows the current message.

SYNTAX **next**

SYNONYM **n**

RELATED COMMAND **previous**

RELATED VARIABLE **wrap**

If no messages are new or unread, the **next** command displays the next message that has not been deleted or saved, beginning with the message after the current message. If no messages match these criteria, **next** displays No more messages.

If the **wrap** variable is set, Z-Mail will also wrap around from the last message of the folder to the first.

open

The **open** command opens a folder and makes it the active folder, adding it to the list of open folders.

SYNTAX **open [*options*] *foldername***

SYNONYM **folder –a**

See the **folder** command on page 145 for more information about the ***options*** and ***foldername*** for **open** command. **open** is synonymous with **folder –a**.

page

Displays a file in a *pager*.

SYNTAX **page *filename* . . .**

In the GUI and Lite interfaces, the **page** command displays the contents of ***filename*** in a dialog. In the command-line interface, the file is displayed using the program named by the **pager** variable. See the **pager** variable on page 298 for more information.

If you specify more than one ***filename***, they are paged in order.

pick

The **pick** command is identical to the **search** command. For more information, see the **search** command on page 199.

SYNONYM **search**

pinup

GUI and LITE ONLY

The pinup command "pins up" a message in its own window, so that several messages may be displayed at the same time.

SYNTAX

pinup [*messagelist*]

RELATED COMMANDS

display, next, previous, top

Pinup works like **display**, except a "pinned-up" message is displayed in a new message reading window. Pin-up windows stay on the screen until they are explicitly dismissed, or the folder to which the messages belong is closed. You may display as many pin-up windows at a time as you like.

The **Pinup** command works like **Display**; all message headers are displayed with the message.

pipe

The **pipe** command directs a list of messages to a UNIX command as input.

SYNTAX

pipe [–p *pattern*] [*messagelist*] [*command*]

RELATED VARIABLES

alwaysignore

You may explicitly specify the ***messagelist*** or it may come from a *pipeline*. See *Piping Commands* on page 88 for details. If you do not specify or pipe a ***messagelist***, Z-Mail uses the current message. All headers are considered part of the message text for purposes of this command unless the value of the **alwaysignore** variable includes the value pipe.

The ***messagelist*** is simplified and placed in ascending order before execution by the **each** command.

EXAMPLE 1

For example, the following command sends the text and headers of messages 3, 5, and 7 to the UNIX *patch* utility:

```
each 3 7 5 pipe patch
```

Note that even though the ***messagelist*** was specified as **3 7 5**, the messages are sent to **pipe patch** in ascending order: **3 5 7**.

If you do not specify **–p** and you omit ***command***, Z-Mail searches for a line beginning with #! and feeds that line and all succeeding lines to */bin/sh*. In other words, a **pipe** command with no arguments treats the current message (from #! to the end) as a shell script. If #! is not present in the message, an error message is displayed.

EXAMPLE 2 The example below sends message 11 to */bin/sh* to run as a shell script:

pipe 11

If you specify a ***pattern*** with the **–p** option, Z-Mail searches each message for a line beginning with that ***pattern***. Z-Mail directs the matching line and all succeeding lines to ***command***. Note that ***pattern*** is not a regular expression; it is treated as a literal string.

An alternative form for ***pattern*** is:

/ pattern1 / , / pattern2 /

in which case displaying begins with the line containing ***pattern1*** and ends with the line containing ***pattern2***, inclusively. Patterns of this form must still match the beginning of a line.

You can also invoke the **pipe** command as **Pipe** (note capitalization), in which case only the body of the messages, and none of the message headers, are sent to ***command***.

When you set Z-Mail's **unix** variable, UNIX commands can appear anywhere in a Z-Mail pipeline except as the first command in a pipeline without explicitly using **pipe**. However, you must still specify **Pipe** to exclude all headers.

Note: All messages you list as arguments to **pipe** or **Pipe** are sent to the standard input of the same process as a continuous stream, in message number order! This may not be desirable when you extract shell (*sh*) scripts, so be careful. To avoid this behavior, use the **each** or **foreach** commands to pipe messages individually in ascending message number order.

preserve

The **preserve** command causes ***messagelist*** to be held in your system folder rather than being automatically deleted or moved by certain operations.

SYNTAX **preserve [*messagelist*]**

SYNONYM **pre**

RELATED COMMAND **unpreserve**

RELATED VARIABLES **hold, keepsave**

The **hold** variable (see page 281) causes all messages to be held in your spool directory automatically. The **keepsave** variable (see page 288) stops the **save** and **write** commands from marking messages for deletion.

Messages that are preserved are indicated in the header list by a P flag.

The **unpreserve** command removes the preserved status from ***messagelist***.

previous

The **previous** command displays the previous undeleted and unsaved message.

SYNTAX **previous**

RELATED COMMANDS **display, next, pinup**

RELATED VARIABLE **wrap**

If no messages are new or unread, the **previous** command displays the previous message that has not been deleted or saved, beginning with the message before the current message. If no messages match these criteria, the **previous** command displays No more messages.

print

This command is an alternate name for the **display** command and is identical to it in all respects. To print a message to a line printer, see the **lpr** command.

SYNONYM **display, type, p, t**

printenv

SYNTAX **printenv [*variable*]**

RELATED COMMANDS **setenv, unsetenv**

Displays the setting of the named environment variable ***variable***. If no ***variable*** is supplied, displays the settings of all environment variables. Environment variables are inherited from the environment of the shell that started Z-Mail, but may be changed during the Z-Mail session with the **setenv** command.

Child processes started by Z-Mail inherit Z-Mail's environment.

pwd

The **pwd** command displays the name of the current working directory.

SYNTAX **pwd**

RELATED COMMAND **cd**

RELATED VARIABLE **cwd**

This command also resets the variable **cwd**. To change the working directory, use the **cd** command.

quit

The **quit** command updates all open folders and exits Z-Mail.

SYNTAX **quit [–f]**

SYNONYM **q**

RELATED COMMANDS **update, folder, x**

RELATED VARIABLES **verify, hold, deletesave, keepsave, dead**

By default, all messages in your system folder that are not marked for deletion remain in your system folder. Z-Mail discards any messages you have marked for deletion (which by default includes saved messages). If you unset the **hold** variable, Z-Mail moves messages you have read from your system folder to your main mailbox. Your main mailbox is either
~/mbox, or the file described by the **mbox** variable.

If you specify the **–f** option, Z-Mail quits 'forcibly': all open folders are updated without asking for confirmation, and all suspended compositions are appended to the folder specified by the **dead** variable (see page 269). If any errors occur during updating, they are ignored.

To exit Z-Mail without updating any folders, use the **x** command.

When updating a folder other than the system folder, Z-Mail discards messages you have marked for deletion, and leaves all other messages including saved messages in that folder. See the **folder** and **update** commands for more details.

If you quit while you have unsent compositions in progress, Z-Mail prompts for confirmation that it should abort the compositions before quitting. For more information, see the **~z** tilde-escape command and the **resume** command.

read

SYNONYM **display**

The **read** command is identical to the **display** command; see **display** on page 137.

redraw

GUI and LITE ONLY

Redraws the screen.

SYNTAX **redraw [–all] [*messagelist*]**

The **redraw** command "redraws" all windows in the GUI interface, or redraws the screen in the Lite interface.

The redraw command with no arguments redraws only the selected messages in the Main Window.

If a ***messagelist*** is specified in the command or piped to redraw, those messages are redrawn and selected in the Main Window.

If **–all** is specified, all message summaries are redrawn in the Main Window.

remove

Removes (deletes) ***filename***.

SYNTAX **remove [–i] [–I] *filename*. . .**

SYNONYM **rm**

RELATED COMMAND **rmfolder**

RELATED VARIABLE **verify**

The **remove** command removes files, like the UNIX *rm* command. You may specify more than one ***filename*** to remove, separated by spaces. If ***filename*** includes one or more space characters, it must be enclosed in quotes.

The **–I** (capital i) option prompts for confirmation before removing files that are not mail folders.

When **remove –I** removes a folder, if the folder has an external index in your **index_dir** (see page 284), then you'll be asked whether you want to remove the external index as well.

The **–i** option causes **remove** to prompt for confirmation before removing ***filename***. External indices to removed folders are removed silently.

***NOTE*:** The **remove** command does not perform expansion of filename metacharacters, such as *.

rename

Renames files and folders.

SYNTAX

rename [*oldname*] *newname*

Rename changes the name of an open, writable folder or file specified by ***oldname*** (or the current folder if ***oldname*** is omitted) to ***newname***. The filename metacharacters **+** and **~** are expanded.

If ***newname*** exists, a confirmation dialog will appear, asking if you want to delete the existing file before renaming.

When you **rename** an open folder, it is *updated* in the process.

NOTE: Z-Mail cannot, in general, **rename** a file or closed folder across devices. Two different disks, or two different partitions of the same disk, are different devices. However, opened folders can be renamed across devices, since Z-Mail copies the folder from ***oldname*** to ***newname***, instead of creating a link.

reply

The **reply** command starts a 'reply' composition to the sender of each message in a ***messagelist***.

SYNTAX

reply [*messagelist*] [–r *path*] [*options*] [*recipients*]

SYNONYMS

r, replysender, respond

RELATED COMMANDS

replyall, mail

RELATED VARIABLES

reply_to_hdr, auto_route, domain_route, metoo, known_hosts, in_reply_to

The **reply** command accepts all options supported by the **mail** command. See the **mail** command on page 171 for details.

The **–r** option prefixes the address of each recipient in the address list with the indicated path. This overrides the value of the **auto_route** variable, but has the same effect.

When constructing a return mail address to the author of a message, the **reply** command searches for special mail headers in the author's message that indicate the most efficient mail path for return mail. By default, Z-Mail searches for the *Reply-To*, *From*, and *Return-Path* headers, in that order. If Z-Mail finds none of these fields in the message, it parses the first line of the message to determine the return path.

If you want Z-Mail to search the headers in a different order, set the **reply_to_hdr** variable to that list of headers in the order you wish them to be searched. If you set **reply_to_hdr**, but specify no string, Z-Mail searches the first *From* header it encounters regardless of what headers the author's message contains. See the **reply_to_hdr** variable on page 306 for more information.

replyall

The **replyall** command works like the **reply** command, except that it starts a composition to the sender *and all recipients* found in the *To* and *Cc* headers of each message in a ***messagelist***.

SYNTAX **replyall [*messagelist*] [–r *path*] [*options*] [*users*]**

SYNONYM **R**

RELATED COMMAND **reply**

RELATED VARIABLES **reply_to_hdr, auto_route, domain_route**

See the **reply** command for details on using **replyall**.

When you reply to all recipients of a message using the **replyall** command, Z-Mail determines the original author's address from the message headers. There is no way to determine the best path to the other recipients of the message from message headers aside from taking their addresses directly from the *To:* and *Cc:* lines.

EXAMPLE **replyall 3, 4, 5 –i 4, 5, 7 –s "Meeting Agenda" Research**

In the above example, you reply to messages 3, 4, and 5 and include the text from messages 4, 5, and 7 in the body of the reply. Z-Mail sets the subject to Meeting Agenda, and automatically sends the reply to all the authors and recipients of messages 3, 4, and 5, as well as to addresses contained in the alias Research.

resume

COMMAND-LINE ONLY

SYNTAX **resume [*job*]**

SYNONYM **fg**

RELATED COMMAND **jobs**

The **resume** command allows you to continue composing a message that you suspended with the **~z** command. The optional ***job*** argument is a number identifying the message you wish to resume. You can obtain a list of the current messages being composed by using the **jobs** command (see page 169). If you do not specify a ***job***, the most recently suspended composition is resumed.

This command has no effect in the GUI and Lite interfaces because compositions have their own input windows and do not need to be suspended.

retain

The **retain** command tells Z-Mail which headers to show when displaying messages.

SYNTAX **retain *headerlist***

RELATED COMMANDS **ignore, unretain**

Headerlist contains the names of headers Z-Mail is to display. You may also use the *Headers* dialog of the GUI and Lite interfaces to select which headers to show and ignore.

EXAMPLE To display only the headers *From, Date, Subject, To,* and *Cc*:

retain from date subject to cc

You may also use filename metacharacters in ***headerlist***; see the **ignore** command on page 168 for details.

To reverse the effects of **retain**, use **unretain**.

return

Use the **return** command to tell Z-Mail to terminate the current function and continue execution at the next Z-Script statement in the calling function. Z-Mail recognizes **return** commands only within functions.

You can specify an optional ***status_code*** that sets the value of the **status** variable so the calling function can determine how the returning function terminated.

SYNTAX

return [*status_code*]

EXAMPLE

This following script displays a dialog, then terminates processing and returns control to the calling routine with a status code of –1. (The **status** variable will be set to –1.)

```
if $report no <= 0
    echo –d "Report request canceled by user."
    return –1
endif
```

rmfolder

Removes (deletes) folders or files.

SYNTAX

rmfolder [–x] [–i] [*foldername* . . . | *filename* . . .]

RELATED COMMAND

remove

RELATED VARIABLE

verify

The **rmfolder** command is equivalent to **remove –l** (see page 192), but also provides a **–x** option. When **–x** is specified, only the external index of ***foldername*** is removed, not ***foldername*** itself.

save

The **save** command saves messages to a file or folder.

SYNTAX

save [–s | –S |–a | –A | –f] [*messagelist*] [*filename* | *directory*]

SYNONYMS

s

RELATED COMMANDS **copy, write**

RELATED VARIABLES **mbox, keepsave, deletesave**

If you do not specify any arguments, the **save** command appends the current message to the folder specified by the **mbox** variable (usually *~/mbox*). If you specify or pipe in a ***messagelist***, Z-Mail saves the messages specified in the list.

If you specify ***filename***, it is used instead of **mbox**. If ***filename*** exists and you are allowed to write to it, Z-Mail appends each message in ***messagelist*** to the end of the file. If ***filename*** does not exist, it is created.

If you use the **–s, –S**, **–a**, or **–A** option, the last argument to the command is treated as a ***directory*** into which the files are written rather than a ***filename***. If you omit ***directory***, the current working directory is used.

The **–s** option tells Z-Mail to use the subject of the message as ***filename***. Spaces, slashes (/), and other unusual characters in the message's subject are changed to underscores (_) when included in the ***filename***.

Messages with identical *Subject* headers are saved to the same file. The **–S** option works like **–s**, except all messages are saved to the same file; the filename is constructed from the message with the lowest message number.

The **–a** and **–A** options work like **–s** and **–S**, except the filename is taken from the name(s) of the author(s) of the message(s).

If you specify the **–f** option, the file is overwritten and its current contents are lost. Note that the **–f** option has no effect when used in conjunction with the **–a** or **–s** options. For compatibility with older versions of Z-Mail, the exclamation point (**!**) may be substituted for **–f**. Do not precede the **!** with a dash.

If the current folder is the system folder or **deletesave** is set, Z-Mail automatically purges messages that are marked as saved when the folder is updated or closed. If the **keepsave** variable is set or the current folder is not the system folder, then messages are not purged.

The **write** command differs from the **save** and **copy** commands in that the message headers are not saved in the file along with the body of text. The **copy** command is like **save** except that messages are not marked as saved, regardless of the value of the **keepsave** variable.

saveopts

The **saveopts** command saves the settings of all variables, aliases, **cmd** definitions, and functions to a Z-Mail initialization file ***filename***.

SYNTAX **saveopts [–A] [–a] [–f] [–o *option*] [*filename*]**

By default, settings are saved to your ~/.zmailrc file. If you specify ***filename***, Z-Mail writes a new initialization file to that ***filename***, overwriting ***filename*** if it exists. You may edit ***filename*** by hand after it is written by **saveopts**.

WARNING: Changes you may have made to an existing ***filename*** are lost when the **saveopts** command overwrites it. You are asked to confirm this command if ***filename*** exists. Use the **–f** option to supress the confirmation.

By default, only settings that differ from those found in the system initialization file are saved. For more information on initialization files, see *Initializing Z-Mail* on page 63. If you specify the **–A** option, all settings, including the ones that don't differ from the system initializiation file, are saved.

If the **–a** option is specified, settings are appended to the file.

options are used to specify what types of settings should be saved.

set	variable definitions
my_hdr	user-defined outgoing headers
alias	mail aliases
alts	your alternate e-mail addresses
retain	specification of headers to be displayed on incoming messages
ignore	headers to be hidden on incoming mail
cmd	command abbreviations
filter	mail filters

functions	user-defined functions
button	user-defined buttons
menu	user-defined menus
map	command=line mode key mappings
map!	composition mode key mappings

To save several types of settings, you must specify **–o** multiple times:

saveopts -o button -o function -o filter

The **saveopts** command is intended for novice users who don't need to place custom Z-Script commands in their initialization files, and who therefore do not wish to learn Z-Script syntax. Experienced users are encouraged to edit their existing initialization file by hand to obtain finer control over their Z-Mail configuration.

The **saveopts** command cannot override some settings that Z-Mail reads from the *system.zmailrc* file. To change these system settings, you must edit your *.zmailrc* file manually or start Z-Mail with the command **zmail –noinit** to prevent it from reading the *system.zmailrc* file.

The following variables are not saved in ***filename*** by **saveopts**, since they usually change from session to session:

cmd_help cwd gui_help home hostname user

search

The **search** command selects messages that match search criteria you specify.

SYNTAX **search [–r *messagelist*] [*options*] *pattern***

SYNONYM **pick**

RELATED VARIABLES **date_received**

The ***pattern*** is a *regular expression* as described by the UNIX *ed* command's manual page. By default, **search** looks for ***pattern*** in the headers and body of all the messages in the active folder. If a ***messagelist*** is specified or piped in, **search** will only examine those messages.

The **search** command has the following options:

+*n*	Select only the first ***n*** messages matched.
–*n*	Select only the last ***n*** messages matched.
–ago *when*	Search for messages relative to today's date.
–d *–date*	Search for messages sent before ***date***.
–d *+date*	Search for messages sent after ***date***.
–b *date1 date2*	Search for messages sent between ***date1*** and ***date2***.
–e	Interpret all remaining arguments as ***pattern***.
–f	Search for ***pattern*** in *From:* header.
–h *header*	Search for ***pattern*** in specified *header* only.
–i	Ignore case of letters when searching.
–p *priority*	Select messages with the specified ***priority*** (A-E)
–r *messagelist*	Search only the messages in ***messagelist***.
–s	Search for ***pattern*** in *Subject:* header.
–t	Search for ***pattern*** in *To:* header.
–x	Select messages not matching ***pattern***.

You can abbreviate the **–ago** option as **–a**. You can only specify one of the **–a**, **–d**, and **–p** options at a time, but you may specify multiple **–p** options to select several priorities. The **-f**, **-h**, **-s**, and **-t** options can be used in combination, but cannot be repeated.

Z-Mail searches all messages including those marked for deletion. Z-Mail scans entire messages for ***pattern*** unless you specify the **–a**, **–d**, **–f**, **–h**, **–p**, **–s**, or **–t** options. The **–x** option may not be used in conjunction with the +**n** and **–n** options.

For the **–d** and **–b** options, ***date*** is of the form:

month/day/year

For the **–d** option, you can put a dash (–) prefix on the ***date*** you specify to find messages older than that ***date***. If you omit fields in ***date***, the fields default to today's values.

EXAMPLES

search –d 4/20	On April 20, this year
search –d –/2/90	On or before the 2nd, this month, 1990
search –d +5/4	On or after May 4 of this year
search –d /	Today only

At least one **/** character must be present in ***date***. Strong date validity checking is not performed; for example, **2/30** is accepted as a valid date.

The ***when*** argument for the **–ago** option is specified as the number of days followed by the word **days**, or the number of weeks followed by the word **weeks**, and so on with **months** and **years**. You may truncate days, weeks, months and years to one character. The following expressions are equivalent:

```
search –ago 1 day, 2 weeks
search –ago 2 Weeks 1Day
search –ago 2 w,1day
search –a 2w1d
```

These examples find all messages that arrived within the 1-day period that was 2 weeks and 1 day ago. All **–ago** dates collapse into day time segments; a **month** is 30.5 days long. If you require more precise date selection, use the **–d** option and indicate specific dates.

Also note that the **–ago** option allows before (–) and after (+) arguments. As a result, you may select all messages older than 1 week with:

```
search –ago –1 week
```

The following command finds all the messages between the dates February 5, 1990 and February 5, 1991 that also contain the subject meeting minutes, and sends them to the printer.

```
search –d –2/5/91 | search –d +2/5/90 | search –s
    "meeting minutes" | lpr
```

This command deletes messages that have Re: in the Subject header.

```
search –s Re: | delete
```

The next command finds all messages from *frank* in the folder named *+project*.

```
folder +project | search –f frank
```

The following command searches all messages that have the header *Return-Path:* and determines if the string "hqvax" is in the header. The header name is matches case-insensitively, while the text pattern **hqvax** is matched cases-sensitively.

```
search –h return-path hqvax
```

The next command finds all messages that are a week or less old and saves them in the folder *current* in the directory indicated by the **folder** variable.

search –ago +1w | save +current

This command finds the first three messages containing the string expo.

search +3 expo

The next command finds the next message with the same subject as the current message.

eval –h "search +2 –r –$ –s %s" | search –1

set

The **set** command assigns a value to a Z-Script variable.

SYNTAX

set *variable1 variable2 . . .*	FOR BOOLEAN VARIABLES
set *variable* = *value*	FOR SINGLE-VALUE VARIABLES
set *variable* = "*value1 value2*"	FOR MULTI-VALUE VARIABLES
set *variable* = *value1,value2*	FOR MULTI-VALUE VARIABLES
set *variable* += *value*	FOR MULTI-VALUE AND STRING VARIABLES
set *variable* –= *value*	FOR MULTI-VALUE VARIABLES
set – *arg1 arg2 arg3. . .*	TO REWRITE $*
set *?variable*	FOR HELP ABOUT VARIABLES

RELATED COMMAND

unset

If you do not specify any arguments, the **set** command displays the values of all variables. If you specify ***variable***, the **set** command assigns a value to the named variable.

EXAMPLES

When you are setting the value of a boolean variable (turning it "on"), or piping in a value, you should not specify an equal sign.

set hold
search –i –s "report" | set reports

You may set more than one boolean variable in a set command by listing the names of all the variables on one line:

set hold auto_route fignore

You set single-value variables by entering the variable's name on the left side of the equal sign and its value on the right. If a string value contains blanks, make sure you enclose it in quotes:

```
set crt = 20
set editor = vi
set autosign = signature.file
set realname = "Wolfgang Mozart"
```

You set multivalued variables in the same way as single-valued variables. You may separate the values in the list that you assign to a multivalued variable with commas or spaces. You must enclose the list of values in quotes if you use spaces to separate the values:

```
set autodismiss = "compose print folder buttons"
```

You can add a value to a multivalued variable, or append to a string variable, with **set +=**

```
set autoiconify += "compose"
```

You can remove a value from a multivalued variable with **set –=**. For example:

```
set autoiconify –= "compose"
```

The special command **set ?all** displays all variables internal to Z-Mail and a brief description of what they do. In the GUI and Lite interfaces, **set ?all** brings up the *Variables* dialog. The command **set ?*variable_name*** displays the same information for one variable instead of all variables.

You can change the value of the special argument-list variable **$*** with **set –**. For example, if the function **rewriter** is defined as follows:

```
function rewriter() {
        echo "Argument list is:" $*
        set – "$3" "yam" "$1"
        echo "Rewritten argument list is:" $*
}
```

Then a call to **rewriter** will look like:

```
rewriter "pie" "we" "made"
Argument list is: pie we made
Rewritten argument list is: made yam pie
```

See the function variables **$***, **$0**, **$1**, etc., starting on page 317 for more information. See also the **shift** command on page 206 for another mechanism for changing the argument list.

The **unset** command clears the values of Z-Mail variables. You may unset all variables by issuing the command **unset ***. This is not recommended, since Z-Mail requires that certain variables remain set to function properly.

You can create your own variables for use in scripts. In the following example, the variable **reports** is set to a list of messages output from a **search** command via a pipe.

search –s "Status Reports" | set reports

The **reports** variable now contains a message list which can be used as the message list argument to any command which accepts a list. For example:

mail –i $reports rhagan

sends mail to *rhagan* and includes the text of all the messages indicated by the value of the **reports** variable.

See *Z-Mail Variables* on page 223 for details about internal Z-Mail variables.

setenv

Sets environment variables that Z-Mail passes to subprocesses it creates.

SYNTAX

setenv *variable* [*value*]

RELATED COMMANDS

printenv, unsetenv

Environment variables describe the “environment” in which a process runs. Z-Mail inherits environment variables from the shell it is started in, and passes those variables, along with modifications and additions you make with **setenv**, to subprocesses it creates. If you omit ***value***, the ***variable*** is set.

Environment variables are usually uppercase by convention.

EXAMPLE

A commonly-used environment variable is *TMPDIR*, which you can set to the name of a directory in which programs should keep their temporary files. Z-Mail and many other programs will respect the setting of *TMPDIR*. For example:

setenv TMPDIR /usr/tmp

will cause temporary files to be written to */usr/tmp*. To display the values of environment variables, see the **printenv** command on page 190.

sh

The **sh** command runs a UNIX command in a shell.

SYNTAX **sh [–t *timeout*] [–m *message*] [*command*]**

SYNONYM **shell**

RELATED VARIABLE **unix**

The kind of shell that is started is named by the *SHELL* environment variable, or */bin/sh* by default. If you specify a ***command***, then the ***command*** is executed in the shell. If ***command*** includes semicolons (**;**) or pipe characters (**|**), it must be enclosed in quotes and special characters must be escaped with backslashes (\ ).

NOTE: If the Z-Script variable **shell** is defined, its value overrides the setting of the environment variable *SHELL*.

If the ampersand character (**&**) is at the end of the ***command***, it executes in the background, returning control to Z-Mail immediately.

Command-line

In the command-line interface, the **sh** command with no arguments starts an interactive shell from Z-Mail. When you are finished with the shell, give the *exit* command at the shell prompt to return to Z-Mail.

If you supply a ***command***, the shell will run in the foreground, then automatically return control to Z-Mail when the ***command*** exits.

If you specify the **–m** option, the ***message*** is displayed immediately before the ***command*** is executed.

The **–t** option is ignored in the command-line interface.

GUI

In the GUI interface, the **sh** command with no arguments will create a new window named by the **window_shell** variable, which should automatically start the shell named by the

SHELL environment variable. Z-Mail will wait for the shell to exit before continuing.

If a ***command*** is specified, then it is run in the shell. Output from the command appears in the *Output* area of the Main Window.

Lite

In the Lite interface, the **sh** command with no arguments performs:

screencmd –p sh $SHELL

This causes a new shell identified by the environment variable *SHELL* to run, suspending Lite until you *exit* the shell. See the Lite on-line help for more information about the **screencmd** command.

GUI and Lite

When you supply a ***command*** in the Lite or GUI interfaces, the command is started in a shell. In the Lite interface, if the ***command*** takes longer than a few seconds to complete, a Task Meter appears. In the GUI interface, no Task Meter appears unless you specify the **–m** option.

In the Lite interface, if ***command*** is an interactive program, you should precede the **sh** command with **screencmd –p**, to allow the command to take control of the terminal.

If you specify **–m** ***message***, the ***message*** is placed in the Task Meter.

If you specify **–t** ***timeout***, if ***timeout*** seconds pass before the command completes, the Task Meter will appear, optionally displaying ***message*** if specified.

shift

The **shift** command parses the arguments of a user-defined function.

SYNTAX

shift [–m] [*count*]

RELATED VARIABLE

output

When you write a Z-Script function that requires arguments, you can use the **shift** command to parse and identify the individual arguments.

If you specify the numeric argument ***count***, the **shift** command removes that number of parameters from the beginning of the argument list. If you do not specify ***count***, the **shift** command removes a single argument from the beginning of the argument list.

The **–m** option tells Z-Mail to remove any numeric message list that precedes the current argument list. You can set a variable to this message list for future reference by setting the variable equal to **$output** (the value of the **output** variable), or by piping the output of the **shift** command to the **set** command.

EXAMPLE 1

The following function called **strip** demonstrates the effect of the **shift** command. This sample script uses the argument variables **$1**, **$2**, **$3**, etc., to refer to the arguments a user enters when they call the **strip** function.

```
function strip () {
     echo "The list of arguments is:" $*.
     echo "The first argument is" $1.
     shift 2
     echo "After shifting, the first argument is" $1.
}
```

If you enter this command:

```
strip aa bb cc dd ee
```

Z-Mail runs the **strip** function, passing it the arguments, and displays:

```
The list of arguments is: aa bb cc dd ee.
The first argument is aa.
After shifting, the first argument is cc.
```

EXAMPLE 2

The next example demonstrates how you can use **shift** to strip off any message numbers at the beginning of an argument list. Essentially, **shift** looks for numeric values at the beginning of the argument list and evaluates the numeric values as a message list.

```
function getmsg () {
     echo "The list of arguments is:" $*.
     shift –m | set mlist
     echo "After shifting, the message list is" $mlist.
     echo "After shifting, the argument list is" $*.
```

```
}
```

If you enter the command:

getmsg 3 4 5 8 xx yy zz

Z-Mail runs the **getmsg** function, passing it the argument list, and displays:

```
The list of arguments is: 3 4 5 8 xx yy zz.
After shifting, the message list is: 3-5 8.
After shifting, the argument list is: xx yy zz.
```

EXAMPLE 3

The next example shows how you can use the value of the **output** variable to set the message list:

```
function getalt () {
      echo "The list of arguments is:" $*.
      shift –m
      set mlist = $output
      echo "After shifting, the message list is:" $mlist.
      echo "After shifting, the argument list is" $*.
}
```

If you enter the command:

getalt 7 4 1-3 taxicab

Z-Mail runs the **getalt** function, passing it the argument list, and displays:

```
The list of arguments is: 7 4 1-3 taxicab.
After shifting, the message list is: 1-4 7.
After shifting, the argument list is: taxicab.
```

sort

The **sort** command orders messages in the active folder by author, date, status, priority or subject.

SYNTAX

sort [–i] [[–r] –a | –d | –l | –p | –R | –s | –S]

RELATED VARIABLE

sort

The **sort** command recognizes the following options:

–i	Ignore case in alphabetical sorts.
–r	Reverse sort order of next option.

–a	Sort by author alphabetically.
–d	Sort by date.
–l	Sort by length of message.
–p	Sort by message priority.
–R	Sort by subject including *Re:*
–s	Sort by subject alphabetically.
–S	Sort by message status (the default).

If you specify no arguments, the **sort** command orders messages by status, in the following sequence:

Indicator	***Status***
N	New
U	Unread
P	Preserved
r	Replied-to
p	Printed
+	Marked
blank	Read
*	Deleted

If messages have the same status, the **sort** command then orders by priority.

If you sort by priority, Z-Mail orders marked messages first, followed by messages of priority A, then B, C, D, E, and finally messages having neither a mark nor a priority. See the **mark** command for information on attaching marks and priorities to messages.

If you specify only the **–r** option, the **sort** command orders by reverse message status (E through A).

If you specify the **–d** option, Z-Mail orders messages by the date they were sent. If the variable **date_received** is set, Z-Mail orders messages by the date you received them.

EXAMPLES

If you specify more than one sort option, the **sort** command applies them in left-to-right order. Thus:

sort –a –d

sorts messages by author and, if the author of any set of messages is the same, sorts within that set by date.

The **–r** option applies to only the next option encountered. Therefore, to sort by both reverse author and reverse date requires:

sort –r –a –r –d

You can also group options in the **sort** command:

sort –ra –rd
sort –rard

You can also sort a consecutive sublist of messages by using pipes. You may specify the message list on the command line:

10–20 | sort –d

sorts messages 10 to 20 by date.

If you set the **sort** variable, the **sort** command orders messages each time Z-Mail activates your system folder as the current folder.

Note: For compatibility with previous versions of Z-Mail, sort options are recognized even if they are not preceded by a dash (–). Also, if you specify a dash separated by spaces from any following arguments, Z-Mail interprets it as **–r**.

source

The **source** command reads Z-Script commands from a file.

SYNTAX **source [*filename*] [*arguments*]**

RELATED VARIABLE **home**

RELATED COMMAND **saveopts**

If you do not specify a ***filename***, Z-Mail searches for a file in the following order:

- the file specified by the environment variable *ZMAILRC*
- the file specified by the environment variable *MAILRC*
- *.zmailrc* in the your home directory

If you specify a ***filename*** on the command line, Z-Mail sources that file.

If you specify ***arguments***, they are made available to the Z-Script commands in the file via the “positional parameters” **$1**, **$2**, **$3**, and so on. See the **$1, $2, . . .** variable on page 318 for more information.

For more information, see *Initializing Z-Mail* on page 63.

stop

The **stop** command causes Z-Mail to send a *stop* signal to itself on UNIX systems with job control.

SYNTAX

stop

RELATED COMMAND

iconify

The stopped Z-Mail can be resumed with the *fg* (foreground) command at the shell prompt.

In the GUI mode, **stop** closes the Z-Mail windows to icon form, but does not actually stop the program. This is equivalent to the **iconify** command.

stty

COMMAND-LINE ONLY

Emulates the UNIX *stty* command, which adjusts terminal settings.

task_meter

GUI and LITE ONLY

Brings up the Task Meter dialog to tell the user an activity is taking place.

SYNTAX

task_meter [–on | –off | –check | –wait] [message]

The **task_meter** command is useful in scripts and functions to tell the user that a time-consuming task is in progress. If a message is specified, it is displayed in the Task Meter.

If **–on** is specified, that Task Meter appears.

If **–off** is specified, the Task Meter disappears.

If **–check** is specified, the Task Meter disappears and set the status variable to 1 if and only if the *Stop* button in the Task Meter was pressed.

If **–wait** is specified, Z-Mail stops processing Z-Script commands until the Stop button is pressed.

thwart

NOT AVAILABLE IN Z-MAIL BASIC

Special command for preventing interposing functions from being executed.

SYNTAX

thwart

RELATED COMMAND

interpose

When called from an interposing function, **thwart** prevents all remaining interposers and the command itself from being executed. If the function is an after-command interposer, **thwart** substitutes its argument for the return value of the command, but does not otherwise affect execution of the command. Like **return**, **thwart** immediately exits the calling function and sets the variable status to its integer argument. If the function is interposed on an operation, the status is sometimes replaced by the status of the command performing the operation.

top

The **top** command displays the top few lines of messages.

SYNTAX

top [*messagelist*]

If you specify ***messagelist***, the top command displays the top few lines of each message in ***messagelist***. Otherwise, **top** displays the first few lines of the current message.

You can specify the number of lines by setting the **toplines** variable. The value of **toplines** defaults to the value of the **crt** variable.

trap

Installs *signal handlers* that execute when a UNIX signal is received by Z-Mail.

SYNTAX

trap [[*command*] *signal* [*signal . . .*]]

The **trap** command installs ***command*** as a *signal handler*. The ***command*** will be executed when the UNIX signal ***signal*** arrives. The ***signal*** must be specified as an integer, not as the signal's name. The ***command*** must be a single-word Z-Script command or function name.

If no arguments are provided to **trap**, the current signal handlers are displayed.

If no ***command*** is provided, the signal handlers for the listed ***signals*** are removed.

The following signals cannot be trapped by Z-Mail, and are therefore not usable in the **trap** command:

ILL (4)	KILL (9)	BUS (10)
SEGV (11)	PIPE (13)	ALRM (14)
STOP (17)	CHLD (20)	CONT (19)
TTIN (21)	TTOU (22)	VTALRM (23)
TSTP/CLD (18)		

type

The **type** command is an alternate name for the **display** command.

SYNONYM

display, print, t, read

unalias

The **unalias** command removes previously defined aliases. Use the **alias** command to define a new alias.

SYNTAX

unalias [*aliasname* | *]

RELATED COMMAND

alias

You may delete all previously defined aliases with:

unalias *

See the **alias** command on page 117 for more information.

unbutton

The **unbutton** command deletes previously defined GUI and Lite Main Screen buttons. Use the **button** command to define new buttons.

SYNTAX

unbutton [*label* | *]

RELATED COMMAND

button

You may delete all previously defined buttons with:

unbutton *

See the **button** command on page 124 for more information.

uncmd

The **uncmd** command deletes command aliases previously defined by **cmd**.

SYNTAX

uncmd *name*

RELATED COMMAND

cmd

You may delete all previously defined command aliases with:

uncmd *

See the **cmd** command on page 131 for more information.

undelete

The **undelete** command reverses the effect of a **delete** command.

SYNTAX

undelete [*messagelist* | *]

RELATED COMMANDS

delete, quit, update

RELATED

deletesave

The **undelete** command removes the asterisk (*) status indicator for all messages in ***messagelist***.

Note: You can only use **undelete** to recover messages that are marked for deletion, but have not actually been deleted by an **update** or **quit** command. Once you update a mail folder, all messages marked for deletion are lost permanently.

See the **delete** command on page 134 for more information.

undigest

The **undigest** command bursts multi-message digests into individual messages.

SYNTAX

undigest [–m] [–p *pattern*] [*messagelist*] [*foldername*]

A *digest* is a mail message which is a collection of other mail messages mailed to a *moderator* by other users. The moderator compiles all the messages into a specially formatted message (the *digest*) and sends the digest to recipients.

If you specify the **–m** option, the **undigest** command merges the new messages into your active folder.

If you specify a ***foldername***, the **undigest** command creates a new folder named ***foldername*** and stores the new messages in that folder.

If you specify the **–p** option, you can indicate an alternate pattern that Z-Mail should recognize as the separator when breaking the digest into individual messages. Z-Mail matches ***pattern*** against character strings at the beginning of each line of the digest. The default pattern is eight hyphens (––––––––). The hyphen pattern is the de facto USENET standard digest article separator and should work for most digests, but some use another separator. The **–p** option is also useful for bursting forwarded messages out of a wrapper message.

EXAMPLE

The following command bursts out messages forwarded by another user of Z-Mail and merges them into the current folder.

undigest –m –p "–– Forward"

If you specify ***messagelist***, each digest in the list is disassembled to the same ***foldername*** you specify. If you do not specify ***foldername*** or the **–m** (merge) option, then the **undigest** command creates a temporary folder. The name of the temporary folder is generated from the *Subject* header of the digest.

unfilter

The **unfilter** command removes filters previously defined by the **filter** command.

SYNTAX **unfilter [*name* | *]**

RELATED COMMAND **filter**

You may remove all previously defined filters with:

unfilter *

See the **filter** command on page 143 for more information.

unfunction

The **unfunction** command removes functions previously defined by the **function** command.

SYNTAX **unfunction [*name* | *]**

SYNONYM **undefine**

RELATED COMMAND **function**

You may delete all previously defined functions with:

unfunction *

See the **function** command on page 155 for more information.

un_hdr

The **un_hdr** removes the definitions of headers created by the **my_hdr** command.

SYNTAX **un_hdr [*header* | *]**

RELATED COMMAND **my_hdr**

RELATED VARIABLE

no_hdrs

The **un_hdr** command deletes the definition of the specified header. To remove the definitions for all personalized headers, issue the command:

un_hdr *

If you set the **no_hdrs** variable, Z-Mail does not add your personal headers to outgoing messages. The **no_hdrs** variable does not delete the definitions of personal headers; it merely stops Z-Mail from using them.

To create your own headers, see the **my_hdr** command.

unhide

Make previously hidden messages visible again.

SYNTAX

unhide [*message-list*]

RELATED COMMANDS

flags, hide

The **unhide** command changes the status of messages from hidden. Hidden messages are not displayed in message summaries, unhidden messages are.

If no list of messages is given, only the hidden flag is removed from only the current message.

uniconify

GUI ONLY

Uniconifies an iconified Z-Mail.

SYNTAX

uniconify

RELATED COMMANDS

iconify, stop

unignore

The **unignore** command cancels the suppression of specific headers when Z-Mail displays messages.

SYNTAX

unignore [*header* | *]

RELATED COMMANDS

ignore, headers

You may tell Z-Mail to display all previously ignored headers by entering:

unignore *

No other metacharacters are recognized by **unignore**.

See the **ignore** command on page 168 for more information.

unmark

The **unmark** command clears temporary marks you placed on messages using the **mark** command.

SYNTAX

unmark [*messagelist*]

RELATED COMMAND

mark

A temporary mark appears as a **+** character immediately preceding the message status. Use the **unmark** command to remove these temporary marks.

You may tell Z-Mail to clear all the temporary marks in your active folder by entering:

unmark *

Z-Mail also allows you to assign a permanent priority to a message; this priority is indicated by A, B, C, D, or E. For information on removing these priority marks, see the **mark** command on page 175 and the **flags** command on page 144.

unmenu

The **unmenu** command removes:

- menus from menu bars
- items from menus
- submenus from menus

SYNTAX

unmenu [*–menulist*] *menuname*

RELATED COMMAND

menu

menuname is the name of a menu item. You may need to use the **menu** command to determine the names of some default items. See the **menu** command on page 178 for more information.

unpreserve

The **unpreserve** command clears the *preserved* status of messages that have been preserved in the system mailbox with **preserve**.

SYNTAX

unpreserve [*messagelist*]

RELATED COMMAND

preserve

The ***messagelist*** may be piped in from another command. If ***messagelist*** is omitted, the currently selected messages are unpreserved.

See the **preserve** command on page 189 for more information.

unretain

Removes ***headers*** from the list of retained ***headers***.

SYNTAX

unretain *headers*

RELATED COMMANDS

retain, ignore, unignore

RELATED VARIABLE

alwaysignore

You may **unretain** all headers that you have previously retained by executing:

unretain *

No other metacharacters are recognized by **unretain**.

See the **retain** command on page 195 for more information.

unset

The **unset** command clears the values of variables previously defined using the **set** command.

SYNTAX

unset [*varname* | *]

RELATED COMMAND

set

You may tell Z-Mail to delete all previously defined variables by entering:

unset *

Some variables are essential to the operation of Z-Mail and are restored to default values even after you explicitly **unset** them.

unsetenv

Unsets environment variables.

SYNTAX

unsetenv ***variable***

RELATED COMMANDS

setenv, set, unset

Removes the environment variable ***variable*** from the Z-Mail environment. See the **setenv** command on page 204 for more information.

update

The **update** command updates the specified folder.

SYNTAX

update [–N] [–n] [–r] [–x | –X] [%[*user*] | #[*n*] | & | *foldername*]

RELATED COMMANDS

quit, folder

The **update** command writes the changed contents of the specified folder to disk just as if you had quit, except that it does not change the New (N) status of messages to Unread (U). Messages that are marked for deletion are purged from the folder.

In the command-line interface, Z-Mail does not list headers when the folder is re-activated.

The **–r** option puts the folder into read-only mode after updating it.

See the **folder** command on page 145 for more information about the options to the **update** command.

version

The **version** command displays the version number of Z-Mail program you are running.

SYNTAX **version**

RELATED VARIABLE **version**

write

The **write** command saves the body text from messages to a file, omitting message headers.

SYNTAX **write [–f | –s | –S | –a | –A] [*messagelist*] [*filename* | *directory*]**

SYNONYMS **w**

RELATED COMMANDS **save, copy**

The **write** command functions identically to the **save** command except that it does not save message headers in the specified file called ***filename*** along with the body of text for each of the messages in ***messagelist***, nor does it set the status of messages to saved (S).

WARNING: The file that **write** creates cannot be used as a folder, since the messages are written without headers.

See the **save** command for details on using the **write** command.

CHAPTER 6

Z-Mail Variables

Most aspects of Z-Mail's behavior are controlled by the settings stored in Z-Mail *variables*. You can set and manipulate variables in any of Z-Mail's interfaces.

You can use the variables supplied automatically by Z-Mail as well as create variables of your own. Z-Mail provides variables that:

- tell Z-Mail how to work,
- define the way Z-Mail works with mail transport agents (MTAs) and other applications,
- tell Z-Mail where to look for data and setup files,
- store transient data about your Z-Mail session, and
- help you write Z-Script scripts and functions.

You can also define your own variables that you use in scripts and functions, or to store a value that you want to refer to later.

If you are a system administrator, you should familiarize yourself with Z-Mail's extensive list of variables. By setting Z-Mail's variables appropriately, you can configure Z-Mail to meet your exact needs and those of your organization.

See the sample scripts provided in the Z-Mail library (usually */usr/lib/Zmail*), including *sample.lib* and *sample.zmailrc*. These are heavily commented files that demonstrate Z-Script programming concepts and technique.

Types of Variables

Z-Mail supports and uses the following types of variables.

System Z-Mail provides many *system variables* whose settings you can modify to make Z-Mail function exactly the way you want. The initial settings of system variables are stored in Z-Mail initialization files and *$ZMLIB/variables*. See *Note for System Administrators* on page 226 for more information.

Internal Some of Z-Mail's system variables are read-only *internal variables* whose values can only be set by Z-Mail. If you try to change the values of these variables, Z-Mail will either warn you that you can't or will reset them to their internal values. Some internal variables have a special syntax when used in conditional expressions; see *Special Cases* in the description of the **if...else...endif** command on page 161 for details.

Function Z-Mail provides a set of *function variables* (or *positional parameters*) that you use in functions to reference the arguments passed to the function.

User-Defined You can create your own variables with the **set** command in scripts, functions, or at the command prompt. You can name variables almost anything you like, except the names of existing Z-Mail system variables. For details on defining your own variables, see the **set** command on page 202.

Types of Variable Values

The types of values you can assign to Z-Mail variables are:

boolean	Set or unset (i.e., *true* or *false*)
string	Set to a specific character string
integer	Set to a specific integer (numeric) value
multivalued	Set to one or more of a list of allowed values

Some Z-Mail system variables may be boolean, string, numeric, or multivalued at the same time.

Setting Variables

You can set variables, and save their settings to an initialization file, in any of Z-Mail's interfaces.

GUI and Lite

To set system variables in GUI and Lite modes, select the *Variables* item from the *Options* menu, and the *Variables* dialog box appears. Select a variable name in the scrolled list on the left side of the dialog box and Z-Mail displays a description of that variable. Use the buttons and fields displayed next to the scroll box to set the variable to the value you desire.

When you are finished setting variables, press the *Save* button to save their settings to a Z-Mail initialization file. By default, your Z-Mail configuration is saved in *~/.zmailrc*, which is normally read by Z-Mail during initialization at startup. See *Initializing Z-Mail* on page 63 for more information.

You may also use the procedures described below for the command-line interface to set variables in the GUI and Lite interfaces.

Command-Line

To set boolean variables to true, type **set** followed by the name of the variable. To set boolean variables to false, type **unset** followed by the variable name. You can **set** or **unset** more than one variable at a time.

```
set hold
unset nosave deletesave
```

To set the value of a numeric variable, type the name of the variable you wish to set followed by an equal sign, and then the new value for the variable. For example, to set the number of lines of a message Z-Mail displays when you issue the **top** command, enter:

```
set top_lines = 17
```

To set the value of a string variable, type the name of the variable followed by an equal sign and then the string. If the string contains blank spaces, it must be enclosed in quotes.

```
set editor = vi
```

```
set realname = "Mink Omelet"
```

To set the value of a multivalued variable, type the name of the variable followed by a quoted text string in which you separate the various values with spaces.

```
set quiet = "complete autosign newmail"
```

For more information about setting variables, see the **set** command on page 202.

To save variable settings to an initialization file, use the **saveopts** command. See the **saveopts** command on page 198 for more information.

Note for System Administrators

Z-Mail system variables take their defaults from three places, in order:

- The Z-Mail program itself
- The *$ZMLIB/variables* file, which is a specially-formatted file that contains the help text displayed in the *Variables* dialog, as well as descriptions of the variables themselves. See the beginning of the *variables* file for a description of its format.
- The system and user initialization files (*$ZMLIB/system.zmailrc* and *~/.zmailrc* by default).

You can alter the default system variable settings for your installation by editing either the *variables* file or *system.zmailrc*. Since the *variables* file is somewhat more complex and unforgiving of errors, it is generally easier to edit *system.zmailrc*.

If you wish to add new system variables to Z-Mail *and* have them appear in the *Variables* dialog for easy user modification, you must create a new, properly-formatted entry in the *variables* file. The *Variables* dialog only displays variables that are described in the *variables* file, not all the variables the Z-Mail is currently using.

For more information about modifying help files, see *Customizing the Help Files* on page 22.

Storing Variable Settings

Z-Mail stores variable settings in initialization files that it reads when you start Z-Mail.

Your system administrator may set variables to configure your organization's Z-Mail installation to work in a specific manner. The variable settings chosen by your system administrator are stored in a system initialization file; see *Note for System Administrators* on page 226 for a description of sources for Z-Mail system variable defaults.

You can override system variable defaults by specifying new values for Z-Mail variables and storing them in your personal initialization file with the **saveopts** command. By default, Z-Mail looks for a file called *.zmailrc* in your home directory and overrides your installation's system defaults with settings it finds in that file. See *Setting Variables* on page 225 for more information.

If you want to configure Z-Mail to work different ways for various groups, you can create several Z-Mail initialization files. For details on using initialization files to configure your Z-Mail installation, see the sections called *Initializing Z-Mail* beginning on page 63 and *Z-Mail Options Summary* beginning on page 36.

Using Variables

When writing scripts or typing commands directly into Z-Mail, you can expand a variable to its value by preceding the variable's name with a **$** character. For example, to set the value of **progname** equal to the value of **editor**, you could enter:

```
set progname = $editor
```

To create a new variable named **formfile** whose value is the pathname of a file called **formlist** in your home directory, you can enter:

```
set formfile = $home/formlist
```

Environment Variables

If you refer to a Z-Script variable that is not set, Z-Mail capitalizes the variable name and checks for an environment variable with that name. For example, if you use **$editor** in a script and the **editor** variable is undefined, Z-Mail looks for and uses the value of the UNIX environment variable *EDITOR*. The lower-to-upper case conversion is performed automatically; Z-Mail will not directly access UNIX environment variables whose names contain one or more lowercase characters.

Note that you cannot set environment variables with this mechanism, only read them. Use the *setenv* command (see page 204) to explicitly set environment variables.

Variable Modifiers

You can modify the expansion of a variable by putting a *variable modifier* at the end of the variable's name. Variable modifiers do not affect the value stored in the variable, only how it is expanded. You can only use one variable modifier per expansion. Valid variable modifiers are:

:t The variable's value is treated as a file path name, and returns the name of the file (the *tail* of the path) as the value of the variable. In other words, Z-Mail returns everything to the right of the last slash (**/**).

:h The variable's value is treated as a path name, and returns the *head* of the pathname as the value of the variable. In other words, Z-Mail returns everything up to, but not including, the last slash (**/**).

:l All alphabetic characters in the variable's value are converted to lower case.

:u All alphabetic characters in the variable's value are converted to upper case.

:num Returns word number ***num*** of the variable's value. Words are separated by spaces. If ***num*** is greater than the number of words in the variable, an empty string is returned.

Note: Z-Mail allows only one variable modifier for each **$** expansion. That is, an expression of the form

```
$realname:u:2
```

is not permitted. Instead, the expansion must be performed in two steps with an intermediate variable:

```
set temp=$realname:u
echo $temp:2
```

EXAMPLES The following table gives examples of how variable modifiers affect expansion of variable references.

Variable	*Value*	*Expression*	*Evaluates to*
home	/usr/dsmith	$home:t	dsmith
home	/usr/dsmith	$home:h	/usr
realname	Derek Taylor	$realname:l	derek taylor
realname	Derek Taylor	$realname:u	DEREK TAYLOR
realname	Derek Taylor	$realname:2	Taylor
realname	Derek Taylor	$realname:3	*empty string*

Message Information Variables

When Z-Mail encounters a **%** character inside square brackets which are preceded by a **$** character, it examines the character string that follows as a *message information variable*. The values of message variables depend on the characteristics of the current message.

SYNTAX **$[%*x*]** where ***%x*** is a valid message information variable

When Z-Mail encounters one of the *message information variables*, described below, inside square brackets, e.g. **$[%a]**, it makes the following substitutions:

%a	address of the author
%c	number of characters (bytes) in the message
%d	date and time of the message
%f	entire *From:* field (author)
%i	the message-id (may not be present)
%l	number of lines in the message
%m	month number from the message date
%M	month name from the message date
%n	name of the author
%N	day number of the month of the message
%p	priority of the message
%s	subject of the message
%t	*To:* field (recipients)
%T	time only of the message
%u	user name (login) of the author
%W	day of the week (Sun, Mon, etc.)
%y	last two digits of the year of the date
%Y	year of the message
%Z	time zone of the message
\n	a newline
\t	a tab

You may optionally specify a field width for each information string. For example, **%20f** displays the first 20 characters of the *From:* line. You may also use colon variable modifiers in conjunction with this bracket notation.

EXAMPLE

The following command saves the current message in a file whose name is the first eight characters of the name of the message's author:

```
save $[%8n]
```

The next command sets a user-defined variable called **_subj** equal to the first twenty letters of the subject of the current message, converted to upper case:

```
set _subj = "$[%20s]:u"
```

Note: Evaluating many message information variables in a single command is inefficient. To expand several variables, use the **–p** option in the **echo** or **eval** commands which also provides better quoting of the interpolated strings. For more details see the **echo** command on page 139 and the **eval** command on page 141.

Folder Information Variables

When Z-Mail encounters a **%** character inside parentheses which are preceded by a **$** character, it interprets the character string that follows as a *folder information variable*. The values of folder variables are taken from the characteristics of the current folder.

SYNTAX **$(%*x*)** where **%*x*** is a valid folder information variable

When Z-Mail encounters a one of the folder information variables, listed below, inside parentheses, it makes the following substitutions:

%d	number of deleted messages
%F	full pathname of the current folder
%f	name of the current folder (tail of **%F**)
%M	current month
%m	current message number
%N	current date (Number of the day in the month)
%n	number of new messages
%T	current time (in HH:MMxx format, xx is am or pm unless **mil_time** is set.)
%t	total number of messages
%u	number of unread messages
%W	weekday name (Sun, Mon, Tue)
%Y	current year (YYYY format)
%y	last two digits of %Y
\n	a newline
\t	a tab

You may specify a field width for each variable. For example, **%3M** displays the first three characters of the current month. You may also use colon variable modifiers in conjunction with this parentheses notation.

EXAMPLES The following command displays the current time:

```
echo $(%T)
```

The following command sets the user-defined variable **_folder** to the first eight characters of the current folder in upper case:

```
set _folder = "$(%8f):u"
```

Note: Evaluation of many folder information variables in a single command is inefficient. To expand several variables, use the **–h** option in the **echo** or **eval** commands which also provides better quoting of the interpolated strings. For more details see the **echo** command on page 139 and the **eval** command on page 141.

Testing the value of a multivalue variable

Certain variables called *multivalue variables* can be set to more than one value. Examples of such variables are the **compose_state**, **main_state**, and **mesage_state** variables.

To test whether a multivalue variable includes a specific value, use the following syntax:

$?*variable*:(*value*)

If ***variable*** contains ***value***, Z-Mail returns 1. If ***variable*** does not contain ***value***, Z-Mail returns 0.

For example:

$?compose_state:(autosign)

determines whether **compose_state** contains the field **autosign**, which indicates that the "automatic signature flag" is set. If the above expression returns "1," the autosign flag is set. If the above expression returns "0," the autosign flag is not set.

Variable Summary

This section summarizes Z-Mail's variables into functional categories. The categories appear in the following order:

Configuration Variables
- Help Variables
- Header Variables
- Display Variables
- Folder Variables
- Directory and File Variables
- Print Variables
- Editor Variables

Addressing and Sending Variables
- Compose Variables
- Address Book Variables (not available in Z-Mail Basic.)
- Addressing, Routing and Transport Variables
- Signature Variables

Interface Variables
- GUI Variables
- Lite Variables
- Command-Line Variables

Programming Variables

UNIX System Variables

For details on individual Z-Mail variables, see the *Variable Reference* beginning on page 248.

Help Variables

cmd_help Z-Mail looks for the file indicated by the **cmd_help** variable to find the general help file used by the Z-Mail **help** command.

gui_help Z-Mail looks for the file indicated by the **gui_help** variable to find the help file used by the graphical user interface versions of Z-Mail.

quiet The **quiet** variable tells Z-Mail to suppress warnings and user messages under specific circumstances.

verify The **verify** variable causes Z-Mail to request confirmation of certain actions.

version Z-Mail stores its current version number in the **version** variable.

warning If you set the **warning** variable, Z-Mail displays warning messages under a variety of situations.

Header Variables

alwaysignore Use the **alwaysignore** variable to tell Z-Mail to ignore specific message headers.

display_headers Use the **display_headers** variable to specify which headers should be shown when the **display** command is used to read a message.

hdrs_only Z-Mail sets the value of the **hdrs_only** variable to true if you specified the **–headers** option in the **zmail** command that started your Z-Mail session.

in_reply_to Set the **in_reply_to** variable to a string that completes the *In-Reply-To:* header.

no_expand If you set the **no_expand** variable, Z-Mail does not expand aliases, so message headers contain aliases exactly as you typed them.

no_hdrs If you set the **no_hdrs** variable, Z-Mail does not include your personalized mail headers in messages.

reply_to_hdr If you set the **reply_to_hdr** variable and provide a list of headers (delimited by spaces or commas), then Z-Mail searches that list for return paths.

summary_fmt The **summary_fmt** variable allows you to control the format of the header information Z-Mail displays for each message.

Display Variables

autoprint If you set the **autoprint** variable, after you delete a message, Z-Mail automatically displays the next message that has not been deleted or saved.

autotyper Specifies the name of an autotyper program. An autotyper program takes the name of a file being attached to a message and returns an appropriate MIME-type name.

date_received If you set the **date_received** variable, Z-Mail displays the date and time you receive messages rather than the default of when they were sent by the author.

filelist_fmt The **filelist_fmt** variable controls the format of the file name entries in the File Finder and Folder Manager dialogs.

hidden The **hidden** variable specifies which types of messages should not appear in the message summary list in the Main Window.

message_panes Set **message_panes** to specify the list of panes you wish Z-Mail to display in the Message Window.

mil_time Set **mil_time** to display time of day in 24-hour format (***hh***:***mm***) instead of 12-hour format (***hh***:***mm*** am | pm).

pop_timeout The **pop_timeout** variable specifies the number of minutes between checks for new mail on the POP server. If you aren't using POP, set the **timeout** variable instead.

show_deleted If you set the **show_deleted** variable, Z-Mail lets you display deleted messages in command-line mode using any command that displays a message.

squeeze If you set the **squeeze** variable, then whenever you **read**, **pipe**, or **save** messages, Z-Mail "squeezes" all consecutive blank lines into one blank line.

timeout The **ti meout** variable specifies the number of seconds between checks for new mail in your system mailbox.

toplines The **toplines** variable tells Z-Mail the number of message lines to display when you issue the **top** command in command-line mode.

wrap If you set the **wrap** variable, when you move beyond the last message in the active folder using the **next** or **delete** commands, Z-Mail moves the pointer to the first message in the folder that hasn't been deleted or saved.

wrapcolumn Set **wrapcolumn** to a column number where you want Z-Mail to automatically wrap lines as you compose messages in GUI and Lite modes. **autoformat** must also be set for **wrapcolumn** to take effect.

Folder Variables

always_sort The **always_sort** variable is similar to **sort**, but it causes your System Folder to be sorted whenever new mail arrives there, rather than when Z-Mail is started. If you set **always_sort**, you must also set **sort**, otherwise it is ignored.

deletesave Set the **deletesave** variable to cause Z-Mail to delete saved messages from folders when they are updated.

folder Set the **folder** variable to the path of a directory where you keep your mail folders.

folder_type Controls the format of new folders and messages that are appended to unopened folders.

hangup If you set the **hangup** variable, when Z-Mail receives a SIGHUP signal, it silently updates all open folders before exiting.

hold Set the **hold** variable to tell Z-Mail to keep any messages you have read in your system folder.

index_size Sets the threshold number of messages that must be exceeded by a folder before Z-Mail creates an index for the folder.

keepsave If you set the **keepsave** variable, the **save** and **write** commands do not mark messages for deletion.

mailbox_name The **mailbox_name** variable specifies a string to be used to describe the system mail folder. Whenever the name of the system mail folder is requested, the value of this variable is displayed.

mbox Set the **mbox** variable to the pathname of the file you want Z-Mail to use as your default folder for saving mail.

main_state Multivalue, read-only variable that contains information about the *current folder*. In line mode, the current folder is the active folder. In Motif, Lite, or Windows, the current folder is the current window's active folder.

message_state Multivalue, read-only variable that contains information about the active folder's current message. The current message is the message that's selected in the currently active folder. In line mode, the current folder is the active folder. In Motif, Lite, or Windows, the current folder is the current window's active folder.

pop_options If you are using POP, you may set the **pop_options** variable to ***preserve*** to have your mail left on the POP server. This is somewhat analogous to setting the **hold** variable, which has no effect in POP environments.

save_empty If you set the **save_empty** variable, it prevents Z-Mail from deleting empty folders.

sort Set the value of the **sort** variable to presort the messages in your system folder when you first start Z-Mail.

thisfolder Z-Mail sets the **thisfolder** variable to the full pathname of the active folder.

Directory and File Variables

attach_types The **attach_types** variable allows you to specify a file or files where you define encoding keys and file types for attachments.

cdpath Z-Mail uses the **cdpath** variable to search for a directory when you issue the **cd** command.

cwd Z-Mail sets the **cwd** variable to the name of the current working directory automatically.

dead Set the value of the **dead** variable to the name of the file to which you want Z-Mail to send interrupted (incomplete) messages.

detach_dir The directory where Z-Mail places temporary attachment files.

folder Set the **folder** variable to the path of a directory where you keep your mail folders.

home Use the **home** variable to define your home directory. Z-Mail initializes the **home** variable to the value of the *HOME* environment variable, but you can modify it any time during your Z-Mail session.

index_dir The directory where Z-Mail should look for and create external folder indices.

logfile Set the **logfile** variable to a filename in which Z-Mail logs the headers of outgoing messages.

mbox Set the **mbox** variable to the pathname of the file you want Z-Mail to use as your default folder for saving mail.

templates Set the **templates** variable to a list of directories, separated by spaces, where you store prepared message templates.

tmpdir Set the **tmpdir** variable to the pathname of the directory you want Z-Mail to use to write all Z-Mail temporary files.

Print Variables

printer Set **printer** to tell Z-Mail which printer to use as the default printer when you use the **lpr** command.

print_cmd Set **print_cmd** to a UNIX command which is to be used to send messages to the line printer.

Editor Variables

autoedit If you set **autoedit**, Z-Mail automatically starts the editor of your choice whenever you send or reply to mail.

autoformat If **autoformat** is set, lines in compositions are automatically wrapped at the right column in GUI and Lite modes.

edit_hdrs If you set **edit_hdrs**, you can use an editor to directly edit the headers of outgoing messages.

editor Set **editor** to the name of the text editor you want to use in the command-line interface.

speller Set **speller** to the name of a program that will be used as the spelling checker for compositions and other editable text.

visual Set **visual** to the name of the text editor you want to use in the GUI and Lite interfaces, if you have not set **wineditor** or the WINEDITOR environment variable. This editor gets invoked from within the program specified by the **window_shell** variable.

wineditor Set the **wineditor** variable to the text edit you want to use in the GUI and Lite interfaces. If the WINEDITOR environment variable is set, **wineditor** is set to its value by default.

Compose Variables

ask If you set the **ask** variable, Z-Mail prompts you for a subject for outgoing mail messages in command-line mode.

askcc If you set the **askcc** variable, Z-Mail prompts you for a list of carbon copy recipients when you are finished editing a message you are composing in command-line mode.

autoclear Set the **autoclear** variable to have the text area of the Compose Window erased automatically after sending a message. Note that this makes it difficult to use the Reuse button in the Compose Window.

autoinclude If you set the **autoinclude** variable, Z-Mail inserts a copy of the message to which you are replying into the body of all reply messages.

autosave_count When a composition is in progress, Z-Mail periodically saves it to a temporary file. The **autosave_count** variable specifies the number of keystrokes between saves.

comp_attach_label Controls information displayed below each attachment icon in the attachments area of a Compose Window. If this variable is not set, or if none of the other information can be found, the attachment type is displayed.

compose_lines Set the **compose_lines** variable to the number of lines to display in the Compose Window in the GUI interface.

compose_panes Set the **compose_panes** variable to specify which panes should appear in the Compose Window.

compose_state Multivalue, read-only variable that contains information about the current composition.

dot If you set the **dot** variable, you can type a period (.) on a line by itself at the end of a composed message to tell Z-Mail's command-line interface that you have finished writing the message.

escape If you set the **escape** variable, you can compose a mail message and place an escape character as the first character on a line.

indent_str If you set the **indent_str** variable, Z-Mail indents each line of old messages you include in outgoing messages, preceding each line with the text string you specify.

msg_attach_label Controls information displayed below each attachment icon in the attachments area of a Message Window. If this variable is not set, or if none of the other information can be found, the attachment type is displayed.

ispeller The **ispeller** variable specifies the name of the program to be used for interactive spell checking.

nosave If you set the **nosave** variable, Z-Mail does not store a backup copy of terminated messages.

post_indent_str If you set the **post_indent_str** variable and you include the body of a message into an outgoing message or reply, Z-Mail inserts a line of text after the last line of the included message.

pre_indent_str If you set the **pre_indent_str** variable and you include the body of a message into an outgoing message or reply, Z-Mail inserts a line of text before the first line of the included message.

record Set the **record** variable to the name of a file where you want Z-Mail to record all outgoing mail.

record_users The **record_users** variable allows you to record all outgoing messages you send to a specific set of users for which you have log folders.

textpart_charset This variable names the character set Z-Mail will use in the text body parts of outgoing messages.

Address Book Variables

Address Book variables are not available in Z-Mail Basic.

address_book If set, this variable contains the name of the address book program used by the Address Browser dialog.

address_cache The **address_cache** variable controls which address book lookups are remembered.

all **-** remembers results of all lookups.

unique **-** remembers only results of lookups that produce a single matching address.

address_check When **address_check** is set, the Address Book toggle (located on the Options menu in the Compose Window) will be on initially. Unless you explicitly turn the toggle off, all addresses will be passed through address book program for verification.

address_filter If you set the **address_filter** variable to a regular expression, Z-Mail will not use the address book program to verify addresses that match the expression.

address_sort The **address_sort** variable is used to specify an external program for sorting addresses as they appear in *To*, *Cc*: or *Bcc* headers. If this variable is set, the addresses are passed to the sorting program whenever the header is edited.

expire_cache The address cache will remain valid for the number of hours specified by **expire_cache**. If the cache is not accessed for the specified time period, all the cached addresses are forgotten.

(Cached address are always forgotten when you exit Z-mail, regardless of the state of this variable.)

lookup_max The **lookup_max** variable specifies the maximum number of alternatives the address book program should present when no exact matching address is found.

lookup_sep The **lookup_sep** variable specifies a separator that distinguishes actual addresses from descriptions of addresses in the output of address book programs. If set, this variable must contain a string.

Addressing, Routing and Transport Variables

alternates When you reply to messages, Z-Mail does not send a copy of the message to your login name at any of the hosts specified by the alternates variable.

alwaysexpand Set the **alwaysexpand** variable to cause mail address aliases to be expanded whenever they are used in an outgoing message. To override this action for a single message, set the **no_expand** variable once you have begun composing it.

auto_route If you set the **auto_route** variable, Z-Mail removes address redundancies from lists of recipients and checks for matches with **known_hosts** machines.

domain_route If you use the **domain_route** variable in combination with the **auto_route** variable, you can force Z-Mail to speed delivery of mail to certain addresses that contain fully-qualified domains.

known_hosts Z-Mail uses the **known_hosts** variable in conjunction with the **auto_route** variable.

metoo Set the **metoo** variable to tell Z-Mail to keep you on the list of recipients when you reply to a message.

no_expand If you set the **no_expand** variable, Z-Mail does not expand aliases, so message headers contain aliases exactly as you typed them.

picky_mta The **picky_mta** variable tunes the formatting and presence of certain headers to the requirements of your MTA (mail transport agent).

sendmail Set the **sendmail** variable to the program and arguments you want Z-Mail to execute when it delivers messages to the mail transport agent.

verbose If you set the **verbose** variable on, Z-Mail passes the verbose (*–v*) option to the mail transport agent when it sends messages.

Signature Variables

autosign The **autosign** variable allows you to append a signature to outgoing mail.

autosign2 Use the **autosign2** variable to specify an alternate signature to use in certain cases instead of the signature specified in the **autosign** variable.

fortune If you set the **fortune** variable, Z-Mail appends a random fortune to the end of all outgoing messages using the UNIX command *fortune* or another program you specify.

fortunates The **fortunates** variable specifies a group of addresses which are to receive fortunes.

presign Use the **presign** variable to have Z-Mail insert your signature into a composition as soon as it's initiated, rather than when it is sent.

GUI and Lite Variables

autodismiss When **autodismiss** is set, Z-Mail automatically removes dialog boxes from the screen once they have completed an operation.

autoiconify The **autoiconify** variable automatically iconifies the windows you specify after performing a specific operation.

baud The **baud** variable is set to the baud rate of your terminal. Z-Mail Lite uses it to optimize terminal output.

folder_title Use the **folder_title** variable to specify the how the title of each open folder is displayed in the folder panel area of the main window.

gui_help Z-Mail looks for the file indicated by the **gui_help** variable to find the help file used by the graphical user interface versions of Z-Mail.

intr_level Sets the number of messages or the file size that must be exceeded by an operation before the *Task Meter* appears.

lite_help Z-Mail looks for the file indicated by the **lite_help** variable to find the help file used by the Lite version of Z-Mail.

mail_icon If you set the **mail_icon** variable to a pathname identifying an icon pixmap, Z-Mail in GUI mode uses the pixmap as its desktop icon when you iconify the main Z-Mail window.

main_panes Controls which panes of the Main Window are visible.

main_folder_title The **main_folder_title** variable describes the information that is displayed for each folder in the *Folder Status* list of the GUI and Lite interfaces.

message_lines Set the **message_lines** variable to the number of lines you want Z-Mail to display in the GUI interface's Read Window.

newmail_icon If you set the **newmail_icon** variable to a pathname identifying an icon pixmap Z-Mail uses the pixmap as its desktop icon when new mail has arrived and you have closed (iconified) the main Z-Mail window.

no_menubars If you set **no_menubars** in Z-Mail Lite, menubars will be hidden from view.

summary_lines Set the **summary_lines** variable to the number of message summaries you want Z-Mail to concurrently display in the GUI interface.

window_shell Set the **window_shell** variable to the name of the program to run when you create shell processes in GUI mode, if the **winterm** variable is not set.

winterm Set the **winterm** variable to the name of the program to use when you create a shell process in GUI mode. This variable defaults to the setting of the WINTERM environment variable, if it exists. If this variable is not set, the value of **window_shell** is used.

Command-Line Variables

complete Set the **complete** variable to enable filename completion.

crt Set the **crt** variable to the number of lines on your display; this affects the operation of pagers.

fignore Set the **fignore** variable to have Z-Mail to ignore a list of filename extensions when it performs filename completion.

history Set the **history** variable to the number of commands you want the shell interface to remember.

ignore_bang If you set the **ignore_bang** variable, Z-Mail ignores exclamation point characters (**!**) as command history references.

nonobang If you set the **nonobang** variable, Z-Mail avoids expanding history references that do not match anything, rather than generating error messages.

pager When Z-Mail displays a message longer than the number of lines you specify with the **crt** variable, Z-Mail executes the program you specify with the **pager** variable.

prompt Set the **prompt** variable customize Z-Mail's command-line prompt.

screen Set the **screen** variable to the number of message summaries you want Z-Mail to concurrently display in the command-line.

Programming Variables

input The **input** variable holds the message list that Z-Mail passes to a function in a command pipeline.

i18n_mail_transport The current composition is passed through the command specified by **i18n_mail_transport** whenever Z-Mail sends a message via the mail transport agent (MTA).

i18n_read_mailbox New mail is passed throught the command specified by **i18n_read_mailbox** is run whenever Z-Mail reads the mail spool to process new mail.

i18n_write_mailbox Existing messages in your system mail folder are passed through the command specified by **i18n_write_mailbox** whenever Z-Mail writes messags to the mail spool.

(All **i18n** variables are intended for use in Asian-language environments where the encoding used for message storage differs from that used for delivery.)

is_lite Z-Mail sets the **is_lite** variable if Z-Mail is in Lite mode.

is_gui Z-Mail sets the **is_gui** variable if Z-Mail is in graphical user interface mode.

is_sending Z-Mail sets the **is_sending** variable if you issue the **zmail** command from the UNIX command line to send a message.

is_shell Z-Mail sets the variable **is_shell** when it has started the Z-Mail shell.

output Z-Mail sets the **output** variable to a message list representing the output of your last successful command.

realname Holds your real name. Z-Mail initializes **realname** from the value of the *NAME* environment variable when you start Z-Mail.

recursive If you set the **recursive** variable, Z-Mail allows user-defined functions to recursively call themselves.

redirect Z-Mail sets **redirect** to true if it believes input to Z-Mail is redirected.

status The **status** variable reports the success or failure status of your most recent Z-Mail command.

thisfolder Z-Mail sets **thisfolder** to the full pathname of the active folder (current mailbox).

$**** Z-Mail sets **$ to a character string containing all the arguments passed to the currently executing function.

***$#*** Z-Mail sets **$#** equal to the number of arguments passed to the currently executing function.

$0 Z-Mail sets **$0** equal to the name of the currently executing function.

$1, $2, ... Z-Mail sets each of **$1, $2, ... $n** equal to the individual arguments passed to the currently executing function.

UNIX System Variables

dot_lock Controls how Z-Mail performs file locking.

hostname Set the **hostname** variable to the name of your computer. Z-Mail uses the value contained in the **hostname** variable to compose a correct *From:* line if you use a mail transport agent that does not create this header automatically.

ignoreeof Set the **ignoreeof** variable so when you type **Ctrl-D** in the command-line interface, Z-Mail does not exit.

newline Set the **newline** variable to a string so that the string is executed as a command when you type a newline (carriage return).

unix If you set the **unix** variable, Z-Mail treats all commands that are not Z-Mail commands as UNIX commands.

user The user that started Z-Mail.

$$ Returns the UNIX process id (PID) of the Z-Mail process.

Variable Reference

The remainder of this section describes all pre-defined Z-Mail variables in alphabetical order.

address_book

STRING NOT AVAILABLE IN Z-MAIL BASIC

Use the **address_book** variable to specify a program to run to verify e-mail addresses in a directory or address book. In order to have addresses checked automatically, you must also set the **address_check** variable.

No address book program is provided by default, although three minimal examples are available in *$ZMAIL/bin*:

lookup.pw	checks that addresses match usernames in the */etc/passwd* file.
lookup.al	checks that addresses match aliases in the */usr/lib/aliases* file.
lookup.ad	checks that addresses match entries in the user's *~/.addresses* file.

The address book program is passed two arguments: the maximum number of possible alternatives to return if the address is not found, and the address to look up.

Note that in GUI mode, the Address Browser dialog passes a value of -1 as the maximum number of alternatives, to indicate that the address book should return all possible matches. You may set the **lookup_max** variable to change this.

Z-Mail interprets the exit status of the address book program as follows:

0	No problems, addresses that match on standard output.
1	Execution failed for some reason, error message on standard output or standard error.
2	More that the maximum number of addresses matched, error message on standard output or standard error.
3	Incorrect number of arguments.
4	No addresses matched, original address on standard output.
5	One known-correct matching address on standard output.

Exit status 5 means that Z-Mail should use the address. Exit status 0 means that Z-Mail should prompt to confirm that the match is correct, even if there was only one match. If Z-Mail cannot prompt, or on any status other than 0 or 5, the original address is used.

If the address book program does its own interactive prompting, it should always exit with status either 4 or 5. If matching more than the maximum number of possibilities is not an error, the program should print the best matches to standard output and exit with status 0.

address_cache

MULTIVALUE   NOT AVAILABLE IN Z-MAIL BASIC

If **address_cache** is set, the result of each address book lookup is remembered (or cached). If the same pattern is passed to the address book again, the remembered match is substituted automatically and you are not asked for confirmation. This variable only applies when addresses are checked automatically (via the **address_check** variable or the Address Book toggle). If you use the Address Browser (in GUI mode) while this variable is set, it will display the results of a lookup, as well as any match in the cache.

You can set the address_cache variable to one of the following:

all	Remembers results of all lookups.
unique	Remembers only results of lookups that produce a single matching address.

address_check

BOOLEAN / MULTIVALUE   NOT AVAILABLE IN Z-MAIL BASIC

When this variable is set, Z-Mail automatically attempts to verify each address by passing it through the program specified by the **address_book** variable.

In GUI and Lite interfaces, address checking may be toggled on or off in each Compose Window. The variable determines whether checking is on or off initially.

You can set the address_check variable to one or more of the following:

alias	(GUI and Lite only) Check each address as it is entered into the Aliases dialog.

compose	**Check each address as it is entered into a composition.**
send	Check all addresses when a composition is sent.

address_filter

STRING NOT AVAILABLE IN Z-MAIL BASIC

When the **address_filter** variable is set to a regular expression, Z-Mail compares any address that will be sent through the **address_book** program against the regular expression. If the address matches, Z-Mail accepts it without checking against the address book.

If the regular expression begins with an exclamation point (!), addresses that do *not* match the remainder of the expression are accepted. To include a literal exclamation point at the beginning of the expression, precede it by a backslash (\).)

This filter is not applied when searching for addresses in the Address Browser.

If this variable is set but has no string value, it has no effect.

address_sort

STRING

The **address_sort** variable specifies an external program to use for sorting E-Mail addresses as they appear in the *To*, *Cc*, or *Bcc* headers. If this variable is set, the addresses in each of those headers are passed through the sorting program whenever the header is edited.

The program should read addresses, one per line, from standard input and print the sorted list, again one per line, on standard output.

The program is run via the UNIX shell, so it may be a shell pipeline and may have command arguments specified.

alternates

STRING

When you reply to messages, Z-Mail does not send a copy of the message to your login name at any of the hosts listed on the alternates list. Hosts may be specified either as a single machine name or as a UUCP path, using “!” characters to separate machine names.

The symbol “*” in the alternates list matches your login name at any host. For instance, if you are bobg@zyrcon.com, then “*” means “do not reply to bobg@andrew.cmu.edu or uunet!bobg or anyone else whose ID is bobg”. If the “*” is followed immediately by some name, then that name is used in place of your login ID. If you have another login name on the local or remote machine, then that login may be specified either as

user@machine

or by preceding the login name or a UUCP path to the login name by a “!” character, as

!machine!user

The leading “!” character tells Z-Mail that the path ends in a login name. Without the “!”, Z-Mail will append your login name.

Note that the **alternates** command changes the value of this variable.

alwaysexpand

BOOLEAN

Setting **alwaysexpand** causes mail address aliases to be expanded whenever they are used in an outgoing message. This is overridden at composition and send times by the setting the **no_expand** variable (so **no_expand** may be changed after starting a composition but before sending the message).

alwaysexpand has no effect when defining new aliases via the **alias** Z-Script command or in the GUI and Lite interface Aliases dialogs.

alwaysignore

BOOLEAN / MULTIVALUE

You use the **alwaysignore** variable to completely stop Z-Mail from showing message headers you have specified in the **ignore** command.

You use Z-Mail’s **ignore** command to specify which headers you want Z-Mail to ignore when it displays messages. When you issue a **display**, **type**, or **print** command, Z-Mail does not display the selected headers. However, if you capitalize the first letter of the commands (i.e., **Display**, **Type**, **Print**), Z-Mail shows the message headers anyway.

Normally, Z-Mail also displays message headers you set with the **ignore** command when you:

- Send messages to the printer,
- Interpolate messages into letters with **~f** or **~I**,
- Use the **–I** option with the **mail** or **reply** commands.

You can set the value of **alwaysignore** to one or more of the following values, Z-Mail ignores headers as indicated:

forward	Ignore headers when forwarding messages (**~f**).
include	Ignore headers when including messages (**~I**).
pipe	The **pipe** command ignores headers.
printer	The **lpr** command ignores headers.

If you set **alwaysignore**, but specify none of the above arguments, it is equivalent to specifying all of the arguments.

Note: Z-Mail never ignores headers during updates or when you use the **save** command, since you might ignore headers required by Z-Mail or any other mail system to read folders.

For more information, see the **ignore** command on page 168.

always_sort

STRING

If you set the **always_sort** variable, your system mailbox will be sorted whenever new mail arrives. This variable is ignored if the **sort** variable is not set. If **sort** is set but **always_sort** is not, mail will only be sorted only when you first invoke Z-Mail.

Both the **sort** and **always_sort** variables use the syntax of the **sort** command. See the **sort** command on page 208.

ask

BOOLEAN

If you set the **ask** variable, Z-Mail prompts you for a subject for outgoing mail messages in command-line mode. Once you are in the message, use the tilde escape **~s** to set the subject or specify the **–s** option in the **mail** command at the Z-Mail command line.

asksub is a synonym for **ask**.

askcc

BOOLEAN

If you set the **askcc** variable, Z-Mail prompts you for a list of carbon copy recipients when you are finished composing a message in command-line mode.

If you also set the **edit_hdrs** variable, Z-Mail does not prompt you for a list of carbon copy recipients, but adds a *Cc:* line to outgoing messages which you may fill in using your text editor.

attach_types

STRING

The **attach_types** variable allows you to specify a file or files where you define encoding keys and file types for attachments. Z-Mail provides you with a default file containing some of these definitions. The files named by the **attach_types** variable may contain alternate or additional encodings and file types. If you specify multiple files in **attach_types**, they should be separated by commas or spaces and are read in the order listed.

You must use the **attach –rehash** command to force Z-Mail to reload the attachment type definitions if you either change the value of the **attach_types** variable or you modify the file(s) while Z-Mail is running.

By default, Z-Mail reads attachment information at startup from *$ZMLIB/attach.types*.

autoclear

BOOLEAN

If the **autoclear** variable is set, the text area of the Compose Window is erased after a composition is sent.

If the **autodismiss** variable is set, the **autoclear** variable is not useful, since the Compose Window will be closed as soon as a composition is sent. Also, you will be unable to reuse a composition (using the Reuse button) if **autoclear** is set.

autodismiss

BOOLEAN / MULTIVALUE   GUI and LITE ONLY

When you set **autodismiss**, Z-Mail automatically removes dialog boxes from the screen once they have completed an operation. You may set **autodismiss** to a list of keywords, separated by commas or spaces, which describe the dialog boxes you want to affect. **Autodismiss** can close the following dialog boxes:

Keyword	***Automatically closes:***
buttons	The Buttons dialog, after installing a button.
compose	The Compose window, after sending mail.
folder	The Find Folders dialog, after changing folders.
message	Message Window after message is deleted.
print	The Print Messages dialog, after printing.
save	The Save Messages dialog, after saving.
sort	The Custom Sort dialog, after sorting.
templates	The Templates dialog, after Use.

If you set **autodismiss** on, but you do not specify a list of dialog box names, Z-Mail closes any of the above dialogs listed above after you use them to perform an operation.

autodisplay

STRING   GUI ONLY

If **autodisplay** is set to a list of comma or space separated attachment type keywords, then attachments of those types are automatically displayed when the message containing them is read.

To bypass **autodisplay** for a particular message, enter (only) the number of the message on the command line.

autoedit

BOOLEAN / MULTIVALUE

If you set the **autoedit** variable, Z-Mail automatically starts the editor of your choice whenever you send or reply to mail. This variable can also be limited to particular circumstances by setting one of the following keywords:

new	Autoedit only when you compose a new message.
reply	Autoedit only when you compose a reply.

The **reply** argument is helpful when you use it in conjunction with the **autoinclude** variable.

In command-line mode, the **editor** variable is consulted for the name of the editor to start. In GUI and Lite modes, the **wineditor** variable is consulted.

autoformat

BOOLEAN GUI and LITE ONLY

When **autoformat** is set, or the *Autoformat* toggle of the *Options* menu is set, then Z-Mail will automatically wrap lines to the width of the message body area. This allows you to type text without having to type **Enter** at the end of every line. See also the *Z-Mail User Guide* for more information about autoformatting.

If **wrapcolumn** is set to a column number, then **autoformat** will wrap lines at that column instead of the default.

autoiconify

BOOLEAN / MULTIVALUE GUI and LITE ONLY

The **autoiconify** variable automatically iconifies any windows you select after performing a specific operation. In the Lite interface, an "iconified" window is sent to the back of the stack of windows.

If you set both **autodismiss** and **autoiconify** for a particular window, **autoiconify** takes precedence and Z-Mail iconifies the window. Setting **autoiconify** without any specific conditions is equivalent to setting all of the possible conditions. **Autoiconify** affects the following windows:

Keyword:	***Automatically iconifies:***
compose	The Compose window after sending mail.
message	The Message Display window on delete.

The **autoiconify** variable only affects Z-Mail's GUI and Lite interfaces.

autoinclude

BOOLEAN

If you set the **autoinclude** variable, Z-Mail inserts a copy of the message to which you are replying into the body of all reply messages. To make it easier for recipients to identify the included message, Z-Mail precedes the included text with the value of the **pre_indent_str** variable, follows the text with the value of **post_indent_str**, and indents the text using the string described by the **indent_str** variable.

autoprint

BOOLEAN

If you set the **autoprint** variable, Z-Mail automatically displays the next message after you delete a message in the Message Window. This is particularly helpful when you are using the GUI or Lite interface.

auto_route

BOOLEAN / STRING

If you set the **auto_route** variable, Z-Mail automatically generates the most efficient address it can for recipients of outgoing messages. When you set **auto_route**, Z-Mail:

- Shortens UUCP addresses whenever possible,
- Takes advantage of implied UUCP connections, and
- Takes advantage of hosts to which your organization has direct mail connections.

When you reply to a message, Z-Mail examines the *From:* and *Cc:* headers for information about return paths to the author and other recipients of the original message.

Note: The **–r** flag of the **reply** and **replyall** commands (see page 193) overrides the value of **auto_route**.

EXAMPLES

The following example explains how Z-Mail constructs return paths when you reply to a message.

How Z-Mail Constructs Return Paths

Assume you are Dana Parsons and your e-mail address is *dparsons@utech.com*. Josh Ford — whose e-mail address is *jford@chemcorp.com* — sends you a message and sends a copy of the message to Regina Tracy at *rtracy@gasco.com*. On the way to you and Regina, the message passes through several UUCP nodes called *blue*, *green* and *orange*. The exact path the message took to get to you is indicated below in the *From:* header of the example. The path the carbon copy took to get to Regina is in the *Cc:* header.

From: orange!green!chemcorp!jford (Josh Ford)
To: dparsons@utech.com (Dana Parsons)
Cc: blue!gasco!rtracy (Regina Tracy)

If you reply to Josh and send a copy of the reply to Regina, Z-Mail constructs the following return paths from the header information above:

From: dparsons@utech.com (Dana Parsons)
To: orange!green!chemcorp!jford (Josh Ford)
Cc: orange!green!blue!gasco!rtracy (Regina Tracy)

Z-Mail knows that since Josh sent mail to Regina via the host called *green*, that his machine can also route your reply via the same path.

Taking Advantage of Known Hosts

Continuing with the same example, assume that your machine, *utech*, has direct network connections to the hosts called *blue* and *black*. You therefore set the value of the **known_hosts** variable with the following command:

set known_hosts = "blue orange black"

If you set **auto_route**, Z-Mail takes advantage of these known hosts when constructing return paths for replies. Z-Mail scans the return path of each addressee of your reply, and discovers that the carbon copy to Regina is routed through the *blue* host. Since you have a direct connection to both *orange* and *blue*, Z-Mail selects the host farthest to the right (closest to Regina's host) and shortens Regina's address in the *Cc*: header to:

Cc: blue!gasco!rtracy (Regina Tracy)

For more information on specifying hosts to which you have direct connections, see the **known_hosts** variable on page 288.

Setting auto_route to a Pathname

If you set **auto_route** to a specific UUCP pathname (e.g., host names separated by **!** characters), Z-Mail prepends that pathname to all addresses in a list of recipients. This is useful for hosts which have a UUCP connection to a node that connects to the *Internet* or *USENET*.

Assume you are on the host named *bigcorp* and it has a direct UUCP connection with the host called *calif*. You can describe your path to the Internet by setting the value of **auto_route** with this command:

set auto_route = "bigcorp!calif"

If you reply to a message from Alan Roberts with a *From:* header such as:

From: west!cosoft!aroberts (Alan Roberts)

Z-Mail generates this address:

To: bigcorp!calif!west!cosoft!aroberts

Eliminating Redundant Hosts

After constructing return paths, Z-Mail eliminates duplicate hosts from each address. Suppose you have defined the value of auto_route with the following statement:

set auto_route = "bigcorp!calif"

If you receive a message from Alan Roberts with the following *From:* header:

From: west!bigcorp!cosoft!aroberts (Alan Roberts)

Z-Mail generates this address:

To: bigcorp!calif!west!bigcorp!cosoft!aroberts

And then, Z-Mail notices that *bigcorp* appears twice in the return path to Alan. There is no need to route the message through the hosts names *calif* and *west* since all they will do is send the message back to *bigcorp*. Therefore, before sending the message, Z-Mail eliminates the redundant hosts and shortens the return path:

To: bigcorp!cosoft!aroberts

autotyper

STRING

Set the **autotyper** variable to the name of an autotyper program. An autotyper program takes the name of a file being attached to a message and returns an appropriate MIME-type name.

Autotyper programs are not provided by default, but the following examples are available in *$ZMAIL/bin*:

type-by-magic	Checks that magic number of a file to determine its format.

type-by-suffix	Checks the suffix of a file's name to determine its format. This script does not look at a file's contents.

To use one of the programs above as your autotyper program, use the Variables dialog to set the **autotyper** variable to one of the program names.

Once the **autotyper** variable is set, you can invoke the autotyper program from the Attachments dialog . This is available on UNIX versions only.

autosave_count

INTEGER GUI ONLY

In GUI mode, compositions are periodically saved to temporary files. The **autosave_count** variable controls how many keystrokes occur between automatic saves.

autosign

BOOLEAN / STRING

The **autosign** variable allows you to append a signature to outgoing mail. Z-Mail does *not* add a signature or fortune to outgoing mail that is created by redirecting input into the **zmail** command.

If you set **autosign** on, but not to a string, Z-Mail appends the text contained in the *~/.signature* file to your outgoing messages. If you do assign a string to **autosign**, Z-Mail interprets it in one of three ways:

- By default, Z-Mail interprets the string as a pathname relative to the current directory. Leading special characters tilde (**~**) and plus (**+**) are expanded for you. If Z-Mail finds the file, it appends the file's contents to the message.
- If you set **autosign** to a string that begins with a backslash (\) or is surrounded by square braces ([...]), then Z-Mail uses the string itself as the signature (not including the \ or [ ] characters). Z-Mail performs no expansion and reads no additional files.
- If you set **autosign** to a string that begins with an exclamation point (**!**), Z-Mail interprets the rest of the string as a command. For compatibility with older products, you may also begin the string with a vertical bar or *pipe* char-

acter (|) instead of an exclamation point.
The addresses (*To*, *Cc*, and *Bcc* headers) of the message are passed to the command on its input, and the command's output is appended to the message. Z-Mail runs the command using the */bin/sh* shell, and does not expand file metacharacters (such as **~** and **+**) in the command string. Depending on the setting of **picky_mta**, the addresses passed to the command will be separated by either commas or spaces.

EXAMPLE

In the latter two cases, enclose the value of **autosign** in single quotes to prevent Z-Mail from interpreting the \ or | characters. For example:

set autosign = '\ Meenah Lavin, lavin@ugl.org'

will sign all outgoing mail with the string

Meenah Lavin, lavin@ugl.org

In the following example, Z-Mail will not append a signature, since its input is redirected:

% **zmail –s report michelle < report_file**

In this command, Z-Mail sends a message to the user *michelle* with a subject of *report*. Z-Mail redirects the contents of the file *report_file,* using the text as the body of the outgoing message. In this case, Z-Mail does not append a signature.

In the following example, the addresses of the message are passed to the program */u/mikem/scripts/buildsig*, the output of which Z-Mail will use as the signature.

set autosign = '!/u/mikem/scripts/buildsig'

autosign2

STRING

You may use the **autosign2** variable to specify an alternate signature to use instead of the signature specified in the **autosign** variable for some cases. You can use **autosign2** for special cases where:

- You do not want to use your default signature.
- You do not want a signature sent to special addresses.
- You want to use a certain signature for special addresses.

When you set the value of **autosign2**, you specify a list of addresses and/or address domains for which you want to use your alternate signature, as well as the signature itself. If you do not specify a signature, Z-Mail does not sign messages whose addresses match the addresses specified in the value of the **autosign2** variable.

SYNTAX The format for setting the value of the **autosign2** variable is:

set autosign2 = "*address1*, *address2*, . . . : *signature*"

The colon (**:**) separates the list of addresses from the optional ***signature***. If you don't specify a colon or omit the address list, Z-Mail treats the entire string as the signature, excluding the colon.

When you send a message, Z-Mail looks at the addresses of all recipients you specify. Z-Mail expands aliases, removes comments, then scans the list of recipients, matching the following patterns against those addresses (***address1***, ***address2***, etc.) you specified in the **autosign2** variable:

empty

If the value of **autosign2** doesn't contain a colon, or has no addresses to the left of the colon, then all addresses will match. This is useful for temporarily changing your signature without altering the setting of **autosign**.

address or address@domain
If you specify an **autosign2** address with no special characters (as defined below), Z-Mail compares each recipient of an outgoing message for an exact character match.

****username***
If you precede an e-mail address in the value of *autosign2* with a * character, Z-Mail performs an exact character match against the username of recipients, ignoring any sites that precede or domains that follow the user name. For example, in hill@hq2.com and alant!mayon!hill, only hill is examined when ***hill** is present in **autosign2**.

!host* or *!hostname!host1!host2...
If you specify a path of hostnames beginning with the **!** character, recipients who reside on the last host of the path match.

For example, in alant!mayon!hill, only mayon is examined if **!mayon** is present in **autosign2**.

@domain* or *@sub.domain
If you specify a domain name beginning with the @ character, all users residing in that domain match. For example, if **@sand** is present in **autosign2**, then hill@sand.com and fritz@cs.sand.edu will match.

If *all* the recipients you specify in your outgoing mail message match at least one of the address patterns present in **autosign2**, Z-Mail uses the **autosign2** signature. Otherwise, Z-Mail uses the signature you specify in value of the **autosign** variable.

Notes:

- You must set the **autosign** variable for the **autosign2** variable to work.
- If you invoke the **zmail** command using input redirection in the calling shell, Z-Mail does not append signatures.

EXAMPLES

The following table gives examples of how Z-Mail matches **autosign2** addresses against the addresses of recipients you specify in outgoing messages.

Autosign2 address	***Recipient:***	***Match?***
empty	***any***	Yes
rtaylor	**rtaylor**	Yes
rtaylor	**rtaylor@xyz.com**	No
***rtaylor**	**rtaylor**	Yes
***rtaylor**	**rtaylor@xyz.com**	Yes
!xyz	**xyz!rtaylor**	Yes
@xyz.com	**rtaylor**	No**
@xyz.com	**rtaylor@xyz.com**	Yes
@xyz	**rtaylor@xyz.com**	Yes

** Assuming the local host is not named *xyz.com*

EXAMPLE 1

Using **autosign** and **autosign2**, you can designate separate signatures for local (internal) and remote (external) mail messages.

```
set autosign = " ~/external.sig"
set autosign2 = "!research @zeno.com *ian tom: ~/internal.sig"
```

In this example, Z-Mail uses the alternate signature in the file *~/internal.sig* whenever you send mail to:

- Anyone on the UUCP host named *research*,
- Anyone within the *zeno.com* domain,
- User *ian* at any host anywhere—even locally, and
- User *tom* on your local machine only.

In all other cases, Z-Mail signs your outgoing messages with the signature in the *~/external.sig* file.

EXAMPLE 2

Similarly, you can designate no signature for internal messages by leaving the signature file off the value of **autosign2** :

```
set autosign = "~/external.sig"
set autosign2 = "!research @zeno.com *ian tom: "
```

EXAMPLE 3

If some of the recipients of your message match the **autosign2** address list, and some of the recipients do not, Z-Mail uses the value of **autosign** and not **autosign2** to sign your outgoing message. For example:

```
set autosign = "~/standard.sig"
set autosign2 = "michelle, shawna, johann: ~/alt.sig"
```

If you send a message to johann and shawna, Z-Mail signs the outgoing message with the contents of the *~/alt.sig* file. But if you send a message to johann, shawna and kim, Z-Mail uses the text in the *~/standard.sig* file since kim is not in the list of **autosign2** addresses.

baud

READ-ONLY INTEGER LITE ONLY

In Lite mode, the **baud** variable holds the baud rate of the connection that the system running Lite has with your terminal. If your terminal is not directly connected to the computer running Lite, then the **baud** variable may hold an erroneously high baud rate.

You can use the baud variable to adjust settings of Lite in your initialization file that depend upon the baud rate of your connection. For example, if you are connected at 2400 baud via modem to the computer running Lite, you may want to increase the threshold at which the Task Meter dialog appears:

```
if $?baud                  # ensure that baud is defined
    if $baud <= 2400       # and if baud is less than 2400
```

```
            set intr_level = 15          # then set intr_level to 15
        endif
    endif
```

cdpath

STRING

Z-Mail uses the **cdpath** variable to search for a directory when you issue the **cd** command (see page 129). You set the **cdpath** variable to a string of pathnames separated by spaces.

child

READ-ONLY

When Z-Mail creates a child UNIX process, this variable holds the child's process-id (PID).

cmd_help

STRING

The file indicated by the **cmd_help** variable is what Z-Mail uses as the general help file used by the **help** command. Set the value of **cmd_help** to the complete pathname of the help file for commands. Normally, only your system administrator sets this variable in Z-Mail's system initialization file, when the default location of the help file has changed. The default setting of **cmd_help** is *$ZMLIB*/command.hlp.

The value of **cmd_help** is not saved by the **saveopts** command.

colors_db

STRING *GUI ONLY*

The string in **colors_db** names the file that contains custom settings of colors for the GUI interface. By default, **colors_db** is set to ~/.zmcolors. You may set **colors_db** in your initialization file (which is usually *~/.zmailrc*) to read a different set of colors.

You may create as many custom color description files using the *Colors* dialog as you like.

comp_attach_label

STRING

Controls the information displayed below each attachment icon in the attachments area of a Compose Window. If this variable is not set, or if none of the variables described below are set, the attachment type is displayed.

comp_attach_label can be set to one of the following:

comment	Display the comment from the local attachment type database.
content	Display the content name supplied by the sender. If no content name was sent, this is the same as **filename**.
description	Display the descriptive text supplied by the sender. If no description was sent, this is the same as **comment**.
filename	Display the file name.
type	Display the attachment type.

complete

STRING *COMMAND-LINE and GUI ONLY*

Set the **complete** variable to enable *csh*-style filename completion. When you type the *completion character* at the end of a string of characters, Z-Mail interactively completes the string as a filename prefix relative to the current directory. If the prefix is not unique, Z-Mail completes the filename as far as possible and rings the terminal bell. You can tell Z-Mail not to ring the bell by setting the **quiet** variable (see page 303).

Z-Mail uses the first character of the value of the **complete** variable as the *completion character*. If you set the **complete** variable, but not to a value, the completion character defaults to **Escape**.

Z-Mail uses the second character of the value of the **complete** variable as the *listing character*. When you type this character, Z-Mail displays all possible completions and prompts you again with the original prefix. If you set **complete** to only a single character, completion listing is disabled; if you set **complete** but specify no string, the listing character defaults to **Ctrl-D**.

The listing character is not supported in the GUI interface.

If the first and second characters you specify in the **complete** variable are the same, Z-Mail completes unique matches as usual. However, partial matches produce a listing of the possible completions before completing as far as possible. See the **fignore** variable on page 275 for ways to exclude filenames from completion.

If the filename you are attempting to complete contains metacharacters, Z-Mail completes all possible matches. See the section *Specifying Filename Patterns* on page 27 for details on filename metacharacters.

EXAMPLES

This table demonstrates the use of **complete**:

Command to:	***Sets completion character to:***	***Sets completion listing character***
set complete	**Escape**	**Ctrl-D**
set complete = \E=	**Escape**	**=**
set complete = \E^L	**Escape**	**Ctrl-L**
set complete = ^F	**^F**	***disabled***

Notes

- Z-Mail cannot provide filename completion if you specify the **–echo** (**–E**) option in the **zmail** command.
- If you are running Z-Mail under a windowing system, the window manager may use **Ctrl** key sequences and the **Esc** key for windowing commands, so you may not be able to use **Ctrl+D** or **Esc**.
- In the command-line interface, if the list of possible completions generated by the *completion character* is longer than the command-line buffer (which is 256 characters long), Z-Mail discards all characters beyond the end of the buffer.
- If you want to silence the bell that Z-Mail sounds when it performs filename completion, set the **quiet** variable so it includes complete as one of its values.

compose_lines

INTEGER   GUI ONLY

Set the **compose_lines** variable to tell Z-Mail how many lines to initially display in the Compose Window in GUI mode. A setting of zero means that Z-Mail should use the default specified in Z-Mail's X Window System application defaults file.

compose_panes

BOOLEAN / MULTIVALUE   GUI and LITE ONLY

Each field in the **compose_panes** variable indicates the "panes" in the Compose window that are visible. By default, all of the panes are displayed.

This variable has no effect when it's unset. That is, the display is not changed in any way.

compose_panes may be set to one or more of the following:

folder	The area that contains the text field where the current folder is specified.
headers	The address headers.
body	The message body that is being edited.
action_area	The button panel where actions are displayed.
attachments	The attachments icons.

compose_state

READ-ONLY STRING

Multivalue variable that contains information about the current composition.

compose_state only has meaning in the context of a Compose Window (or, in line mode, if there is a current composition). If there is no active composition associated with the current window, it has no value.

In the context of a Compose Window, **compose_state** has the subvalue "active," as well as any number (or none) of these subvalues:

autosign	"Automatic signature" flag is set.
autoformat	"Automatic formatting" flag is set.
return_receipt	"Return receipt" flag is set.
edit_headers	"Edit headers" flag is set.

record_msg	"Record message to file" flag is set.
sort_addresses	"Sort addresses" flag is set.
directory_check	"Directory check" flag is set.
sendtime_check	"Directory check on send" flag is set.
log_msg	"Log message header to file" flag is set.
verbose	"Verbose" flag is set.
synchronous	"Synchronous send" flag is set.
record_user	"Record message to file based on user name" flag is set.
confirm_send	"Confirm addresses on send" flag is set.
pri_none	Message has no priority.
pri_*XXX*	This message has the ***XXX*** priority associated with it. ***XXX*** can be any string. The user can define priorities with the **priority** command.

If **active** is set, you can also set or clear any of these subvalues (except **active**). Setting **pri_*XXX*** clears **pri_none**, and setting **pri_none** clears any **pri_*XXX*** values that may be set.

You cannot assign values directly to **compose_state**; you can only set or clear subvalues using the commands:

set compose_state += *value*

and

set compose_state -= *value*

For more information on testing fields of multivalue variables, refer to the section "Testing the value of a multivalue variable" on page 232.

crt

INTEGER COMMAND-LINE ONLY

For the command-line interface, set the **crt** variable to the number of lines on your display. If the number of lines in a message exceeds the value of **crt**, Z-Mail invokes a pager so you can view the message in sections. By default, **crt** is set to 18.

cwd

STRING

Z-Mail automatically sets the **cwd** variable to the name of the current working directory. Z-Mail sets the value of **cwd** at start-up and resets it each time you use the **cd** or **pwd** command. You may change this variable like any other shell variable, but it does *not* cause the working directory to change.

The value of **cwd** is not saved by the **saveopts** command.

date_received

BOOLEAN

If you set the **date_received** variable, Z-Mail uses the date and time you receive messages rather than the default of when they were sent by the author for sorting and display.

When you sort messages by date, Z-Mail checks this variable to determine whether the messages should be sorted by date sent or date received.

dead

STRING

Set the value of the **dead** variable to the name of the folder to which you want Z-Mail to append interrupted or aborted messages. If you do not give **dead** a value, Z-Mail appends interrupted messages to a folder called *~/dead.letter*. You may set the **dead** variable to a UNIX command by prefixing the value with the pipe character (**|**).

For more information, see the **nosave** variable.

deletesave

BOOLEAN

If you set the **deletesave** variable, then when Z-Mail updates your active folder it deletes messages you have saved (and not explicitly preserved) since the folder's last update. Z-Mail does this by default for the system folder, but in other folders, Z-Mail does not normally delete saved messages.

Messages you have preserved with the **preserve** command are not affected by **deletesave**, and are not deleted by Z-Mail.

If you set both **keepsave** and **deletesave**, **keepsave** takes precedence and Z-Mail does not delete messages you have saved.

detach_dir

STRING

Z-Mail places temporary attachments and detached attachment files the directory specified in **detach_dir**. By default, **detach_dir** is +detach.dir. Temporary attachment files are deleted from the **detach_dir** directory when the folder containing the messages from which they were detached is closed or updated.

display_headers

BOOLEAN / MULTIVALUE

The **display_headers** variable specifies which headers will be displayed when the **display** command is used to read a message. It may be set to one of the following values.

all	Display all headers
none	Do not display any headers
retained	Display only headers that have been retained by the **retain** command
standard	If there are retained headers, display only those; otherwise display all headers that have not been ignored with the **ignore** command
unignored	Display only headers that have not been ignored with the **ignore** command

domain_route

BOOLEAN / STRING

If you use the **domain_route** variable in combination with the **auto_route** variable, you can force Z-Mail to speed delivery of mail to certain addresses that contain fully-qualified domains.

If your machine already knows how to deal with domains, set **domain_route** but do not specify a list of domains. Z-Mail then eliminates all intermediate hosts from outgoing addresses, but does not add routing information about your path to an Internet machine.

If your machine doesn't handle domains, set **domain_route** to a UUCP path from your machine to an Internet host or other machine that does understand domains.

Note: You can *only* apply domain short-circuiting to addresses containing a fully-qualified domain. Z-Mail short-circuits the route in spite of any path you specify with the **reply –r** option or through the string value of **auto_route**. As a result, Z-Mail is likely to bypass any **auto_route** or **–r** options you have set. For more information, see the **auto_route** variable on page 256.

EXAMPLE

Instead of routing mail via all the hosts specified in a routing path, you can tell Z-Mail to deliver directly its final domain via your Internet connection.

For example, suppose that your machine named *thinker* is not directly connected to the Internet. Instead, you must route Internet mail through a host named *research* which is connected to a machine named *wserver* which is on the Internet. You can set **domain_route** on *thinker* to automatically route responses to messages you receive:

set domain_route = research!wserver

Then, if you received mail with an address like:

uunet.UU.NET!west!victoria.vic3.com!sam

Z-Mail uses **domain_route** to shorten it to:

victoria.vic3.com!sam

and then prepends the value of **domain_route**, resulting in:

research!wserver!victoria.vic3.com!sam

This allows Z-Mail to send the mail through *research* and *wserver* and lets *wserver* decide the best route to *victoria*.

dot

BOOLEAN *COMMAND-LINE ONLY*

If you set the **dot** variable, you can type a period (**.**) on a line by itself at the end of a composed message to tell Z-Mail you have finished writing the message. Z-Mail continues to accept **Ctrl-D** as a valid way to perform the same action. In the GUI and Lite interfaces, the **dot** variable has no effect; instead, you either select the *Send* option from the *Deliver* menu, or press the *Send* button at the bottom of the *Compose* dialog box when you are finished composing messages.

dot_lock

MULTIVALUED / BOOLEAN

The **dot_lock** variable controls how Z-Mail performs *file locking*. Whenever Z-Mail writes to a file, it must *lock* it to ensure than no other program is trying to write to the file at the same time. There are several different ways that Z-Mail can perform file locking. Your system administrator should configure the
dot_lock variable to make Z-Mail use the correct method of file locking for your environment.

There are two basic types of file locking: kernel locking, and dot locking. A *kernel lock* is a file lock that is handled by the operating system, or kernel. A *dot lock* is a special file that is created in the same directory as the file that is to be locked, and that most programs agree to recognize as a lock. Many MTAs use dot locking.

dot_lock can be set to perform the following kinds of locking:

dot_lock Value	**Meaning**
unset	Perform kernel locking only, except on those platforms where the MTA is known to use dot locks. This is the default.
set *(but empty)*	Performs kernel and dot locking in the spool area*, and kernel locking elsewhere.
lock_only	Performs dot locking in the spool area*, and no locking elsewhere. *This can be dangerous to use without also setting* **lock_all.**
lock_all	Performs both dot locking and kernel locking everywhere, unless **lock_only** is also specified.
lock_only, **lock_all**	Dot locking is performed everywhere; kernel locking is not used at

all.

* The *spool area* is where mail is delivered and users' system folders reside; usually */usr/spool/mail* or */usr/mail* on most systems.

Setting **dot_lock** is recommended for systems that have an NFS-mounted spool area, since NFS kernel locking is generally unreliable.

edit_hdrs

BOOLEAN

If you set the **edit_hdrs** variable (or turn on the *Edit Headers* toggle in the *Options* menu of the GUI or Lite interfaces), you can use an external editor such as *vi* or *emacs* to edit the headers of outgoing messages. Z-Mail makes this possible by placing the headers of your outgoing message in the same editing buffer as the text of the letter. In the GUI and Lite interfaces, the headers are pulled into the message body editing area.

Once you are editing a message, changing **edit_hdrs** does not affect the message you are currently composing, although changing the *Edit Headers* toggle will affect the current message.

You should exercise caution when you edit the headers of an outgoing message, noting these rules:

- Z-Mail requires a *To:* header. Without this header, Z-Mail cannot send your message. Even if you specify a valid *Cc:* header, you must still include a *To:* header. You may have as many *To:* and *Cc:* headers as you like.
- You should not change the *From:* header. If you change this header to an address that Z-Mail is unable to identify as authentic, Z-Mail silently inserts your original *From:* header.
- Z-Mail always replaces the *Date:* header with a more accurate time and date stamp. Your changes to this header are ignored.
- You cannot add a *Status:* header; Z-Mail will remove it if you try to add one.
- Z-Mail drops blank headers. For example, if your message contains an empty *Cc:* header, Z-Mail removes the header

from the outgoing message.

- If you specify any *aliases* on the command line, Z-Mail expands them and places them into the message buffer in their expanded form, regardless of the value you set for the **no_expand** variable.
- If you set the **no_expand** variable and you use an external editor to edit the headers of a message, Z-Mail does not expand the aliases you enter in the message headers.
- Z-Mail always removes *Bcc:* headers, but still delivers messages to the addresses you specify.
- You can add an *Fcc:* header to specify a *File Carbon Copy*. This must be a full or partial pathname to a file or program. Partial pathnames are relative to the current working directory. For programs, you must precede the pathname with a pipe character (**|**). Z-Mail interprets all addresses on the *Fcc:* line that do not begin with a **|** character as filenames. Do not put any other types of addresses in an *Fcc:* header.
- If you include a newline in a header, it must be followed by a space or tab to indicate that the header continues on the next line.

editor

STRING COMMAND-LINE ONLY

Set the **editor** variable to the name of the text editor you want to use in command line mode with the **edit** command. If you set **editor**, but do not specify the name of an editor, Z-Mail uses the value of the *EDITOR* environment variable or the **visual** variable (see page 314).

escape

STRING COMMAND-LINE ONLY

If you set the **escape** variable, you can compose a mail message and place an escape character as the first character on a line. This causes Z-Mail to examine the next character and execute a function associated with that *escape* command. See *Tilde-Escape Editing Commands* beginning on page 91 for more information.

By default, **escape** is set to **~** (the tilde character).

expire_cache

BOOLEAN / NUMERIC   NOT AVAILABLE IN Z-MAIL BASIC

Set the **expire_cache** variable to the number of hours that the **address_cache** should remain valid. The cache is considered to be valid as long as it continues to be modified. If the cache is idle (no new entries are made and no old entries are removed) for more than the indicated number of hours, the entire cache is discarded (all cached addresses are forgotten).

If this variable is not set, or is set to zero (0), the cache never expires.

fetch_time_out

INTEGER

The setting of the **fetch_time_out** variable determines the frequency with which **fetch_mail_hook** function is executed, in minutes. For more information, see *Hook Functions* on page 157.

fignore

STRING   COMMAND-LINE ONLY

Set the **fignore** variable to a list of patterns to ignore when performing filename completion. You may set the value of **fignore** to a list of filename extensions (e.g., .c .o), a list of filename patterns containing metacharacters (* **? { } [ ]**), or a mixture of the two. You must separate each element in the list from the others with a space. When you ask Z-Mail to perform filename completion, it omits any filenames with the extensions or patterns specified in the value of **fignore** from the list of possible completions.

Notes:

- Z-Mail checks the value of **fignore** only when you enter the completion character, not the completion listing character. For more information, see the **complete** variable on page 265.
- If all files in the current directory match the extensions or patterns, Z-Mail disables **fignore** and performs completion. For this reason, it is usually not a good idea to include * in the list of values for the **fignore** variable, since this will effectively disable **fignore**.

EXAMPLE

The following command causes any filename ending in *.o* or *.s*, and any filename beginning with *Make* or *make*, to be excluded from completions:

```
set fignore = ".o .s [Mm]ake*"
```

filelist_fmt

STRING

This variable controls the format of the filename entries in the file finder dialog. The string may contain formatting characters to show information about each file.

%f	the filename
%t	the file's type, as in "[file]", "[empty]", etc.
%s	the file's size in bytes
%m	the month of the file's last-modified date
%d	the day of the file's last-modified date
%y	the year of the file's last-modified date
%T	the time of day of the file's last-modified date
%%	a literal %

The following conditions can be used for this construct:

f	the file is a plain file
d	the file is a directory
z	the file is zero size
.	the file is the directory ".."

Any of the substitutions may take a width specification in the form of a number just after the '%'. For example, "%10f" is a field ten characters wide which contains the filename. A string of this form:

```
%(f?plain file%)
```

is replaced with "plain file" if the file is a plain file, and is removed otherwise. Also, a string of the form:

```
%(f?plain file%|not plain file%)
```

is replaced with "plain file" if the file is a plain file, or the string "not plain file" otherwise.

A string of this form:

```
%n(text%)
```

(where ***n*** is a number) specifies that the "text" inside the parentheses should be filled or truncated to exactly ***n*** characters.

folder

STRING

The **folder** variable holds the path of the directory where you keep your mail folders. Z-Mail uses this path for various commands and to expand the *plus* metacharacter (**+**). By default, **folder** is *~/Mail*.

For more details, see the **folder** command on page 145 and *Specifying Filename Patterns* on page 27.

folder_title

STRING   GUI ONLY

The title of each open folder is displayed in the folder panel area of the main window according to the format specified in the **folder_title** variable. The string may contain the following formatting characters:

%F	full path of the active folder
%f	name (path tail) of the active folder
%m	current message number
%n	number of new messages
%u	number of unread messages
%d	number of deleted messages
%t	total number of messages
%T	current time
%N	day of the month (number) (today)
%W	weekday name (today)
%M	month name (this month)
%Y	year (this year)
%y	year (last 2 digits only)
%$	introduces a variable name
\n	newline
\t	tab

folder_type

STRING

The **folder_type** variable specifies the format in which new folders are created and messages are appended to unopened folders. There are two valid string values for **folder_type**:

standard	new folders are created in *V7* format
delimited	new folders are created in *MMDF* format

The default setting of **folder_type** is standard. If **folder_type** is set to any value other than standard or delimited, Z-Mail will default to *V7* format.

See *Mail Folders* beginning on page 17 for a description of *V7* and *MMDF* formats, and information about other supported folder types.

fonts_db

STRING GUI ONLY

The string in **fonts_db** names the file that contains custom settings of fonts and labels for the GUI interface. By default, **fonts_db** is ~/.zmfonts. You may set **fonts_db** in your initialization file (which is usually *~/.zmailrc*) to read a different set of fonts and labels.

You may create as many custom font and label description files using the *Fonts* dialog as you like.

fortune

BOOLEAN / STRING

If you set the **fortune** variable, Z-Mail appends a random fortune to the end of all outgoing messages using the UNIX command *fortune*. If you set the **fortune** variable to something that starts with a hyphen (e.g. **– o**), Z-Mail interprets it as an option to the *fortune* command.

By default, Z-Mail uses the command /usr/games/fortune –s which adds a short fortune to outgoing mail messages.

If the value of the **fortune** variable starts with a **/**, then Z-Mail executes the program described by the string instead of looking for the *fortune* program. File metacharacters in the **fortune** string are not expanded.

Note: The pathname for the *fortune* program can vary, so it may be different on your machine.

The optional **fortunates** variable is used to specify a list of people who are to receive fortunes.

fortunates

STRING

The **fortunates** variable can be set to a group of addresses who are to receive fortunes generated according to the **fortune** variable's setting. You set the value of **fortunates** to a list of addresses or aliases separated by spaces or commas, following the same rules as the address portion of the **autosign2** variable.

In an outgoing message, if the *To:* and *Cc:* lines contain only addresses specified in the value of **fortunates**, then Z-Mail appends a fortune to the message. If one or more addresses in the To: and Cc: lines contain addresses that are not in the **fortunates** list, then Z-Mail does not append a fortune.

Once you set **fortunates**, you cannot manually override its effect. In other words, you cannot use the **~F** tilde command to force a fortune to be added unless the individuals on the recipient list are all included in the **fortunates** list.

You must set the **fortune** variable in order for the **fortunates** variable to function.

EXAMPLE An alternate use of the **fortune** and **fortunate** variables is to provide extra information in your signature. For example:

```
set fortunates = @gamma.com
set fortune = '/bin/echo "Vice President, Betadata" '
```

Whenever you send mail to anyone at *gamma.com*, Z-Mail appends the string *Vice President, Betadata* to your messages.

gui_help

STRING GUI ONLY

Z-Mail looks for the file indicated by the **gui_help** variable to find the help file used by the Z-Mail's graphical user interface. Set the value of **gui_help** to the complete pathname of the GUI help file. Normally, only your system administrator resets this variable in Z-Mail's initialization files, when the default location of the **help** file has changed.

The value of **gui_help** is not saved by the **saveopts** command.

hangup

BOOLEAN

If you set the **hangup** variable, when Z-Mail receives a SIGHUP signal, it updates all open folders before exiting. This is useful if you frequently read mail via modem over a noisy phone line that often loses its connection. The *hangup* variable also forces an update when the X Window System server shuts down in Z-Mail's GUI mode.

WARNING: Z-Mail ignores certain errors during this update process because it presumes that it is impossible to ask you for instructions on how to recover from the errors. Z-Mail forces the updated folders to contain all messages that were not deleted at the time of the hangup. This means if an error occurs as Z-Mail loads new mail, the new mail is lost. On write errors, Z-Mail preserves both the working tempfile and as much of the active folder as possible.

hdrs_only

READ-ONLY STRING *SEND-ONLY MODE*

Z-Mail sets the value of the **hdrs_only** variable to the type of headers you requested to display if you specified the **–headers:*c*** option in the **zmail** command that started your Z-Mail session. You can test **hdrs_only** in initialization files to conditionally execute scripts. The type of headers you asked to display is represented by the single character ***c*** in the option **–headers:*c*** to the **zmail** command.

EXAMPLE

You can put the following commands at the beginning of initialization files to check if you are running Z-Mail only to display header summaries. In the following script, the **exit** command script stops reading the initialization file after you have set the **hdr_format** variable for displaying the headers. This reduces Z-Mail's startup time by bypassing unnecessary interface initialization.

```
set hdr_format = "%25f %7d (%l/%c) %25s"
if $hdrs_only == o # if we asked to see old messages' headers
    exit
endif
```

hidden

BOOLEAN / MULTIVALUE

This variable specifies which messages should be hidden from the headers list in the Main window. By default, only deleted messages are hidden, unless the **show_deleted** variable is set, in which case nothing is hidden.

If this variable is set, but does not have any values, then all types of messages are displayed.

hidden may be set to one or more of the following:

deleted	Hide deleted messages
forwarded	Hide forwarded messages
preserved	Hide preserved messages
replied	Hide messages that have been replied to
new	Hide new messages
saved	Hide saved messages
old	Hide old messages
unread	Hide unread messages

history

INTEGER COMMAND-LINE ONLY

Set the **history** variable to the number of commands you want the command-line interface to remember. This variable functions like the *history* variable used in *csh*. If **history** is set to zero, or is unset, Z-Mail remembers only the last command. For more information, see the **history** command on page 160.

hold

BOOLEAN

If **hold** is not set when you update your system folder, Z-Mail saves any messages you have read, except those marked as *preserved*, to your main mailbox (defined by **mbox**). The **hold** variable only affects the behavior of the system folder.

home

STRING

Use the **home** variable to define your home directory. Z-Mail initializes the **home** variable to the value of the *HOME* environment variable when Z-Mail starts.

Z-Mail stores temporary files for editing and looks for initialization files in the directory named by **home**. The **~** metacharacter is substituted with the value of **home** if it appears at the beginning of a filename. If Z-Mail cannot find or read your home directory, the */tmp* directory is used instead.

If you unset **home**, it is reset to your home directory.

The value of **home** is not saved by the **saveopts** command.

hostname

STRING

The **hostname** variable is set by default to the name of your computer. Z-Mail uses the value of the **hostname** variable to compose a correct *From:* line if you use a mail transport agent that does not create this header automatically. This feature aids recipients of your mail when they reply to your messages. Z-Mail also uses the **hostname** variable when it performs auto-routing.

Note: You normally do not have to set **hostname** since it should be set automatically by your system. However, some systems' hostnames cannot be queried and you may have to set this variable manually. In most cases, if this is an issue, you system manager will have already set your **hostname** in Z-Mail's system initialization file *system.zmailrc*.

The value of **hostname** is not saved by the **saveopts** command.

i18n_read_mailbox

STRING

The command specified in the **i18n_read_mailbox** variable is run whenever Z-Mail reads the mail spool to process new mail. New messages which have arrived since the last check are supplied to the command's standard input, and Z-Mail expects the message to appear on the command's standard output.

The command specified by this variable must not modify the headers of messages.

i18n_write_mailbox

STRING

The command specified by the **i18n_write_mailbox** variable is run whenever Z-Mail writes messages to the mail spool. Messages that are intended for writing to the spool are supplied to the command's standard input. Z-Mail expects the messags to appear on the command's standard output. The modified messages are then written to the mail spool.

The command specified by **i18n_write_mailbox** must not affect the headers of messages.

i18n_mail_transport

STRING

The command specified by the **i18n_mail_transport** variable is run whenever Z-Mail sends a message via the mail transport agent (MTA). The message is supplied on the command's standard input, and the command's standard output is passed to the MTA.

The command specified by **i18n_mail_transport** must not affect the headers of messages.

ignore_bang

BOOLEAN *COMMAND-LINE ONLY*

If you set the **ignore_bang** variable, Z-Mail ignores exclamation point characters (**!**) as command history references. Note that setting **ignore_bang** severely limits use of the **cmd** command, which depends upon history references for argument substitutions.

ignoreeof

BOOLEAN / STRING *COMMAND-LINE ONLY*

When the **ignoreeof** variable is set, Z-Mail does not exit if the end-of-file character (usually **Ctrl-D**) is typed at the beginning of a line. If you set **ignoreeof** to a string, Z-Mail executes that string as a command when you type the end-of-file character at the beginning of a command line. This does not effect termination of messages under the **mail** and **reply** commands.

EXAMPLE

```
set ignoreeof = "echo Please type quit to exit Z-Mail."
```

indent_str

STRING

If you set the **indent_str** variable, Z-Mail indents each line of messages you include in outgoing messages, preceding each line with the text string you specify.

The value of **indent_str** may contain the formatting characters described in *Message Information Variables* on page 229. These characters are expanded from the headers of the message that is being indented. If you unset **indent_str**, Z-Mail indents included messages with **>** and a space. For more information on identifying messages you include in outgoing mail, see the **pre_indent_str** and **post_indent_str** variables.

EXAMPLE

If **indent_str** is defined with the command:

set indent_str = ">>>"

then if you include a message in an outgoing message, Z-Mail indents it like this:

From: **dsmith@bigcorp**
To: **rwillet**
Subject: **Directions to offsite meeting**

Here's the directions to the meeting:
>>>From the airport:
>>>Take route 71 south for 5 miles.
>>>Exit at Rolling Road. Go west for 1.3 miles.
>>>Turn left at Moore Avenue. Go 3 blocks.
>>>The Prescott Hotel is on your right.

index_dir

STRING

The directory name in **index_dir** points to the directory where external indices are to be kept. For example, the **update –external** command will create an external index for the active folder in the directory specified by **index_dir**. By default, **index_dir** is set to +index.dir. If **index_dir** is not set, or is set to an empty string, then the default is used.

The file metacharacters ~ and + may be used in the value of **index_dir**.

index_size

INTEGER

If a folder with more messages than ten times the value of **index_size** is updated, then an index will be created for the folder. If the folder is read-only, then an external index is created, otherwise an internal index is created. If **index_size** is set to true or zero, then an index is always created for folders at update. The default **index_size** is 5, so folders with more than 50 messages will have indices created for them.

Opening and closing a folder without changing it does not create an index. However, if the index is out of date, Z-Mail will update it.

input

READ-ONLY STRING

The **input** variable holds the list of messages that Z-Mail passed to the current function via a pipeline. This variable is most useful in command-line mode and in scripts. You can use **input** within a function to test if the function is being used in a pipeline, or to test for the presence of a specific set of messages.

If Z-Mail pipes no messages to a function, **input** is undefined by default. However, if you set **input** to a value, that value will be the definition of **input** thereafter in functions that are not piped into.

EXAMPLE

```
function tellinput() {
     if $?input
         echo The messages piped to this function are:
     $input
    else
         echo This function is not in a pipeline.
    endif
}
```

Note that in the above example, it is assumed that **input** was not previously defined, and so the **$?input** test is reliable.

For more information on piping message lists, see the **output** variable on page 298.

in_reply_to

STRING

Set the **in_reply_to** variable to a string that completes the header *In-Reply-To:*. The metacharacters that you may place in this string are described in *Message Information Variables* on page 229. Z-Mail will prepend a space character to the string generated by **in_reply_to** to ensure that the resulting header is properly formatted. Note that if you embed a newline (**\n**) character in **in_reply_to**, it must be immediately followed by a space or tab (**\t**) to maintain valid header syntax.

EXAMPLE

If you respond to a message from Carol Scott that was sent on August 21, 1991, at 10:39pm, with **in_reply_to** set to the string:

%n's message of %d.

Z-Mail adds this header line to your message:

In-Reply-To: Carol Scott's message of Aug 21, 1991, 10:39pm.

intr_level

INTEGER GUI and LITE ONLY

In GUI and Lite modes, the *Task Meter* dialog appears during lengthy operations to tell the user the progress of the operation. **intr_level** sets the threshold at which the Task Meter appears.

For operations on messages (such as updating a folder), if the number of messages exceeds the setting of **intr_level** multiplied by 10, then the Task Meter will appear.

For operations on files (such as paging a file), then the Task Meter will appear if the size in bytes of the file is larger than the setting of **intr_level** multiplied by 1000.

For example, if **intr_level** is set to 5 (the default), then the Task Meter will appear during operations on more than 50 messages or files larger then 5000 bytes.

If **intr_level** is unset, then the Task Meter will never appear.

is_gui

READ-ONLY BOOLEAN

The **is_gui** variable is true if Z-Mail is in GUI mode, even though the interface may not yet be started fully. You can test the value of **is_gui** in initialization files to decide what variables to set or what buttons to install to properly configure your Z-Mail session.

Note that **is_gui** is a special variable that does not require a **$** in conditional expressions.

EXAMPLE

For example:

```
if  is_gui                              # note: no '$'
        button  Phone  phone_msg
        button  "Purch Req"  purch_req
        button  Room  reserve_room
endif
```

This **if** statement checks if you are running in GUI mode and if so, it installs three buttons.

This variable is read-only. It is particularly useful in initialization files as well as in user-defined functions that work differently in each of Z-Mail's interfaces.

is_lite

READ-ONLY BOOLEAN

The **is_lite** variable is true if Z-Mail is in Lite mode, even though the interface may not yet be started fully.

See **is_gui**, above, for an example that is also appropriate for **is_lite**. You may conditionally execute scripts that are appropriate for both Lite and GUI modes by setting and then later testing the value of the new variable **is_gui–or–lite**, which is defined by this example script:

```
unset  is_gui–or–lite
if  is_gui
        set  is_gui–or–lite
endif
if is_lite
        set  is_gui–or–lite
end if
```

is_sending

READ-ONLY BOOLEAN

The **is_sending** variable is true if you issue the **zmail** command from the UNIX command line to send a message. You can test the value of **is_sending** in scripts to take actions that are appropriate for sending mode. You can also skip setup procedures since you know that Z-Mail is only being used to send a message and that the Z-Mail interface will not be started.

This variable is read-only.

is_shell

READ-ONLY BOOLEAN

The variable **is_shell** is true when Z-Mail's interface (command-line, GUI, or Lite) is completely initialized. The value of **is_shell** is always false at startup while Z-Mail is reading initialization files. It is always true after startup. Therefore, note that **is_shell** is always true when you load initialization files manually using the **source** command.

This variable is read-only. It is particularly useful in initialization files to determine if Z-Mail is ready to perform interactive commands.

This example script only asks a question if the interface is started:

```
if  is_shell
      ask  "Open to–do folder?"  # ask is an interactive command.
      if  status == 0            # if the answer was yes,
          open  +to–do           # then open the +to–do folder
      endif
endif
```

ispeller

STRING *LITE ONLY*

The **ispeller** variable names a program to be used for interactively spell checking in compositions and other editable texts. This program should expect a file name as its sole argument.

keepsave

BOOLEAN

If you set the **keepsave** variable, then the **save** and **write** commands do not automatically mark messages for deletion. Z-Mail checks the value of **keepsave** when the folder is updated, not when the **save** or **write** is performed.

Also see the **hold** and **deletesave** variables.

known_hosts

STRING

Z-Mail uses the **known_hosts** variable in conjunction with the **auto_route** variable. Your system administrator usually sets this variable. **Known_hosts** contains a list of hosts with which your machine has direct UUCP connections; Z-Mail will shorten addresses of reply messages if they contain one of these hosts. The host names in the list are separated by commas or spaces.

When you reply to mail, the return path that Z-Mail constructs often has a number of hostnames through which it could route your reply. Instead, Z-Mail looks for hosts it knows it can contact directly and routes the messages through those machines.

EXAMPLE If you respond to mail that contains the path:

unicam!pixyr!root

but you know your machine already calls *pixyr*, then it is unnecessary to send the mail through *unicam* first. If you set your **known_hosts** string to include *pixyr*, Z-Mail routes the message directly to *pixyr* automatically:

pixyr!root

See *Specifying E-Mail Addresses* beginning on page 29 for general information about addressing messages.

logfile

STRING

Set the **logfile** variable to a filename in which Z-Mail logs the headers of outgoing messages. If you set the **logfile** variable on, but specify no filename, Z-Mail uses the *~/mail.log* file as the default log file.

To log entire messages including their body text, see the **record** variable. If you set both the **logfile** and the **record** variables, then the log file and the record files match closely. You can then use the log file to quickly scan for specific messages and then check the record file for its full text. Note, however, that opening an indexed record folder is often faster than opening an unindexed logfile, even through the record file is much larger.

lookup_max

NUMERIC NOT AVAILABLE IN Z-MAIL BASIC

The **lookup_max** variable specifies the maximum number of alternatives that the **address_book** program should present when no matching address is found. It is provided as the first argument to the **address_book** program.

lookup_sep

STRING NOT AVAILABLE IN Z-MAIL BASIC

When the output of the **address_book** program is examined by Z-Mail, each line is searched for the first occurrence of the string value of this variable. Everything to the left of the string specified by **lookup_sep** is presented to the user as a description of the address that was returned. Everything to the right is taken to be the actual address(es), which are returned as the value of the user's selection.

If this variable is set but has no string value, it has no effect.

mail_icon

STRING GUI ONLY

If you set the **mail_icon** variable to a pathname identifying an X Window System icon pixmap, Z-Mail uses the pixmap as its desktop icon when you close (iconify) the main Z-Mail window.

Once you have set and saved **mail_icon**, the next time you start Z-Mail it will use the new icon.

mailbox_name

STRING

The **mailbox_name** variable specifies the string used to describe the system folder. Whenever the name of the system mail folder is displayed,this string will be used instead. If **mailbox_name** is set to **%n**, the user's login name will be displayed.

main_folder_title

STRING GUI and LITE ONLY

The **main_folder_title** variable contains a string that is used as the folder label for each folder in the *Folder Status* list of the

Main Window. You may use folder information strings, as described in *Folder Information Variables* on page 231.

For example, if you set **title** as follows:

set title = "%f\: %n new of %t"

then the Folder Status list will contain folder information similar to:

```
[1]   doc: 3 new of 103
[2]–  temp: 12 new of 17
[3]+  outgoing: 0 new of 629
```

main_panes

MULTIVALUED   GUI and LITE ONLY

Each value of **main_panes** corresponds to a pane of the Main Window in GUI and Lite modes. By default, all of the possible values of **main_panes** are set:

status	the status of the active folder
folder	the name of the active folder
messages	the message summary list
buttons	the button panel
output	the output area
command	the command entry area (not available in Z-MAIL BASIC.)

In the GUI and Lite interfaces, you may also choose which panes are displayed by setting the toggles in the *Panes* menu or the Main Window.

main_state

READ-ONLY STRING

main_state is a multivalue variable which gives information about the *current folder*. In line mode, the current folder is the currently open folder. In motif, lite, or windows, the current folder is the current window's current folder.

For example, if you add a button to the main, message, and compose windows that executes the command:

echo $main_state

When you press your newly created button in the main or compose windows, it tells you information about the main window's currently open folder. If you press the button in the message reading window, Z-Mail tells you about the folder from which the message originated

The subvalues of main_state are as follows:

has_folder	"True" if there is an open current folder.
folder_writable	"True" if the current folder is not open readonly.
folder_has_messages	"True" if the current folder has one or more messages.
folder_needs_update	"True if the current folder needs to be updated before it's closed.

You can't set **main_state**; you can only look at its value.

For more information on testing fields of multivalue variables, refer to the section "Testing the value of a multivalue variable" on page 232.

mbox

STRING

Set the **mbox** variable to the pathname of the file you want Z-Mail to use as your default folder for saving mail (your *main mailbox*). If you do not set the **mbox** variable, Z-Mail uses *~/mbox*.

message_lines

INTEGER *GUI ONLY*

If you are using the GUI interface, set the **message_lines** variable to the number of lines you want Z-Mail to display in the Read Window. If a message exceeds that number of lines, use the scroll bar on the side of the window to move the text so you can read the entire message.

message_panes

BOOLEAN / MULTIVALUE

Each field in the **message_panes** variable indicates the panes in the Message window that are visible. By default, all of the panes are displayed.

This variable has no effect when it's unset. That is, the display is not changed in any way.

message_panes may be set to any of the following:

folder	The area that contains the text field where the current folder is specified.
headers	The summary of the current message headers.
body	The message body that is being displayed.
action_area	The button panel where actions are displayed.
attachments	The attachments icons.

message_state

READ-ONLY STRING

Multivalue variable that contains information about the current folder's current message.

For example, if you add a button to the main, message, and compose windows that executes:

echo $message_state

when you press the newly created button in the Main or Compose Windows, the command line displays information about the currently selected message (or about the first selected message, if there is more than one). Pressing this button in the Message Window displays information about the currently displayed message.

Possible subvalues for **message_state** are:

is_next - true	If this message has a "next" message. The keepsave, deletesave, wrap, and hidden variables affect this.
is_prev - true	If thecurrent message has a "previous" message. The *keepsave*, *deletesave*, *wrap*, and *hidden* variables affect this.
pinup - true	If the current message window is a pinup window.
attachments	"True" if this message has attachments.

For more information on testing fields of multivalue variables, refer to the section "Testing the value of a multivalue variable" on page 232.

metoo

BOOLEAN

Set the **metoo** variable to tell Z-Mail to keep you on the list of recipients when you reply to a message. By default, Z-Mail removes your address from the list of recipients. See the **alternates** command on page 119 for information on how Z-Mail determines whether you are on the list of recipients.

mil_time

BOOLEAN

Whenever Z-Mail displays time in a message header or in the command-line prompt, it can display it in either 12-hour am/pm or 24 hour format. Set the **mil_time** variable to tell Z-Mail to display the time in 24-hour (or "military") format. Unset
mil_time to display time in 12-hour format. **mil_time** is unset by default.

msg_attach_label

STRING

Controls information displayed below each attachment icon in the attachments area of a MessageWindow. If this variable is not set, or if none of the variables below are set, the attachment type is displayed.

msg_attach_label can have one of the following values:

comment	Display the comment from the local attachment type database.
content	Display the content name supplied by the sender. If no content name was sent, this is the same as **filename**.
description	Display the descriptive text supplied by the sender. If no description was sent, this is the same as **comment**.
filename	Display the file name.
type	Display the attachment type.

msg_win_hdr_fmt

STRING GUI ONLY

The **msg_win_hdr_fmt** variable controls the format of the text in the header window. The following formatting characters may be used:

%a	address of the author
%c	number of characters (bytes) in the message
%d	date and time of the message
%f	"From" field (author name and address)
%i	the message-id (sometimes not available)
%l	number of lines in the message
%M	month name from the message date
%m	month number from the message date
%N	day of the month (number)
%n	name of the author
%s	subject of the message
%t	"To" field (recipients of the message)
%T	time of the message (see **mil_time** variable)
%u	user name (login) of the sender of the message
%W	day of the week (Sun, Mon, etc.) of the message date
%Y	year of the message date
%y	year, last 2 digits only
\n	a newline
\t	a tab

newline

STRING COMMAND-LINE ONLY

Set the **newline** variable to a string that Z-Mail executes as a command when you type a newline (**Enter**) in command-line mode. The default behavior of Z-Mail is to give you a prompt whenever you type a newline.

For example, other MUAs act as if you typed a *next* command when you type a newline. If you are comfortable with an older MUA, you may wish to tell Z-Mail to act the same way. You do this by setting the **newline** variable accordingly:

```
set newline = next
```

newmail_icon

STRING GUI ONLY

If you set the **newmail_icon** variable to a pathname identifying an X Window System icon pixmap, Z-Mail uses the pixmap as its desktop icon when new mail has arrived and you have iconified the main Z-Mail window.

Once you have set and saved **newmail_icon**, the next time you start Z-Mail, it will use the new icon when new mail is waiting.

no_expand

BOOLEAN

If you set the **no_expand** variable, Z-Mail does not expand aliases, so message headers contain aliases exactly as you typed them. By default, when you use a Z-Mail alias to refer a list of addresses, Z-Mail expands the list on the *To:* and *Cc:* lines to indicate the complete list of all the recipients of the message.

EXAMPLE

If you have defined an alias named *all* to contain the addresses of all 250 people in your organization, you may not want Z-Mail to expand the alias so the *To:* header contains 250 addresses. If you set **no_expand** before sending the message, Z-Mail sends the following header on the message:

To: all

WARNING: The **no_expand** variable is intended as a tool for managing mailing lists, not for sending everyday person-to-person messages. When you set **no_expand**, the recipients of your outgoing messages receive the alias names you enter, just as you type them. If the recipients of your messages do not have access to the same alias definitions that you do, they are unable to construct return paths to reply to the people on the mailing list. Therefore, before you set **no_expand**, make sure that either you don't want people to be able to reply to others in your alias lists, or that everyone on the alias lists has access to their definitions.

no_hdrs

BOOLEAN

If you set the **no_hdrs** variable, Z-Mail does not include your personalized mail headers (defined with **my_hdr**) in messages. This does not unset your personalized headers, it just disables them.

no_menubars

BOOLEAN LITE ONLY

If **no_menubars** is turned on, menubars will be hidden. Changing this variable after the program started does not dynamically hide or unhide the menubar from the current screen, but it will take effect upon the next screen-change and thereafter. To have the correct appearance at startup, set or unset this variable in an initialization file.

nonobang

BOOLEAN COMMAND-LINE ONLY

If you set the **nonobang** variable, Z-Mail avoids expanding "bang" (**!**) history references that do not match anything. This is useful if you want argument referencing in **cmd** expansions but do not want to escape every exclamation point (**!**) you type in UUCP addresses.

See also the **ignore_bang** variable on page 283.

For example, if **nonobang** is *not* set, then the command

mail chiapt!hank

will result in an error if there is no command in the command history that starts with **hank**. Setting **nonobang** on will cause Z-Mail to accept **!hank** without complaint.

nosave

BOOLEAN

If you set the **nosave** variable, Z-Mail does *not* store a backup copy of aborted message compositions.

Normally, when you abort a message, Z-Mail saves a copy of it to the *dead.letter* file in your home directory or to the file described by the **dead** variable. In command-line mode, the **~E** command saves the message you are editing in the **dead** file and clears the message buffer so you can compose a new message. For more information, see *Tilde-Escape Editing Commands* beginning on page 91.

The setting of **nosave** overrides the setting of **verify**.

openfolders

READ-ONLY STRING

The list of currently open folders is held as a string of numbers in the **openfolders** variable in the format:

#0 #1 #2 #5 ...

This variable is convenient for performing an operation on every open folder. For example, the following one-line script updates every open folder:

```
foreach f ($openfolders) 'update $f'
```

output

READ-ONLY STRING

Z-Mail sets the **output** variable to the message list output by your last successful command. This variable is most useful in command-line mode and in scripts. You can use **output** to recover from broken pipes or to capture the output of a command without affecting the information it displays.

Commands which return an error code (see the **status** variable) do not affect the value of **output**. Successful commands that return no message list set **output** to an empty string.

For more information on piping message lists, see the **input** variable on page 285.

pager

STRING *COMMAND-LINE ONLY*

When Z-Mail displays a message longer than the number of lines you specify with the **crt** variable, or you execute the **page** command, Z-Mail executes the program you specify in the **pager** variable.

To tell Z-Mail to use its own internal pager, set the **pager** variable to internal or to an empty string.

If you do not set the **pager** variable, Z-Mail checks your *PAGER* environment variable. If this isn't set, Z-Mail uses the pager commonly used for your operating system, if available. If Z-Mail cannot find value to use for **pager**, it uses its internal pager.

Z-Mail checks the value of **pager** when you enter a **~p** tilde-escape command and at other times when Z-Mail is displaying lengthy text objects.

picky_mta

MULTIVALUED / BOOLEAN

The **picky_mta** variable controls how Z-Mail formats certain headers when it passes mail to the MTA (mail transport agent).

- If **picky_mta** is set on with no value or includes omit_from as a value, Z-Mail does not pass *From:* and *Date:* headers to the mail transport agent when you are sending mail. You should use this setting if your MTA generates its own *From:* and *Date:* headers.
- The omit_commas value causes Z-Mail to omit commas from outgoing mail addresses.
- uucp_style causes Z-Mail to format all outgoing mail addresses in UUCP format (*host!user*). Use uucp_style if the MTA has trouble with Internet format (*user@host.domain*) addresses.

pop_options

BOOLEAN / MULTIVALUED

Set the **pop_options** variable to **preserve** to have your mail left on the POP server. (This is analogous to setting the **hold** variable in non-POP environments.)

pop_timeout

NUMERIC

The **pop_timeout** variable specifies the number of minutes between checks for new mail on the POP server. If you aren't using POP, set the **timeout** variable instead.

post_indent_str

STRING

If you set the **post_indent_str** variable and you *include* the body of a message into an outgoing message, Z-Mail inserts the contents of **post_indent_str** after the last line of the included message. You can use any of the formatting characters listed in *Message Information Variables* on page 229 in **post_indent_str**.

EXAMPLE For example, if **post_indent_str** is set as follows:

```
set post_indent_str = 'End of excerpt "%s" from %n. \n'
```

then when you include a message in an outgoing message, Z-Mail follows the included message with your newly defined text string:

```
> . . .
> This is a test message to show how
> you can use post_indent_str.

End of excerpt "Design meeting" from Dan Smith.
```

Z-Mail also precedes each line of the included message with the **>** character unless you set the value of the **indent_str** variable to something else. For details, see the **indent_str** variable on page 283.

pre_indent_str

STRING

If you set the **pre_indent_str** variable and you include the body of a message into an outgoing message or reply, Z-Mail inserts the contents of **pre_indent_str** before the first line of the included message. You can use any of the formatting strings listed in *Message Information Variables* on page 229 in **pre_indent_str**.

EXAMPLE For example, if **pre_indent_str** is set as follows:

```
set pre_indent_str = 'Message “%s”, \n on %7d from %n: \n'
```

then when you include a message in an outgoing message, Z-Mail precedes the included message with your newly defined text string:

```
Message “Design meeting”,
```

```
on Jan 19 from Dan Smith:

> This is a test message to show how
> how you can use pre_indent_str.
> . . .
```

Z-Mail precedes each line of the included message with the **>** character unless you set the value of the **indent_str** variable to something else. For details, see the **indent_str** variable on page 283.

presign

BOOLEAN / STRING

When **presign** is set, Z-Mail will add the signature to a composition as soon as it is initiated, rather than when it is sent. The contents of the signature are determined by the **autosign**, **autosign2**, and **fortune** variables.

If **presign** contains an address list, those addresses are used to determine the signature if the compositon has an empty To: field when it is initiated. For example, if **presign** contains an address that is also in **autosign2**, then the signature file specified in **autosign2** is generated whenever a composition is generated with an empty address list.

printer

STRING

Set the **printer** variable to the name of the printer to use as the default printer when you use the **lpr** command. Normally, the printer name you supply is passed directly to the UNIX *lpr* command, prefixed with *–P* or *–d*, depending on your system. See also the **printer_opt** variable.

If **printer** starts with a hyphen (–), then no prefix is added when its value is passed to the *lpr* command.

If printer contains a comma-separated list of printer names, then Z-Mail will display the list when you issue the **lpr** command in the command-line interface so that you may choose from the list of names. In the GUI and Lite interfaces, the list of printers is available in the *Printer* dialog box.

Note that if either **printer** or **printer_opt** is set *and* empty, then both variables are ignored. This usually causes output to go to your system's default printer.

printer_opt

STRING

The value of **printer_opt** is appended as command line options to the **print_cmd** (usually *lpr*) that is invoked when you print messages.

If **printer_opt** is not set (undefined), then the default option of *–P* (for BSD UNIX) or *–d* (for other systems) is used.

The string **%P** in **printer_opt**, if present, is replaced with the setting of the **printer** variable.

Note that if either **printer** or **printer_opt** is set *and* empty, then both variables are ignored.

print_cmd

STRING

Set the **print_cmd** variable to specify the UNIX command Z-Mail will use to send messages to a printer for output.

If **print_cmd** is empty or unset, then either *lpr* or *lp* is used, depending on your operating system. The **printer** and **printer_opt** variables will be appended to the *lpr* or *lp* command as described in those variables' sections above.

If you supply a command in **print_cmd**, if the **%O** string appears in the command it will be replaced by the value of **printer_opt**, and if **%P** appears it will replaced by the value of **printer**. Note that if you use **%O**, the value of **printer** is already embedded in the **printer_opt** string, so you do not need to specify **%P** as well.

EXAMPLE The following commands send all messages from the fifth message through the last message to the printer specified by **printer** using the *ptroff* command:

```
set print_cmd = 'ptroff –ms –P%P'
lpr 5–$
```

prompt

STRING COMMAND-LINE ONLY

In the command-line interface, Z-Mail prompts you when it is ready for your next command. You can customize this prompt to display information you consider useful.

When Z-Mail displays a prompt, it evaluates the **prompt** variable and echoes each character in the string unless it encounters a **%** character identifying the beginning of a folder formatting string. For a list of valid strings, see *Folder Information Variables* on page 231.

By default, the **prompt** variable is set to the string:

Msg %m of %t:

So if you have 10 messages and your current message is 5, then your Z-Mail prompt looks like:

Msg 5 of 10:

To include the values of variables in your prompt string, you can use the format **%$*variable***, where ***variable*** is the name of any variable. If you do not precede the **$** character with a **%** character, Z-Mail treats the **$** as a normal character.

For example, two equivalent ways of including the name of the current folder in your prompt are:

set prompt = '%F> '
set prompt = '%$thisfolder> '

The single quotes prevent Z-Mail from expanding the **thisfolder** variable at the time you set the value of the **prompt** variable. Instead, Z-Mail evaluates **thisfolder** each time it displays the prompt. The only difference between these two examples is that **%F** adds the string [read-only] if Z-Mail opened the folder in read-only mode.

quiet

BOOLEAN / MULTIVALUE

The **quiet** variable tells Z-Mail to suppress warnings and user reports under specific circumstances. You may set **quiet** to one or more of these keywords separated by spaces or commas:

autosign	Suppress report when appending signature.
blink	Suppress flashing and beeping of error dialogs in GUI interface.

complete	Suppress word completion error bells.
fortune	Suppress report when appending fortune.
newmail	Suppress new mail notification report.
pick	Suppress descriptions of pick searches.
save	Suppress verbose descriptions of saving.
startup	Suppress the startup report.

If you set **quiet** on, but specify no list of keywords, it is the same as setting **quiet** to **startup**, and Z-Mail suppresses its startup report.

The **newmail** setting does not prevent automatic inclusion of new mail, it only suppresses the indicators of its arrival (such as the bell Z-Mail sounds in the GUI interface).

Z-Mail always reports error conditions for signatures and fortunes. See the **autosign** and **fortune** variables for more details.

realname

STRING

Z-Mail initializes **realname** to the value of the *NAME* environment variable when you start Z-Mail. If *NAME* is not set, Z-Mail looks in the */etc/passwd* file, if available, for your name.

The **realname** variable is used when a *From* address is generated for inclusion in your outgoing mail. Note that some MTAs ignore *From* headers that Z-Mail generates, so setting **realname** may have no effect on your system.

If you want to reset **realname** after you have started Z-Mail, you can issue a command like this:

```
set realname = "Frank Gigliotti"
```

record

STRING

Set the **record** variable to the full pathname of a folder where you want Z-Mail to save a copy of all outgoing mail. The pathname may begin with a plus sign (**+**) indicating the value of the **folder** variable, or with a tilde character (**~**), indicating the value of the **home** variable.

The **record** variable tells Z-Mail to save entire messages including their body text. To save just the headers of outgoing messages, see the **logfile** variable. If you set both the **logfile** and the **record** variables, then the log file and the record files match closely.

You may toggle message recording off for a particular composition by unsetting the *Record Message* toggle in the *Options* menu.

record_users

BOOLEAN

The **record_users** variable allows you to record all outgoing messages you send to a specific set of users. If you set the **record_users** variable, Z-Mail compares folders in the **folder** directory (*~/Mail* or the value of the **folder** variable) against the user name of each message recipient. If a folder matches a recipient's name, Z-Mail appends a copy of the outgoing message to that folder.

Recipient names and corresponding folder names are determined by taking the word to the left of a **@** character, or to the right of a **!** character, or both, in outgoing addresses. For example, the following addresses all look for a folder **+steve**:

steve@cmu.edu	Internet form
pixied!marvin!steve	UUCP form
hobbit!steve@cmu.edu	Combination

If **record** and/or **logfile** are also set, Z-Mail will also append the outgoing message to those folders.

You may temporarily turn off **record_users** for a particular composition in the Lite and GUI interfaces by unsetting the *Record Users* toggle in the *Compose Options* dialog. Neither unsetting the *Record Message* toggle in the *Options* menu of the Compose window, nor the **~l!** tilde-escape command, affects **record_users**.

Note: The folder must exist before Z-Mail places a copy in it. To create a new folder, save a message to a folder whose name matches the username for which you want to record outgoing messages, or use the *File* toggle of the *Folder Manager* dialog, or use a UNIX command such as *touch* to create the folder first.

recursive

BOOLEAN

If you set the **recursive** variable, Z-Mail allows user-defined functions to call themselves up to 100 times. If you unset **recursive**, Z-Mail does not allow functions to call themselves.

WARNING: If you set the *recursive* variable, make sure the functions you call have a way of terminating themselves.

redirect

READ-ONLY BOOLEAN *SEND-ONLY MODE*

Z-Mail sets **redirect** to true if input to Z-Mail is redirected. Any time Z-Mail is running but it is not directly connected to a terminal, it believes that input has been redirected.

If you specify the **–interact** option in your **zmail** command to force an interactive session, Z-Mail sets **redirect** to false. This variable is read-only; you cannot reset its value, but it is useful for testing in initialization files.

reply_to_hdr

STRING

If you set the **reply_to_hdr** variable to a list of headers (delimited by spaces or commas), then Z-Mail searches that list for return paths. If none of the headers listed in the **reply_to_hdr** variable exist in the message, then Z-Mail displays a warning message and searches the *Reply-to*, *From*, and *Return-path* headers, in that order.

If Z-Mail finds no return paths, it parses the first line of the message created by the mail transport agent. A special message header is created by most transport agents (*MMDF* does not). This header is called the *From_* header because it has no colon separating the header name from the rest of the header, and contains the return address of the message's author. If Z-Mail cannot find the return path address in the message, and the mail transport agent inserted the special *From_* header, Z-Mail uses this as the return address.

EXAMPLE You can specify the special *From_* header as one of the headers to search for:

```
set reply_to_hdr = "reply-to return-path From_"
```

Z-Mail searches the headers listed in **reply_to_hdr** in the order specified. If Z-Mail does not find the first header, then it looks for the next in the list. If Z-Mail finds none of the listed headers, it searches the headers *Reply_to*, *From*, and *Return_path*.

For more information, see the **reply** command on page 193.

save_empty

BOOLEAN

If you set the **save_empty** variable, it prevents Z-Mail from deleting empty folders. Normally, when you delete all the messages in a folder and you close the folder or change to a new folder, Z-Mail deletes the empty folder.

Note: Z-Mail never deletes the system folder, even when it is empty, because many programs require the presence of a system folder in order to function properly.

screen

INTEGER *COMMAND-LINE ONLY*

Set the **screen** variable to the number of message summaries you want Z-Mail to display in the command-line interface.

sendmail

STRING

Set the **sendmail** variable to the program and arguments you want Z-Mail to execute when it passes messages to the MTA for delivery. Normally, if the **sendmail** variable needs to be set on your system, your system administrator will have previously set it for you.

By default, **sendmail** is unset. Z-Mail normally uses a built-in MTA command that is appropriate for your system. Use caution if you decide set **sendmail** yourself, since it could completely prevent you from sending mail if you specify **sendmail** incorrectly.

See also the **verbose** variable on page 313.

show_deleted

BOOLEAN

If you set the **show_deleted** variable, Z-Mail lets you display deleted messages in command-line mode using any command that displays a message. Z-Mail also displays the headers of deleted messages with the **headers** command. In Lite and GUI modes, Z-Mail ignores **show_deleted** and displays deleted messages with other messages so that you can **undelete** them easily.

sort

BOOLEAN / STRING

Set the value of the **sort** variable to presort the messages in your system folder when you start Z-Mail. Set the **sort** variable to a string of arguments as specified in the **sort** command on page 208.

EXAMPLE The following command sorts your system folder by reverse date and status:

set sort = " –r –d –S "

The next time you start Z-Mail, the initialization file sets the **sort** variable and presorts your system folder automatically.

speller

STRING GUI and LITE ONLY

The **speller** variable holds the name of a program that will be used to check the spelling of your outgoing compositions and other editable text.

The **speller** program should expect a filename as its input, and should return the list of misspelled words it finds as its output, with one word per line.

Z-Mail will then place the list of misspelled words into the misspelled word list of the *Search and Replace* dialog for you to correct.

squeeze

BOOLEAN

If you set the **squeeze** variable, whenever you read, pipe, or save messages, Z-Mail "squeezes" all consecutive blank lines into one blank line.

status

READ-ONLY INTEGER

The **status** variable reports the success or failure status of the most recent Z-Mail command. Most commands return 0 (zero) for success or –1 (negative one) for error.

EXAMPLE

This variable is most useful in scripts to test the success of an operation before proceeding:

```
if $status < 0              # an error occurred
     return                 # so, terminate the script
endif
```

summary_lines

INTEGER GUI ONLY

Set the **summary_lines** variable to the number of message summaries you want Z-Mail to display in the message summary list of the GUI interface.

summary_fmt

STRING

The **summary_fmt** variable allows you to control the format of Z-Mail message listings, such as the **ask –list** dialog and the Main Window's *message summary list.*

When Z-Mail displays header information for a message, it evaluates the **summary_fmt** variable and echoes each character in the string unless it encounters a % character identifying the beginning of a message information variable. For a list of valid variables, see *Message Information Variables* on page 229.

You may specify a field width for each variable. For example, **%20f** displays the first 20 characters of the *From:* line.

EXAMPLES

For example, if you issue the command:

```
set summary_fmt = "%20n %M %–2N %7T   %20s"
```

Z-Mail lists headers like this:

```
  1  !  Sean O'Brien     Jul 15   8:49am   Meeting at 11:30
 >2  N  Jon Yost         Jul 16 12:10pm   Porting standards
  3     Ian Trainor      Jul 16   1:05pm   Maintenance pro-
gram
```

Note how Z-Mail always displays the message number as the first piece of information on a line.

Z-Mail extracts the address and name of authors from the *From:* header.

Z-Mail also lets you refer to the contents of headers not shown in the list of *Message Information Variables* on page 229. If **summary_fmt** contains a string of the form **%?*hdrname*?**, Z-Mail substitutes the contents of the header ***hdrname***. ***hdrname*** is not case-sensitive.

For example:

set hdr_format = "%a %n %?phone?"

causes Z-Mail to display the sender's e-mail address, name, and the *Phone:* header if present.

templates

STRING

Set the **templates** variable to a list of directories, separated by spaces, where you store prepared *message templates*. Z-Mail searches these directories when you use *Templates* option of the *Compose* command in GUI mode or the **–p *filename*** option of the **mail** command.

If the same name appears in more than one directory, Z-Mail uses the file in the last directory. Z-Mail always searches the *forms* directory in the Z-Mail library (*$ZMLIB/forms*) first.

All files that Z-Mail finds in the directories listed in **templates** are assumed to be actual forms, and are presented for selection in the *Templates* dialog.

textpart_charset

STRING

The **textpart_charset** variable specifies the character set that Z-Mail will use by default when sending text body parts in outgoing messages. If you generated text message body parts that use non-ASCII characters (Hebrew, for example), you should set this variable to the name of the character set you are using.

This variable is normally set by the system administrator. Any one of the following character sets may be specified:

Value	Common Name	Example Languages
us-ascii	ASCII	English (7-bit)

ISO-8859-1	Latin 1	Western Europe
ISO-8859-2	Latin 2	Eastern Europe
ISO-8859-3	Latin 3	Dutch, Esperanto
ISO-8859-4	Latin 4	Northern Europe
ISO-8859-5	Latin/Cyrillic	Cyrillic
ISO-8859-6	Latin/Arabic	Arabic
ISO-8859-7	Latin/Greek	Greek
ISO-8859-8	Latin/Hebrew	Hebrew
ISO-8859-9	Latin 5	Various European
ISO-2022-JP	JUNET	Japanese

The contents of this variable are not case-sensitive.

You may specify a character set that is not listed here, but you must preface it with "X-" to maintain compliance with the MIME specification.

thisfolder

READ-ONLY STRING

Z-Mail sets the **thisfolder** variable to the full pathname of the active folder (current mailbox). You cannot modify this variable. Z-Mail sets the value of **thisfolder** whenever you change active folders.

Note: Use caution when referring to **thisfolder** in initialization files since at startup, Z-Mail has not yet activated a folder and **thisfolder** does not yet have a value.

timeout

NUMERIC

The **timeout** variable specifies the number of seconds between checks for new mail.

tmpdir

STRING

Set the **tmpdir** variable to the pathname of the directory you want Z-Mail to use to write all Z-Mail temporary files. By default, Z-Mail uses your **home** directory. If Z-Mail cannot access your **tmpdir** or **home** directory, it uses a directory writable by all users, typically the */tmp* directory.

toplines

INTEGER COMMAND-LINE ONLY

The **toplines** variable tells Z-Mail the number of message lines to display when you issue the **top** command in the command-line interface. If you do not assign a value to **toplines**, Z-Mail uses the value of the **crt** variable to determine the number of lines to display.

Z-Mail displays only message bodies when you use the **top** command. Z-Mail does not count message headers as lines since it does not display them.

unix

BOOLEAN

If you set the **unix** variable, Z-Mail attempts to execute all commands that are not Z-Mail commands as UNIX commands. If you set **unix**, you can execute UNIX commands simply by typing them instead of using shell escapes or the **sh** command.

In the GUI and Lite interfaces, if the UNIX command you execute does not complete within a few seconds, a *Task Meter* appears to let you know that an external program is running.

If a Z-Mail command name conflicts with a UNIX command, use the **sh** command to force Z-Mail to execute the command using a UNIX shell, or specify the full pathname of the command (i.e., starting with a slash **/**).

Note: Due to the lack of complete job control, input/output redirection, and UNIX command piping, this mode of Z-Mail is not intended to be used as a login shell. For systems that support job control (*SIGTSTP*), the key sequence **Ctrl-Z** usually stops the entire Z-Mail command-line interface as well as the process being executed. When *SIGCONT* (e.g. *fg*) is generated, both processes receive the signal and Z- Mail continues to wait for the job to finish.

WARNING: Z-Mail pipes transmit message lists, not text. You cannot pipe the output of UNIX commands to or from Z-Mail commands or other UNIX commands with the Z-Mail pipe mechanism. You can, however, pipe Z-Mail commands to a final UNIX command. See the **pipe** command for details.

user

READ-ONLY STRING

The **user** variable holds the username of the user that started Z-Mail. This information is obtained from the system's */etc/passwd* file. You may set **user** to a different value if you wish; Z-Mail does not internally reference this variable.

The value of **user** is not saved by the **saveopts** command.

verbose

BOOLEAN

If you set the **verbose** variable, Z-Mail passes the *verbose* (*–v*) option to the mail transport agent (MTA) when it sends messages. Setting **verbose** also causes Z-Mail to display technical information about the sending process as it progresses.

If you set **verbose**, communication with the MTA takes place synchronously. This means that Z-Mail waits for the entire sending process to complete before returning control to you. In most cases the send process takes only a few seconds, but it can take as long as several minutes.

Note that the *–v* (verbose) option that Z-Mail passes to the MTA in the sending process when **verbose** is set is not a valid option for some MTAs. If your MTA does not recognize *–v*, you may instead place the correct mail-sending command and options in the **sendmail** variable. Use caution if you decide to try this, since it could completely disable mail sending if you specify **sendmail** incorrectly.

verify

BOOLEAN / MULTIVALUE

The **verify** variable causes Z-Mail to request confirmation of certain actions. You can set *verify* to one or more of these keywords, separated by spaces or commas:

addresses	Display addresses and ask for confirmation before sending letter.
close	Confirm updating a folder only before it is closed.
dead	Confirm saving a dead letter (GUI and Lite only).
mail	Confirm sending of messages.
mkdir	Ask for confirmation before automatically creating directories.
remove	Confirm before removing a file.

update	Confirm before updating a folder.
quit	Confirm before exiting.

By default, **verify** is set to dead,remove,quit. If you set **verify** on but empty, this is equivalent to setting the mail value.

Z-Mail asks for verification whenever it appends messages to files that are not folders, regardless of the setting of **verify**.

version

READ-ONLY STRING

Z-Mail stores its current version number in the *version* variable. Z-Mail displays this information when it starts unless you suppress its display by setting the *quiet* variable. Z-Mail also places current version information in the *X-Mailer:* header in outgoing messages.

visual

STRING

Set the **visual** variable to the name of the text editor you want to use when you use the **~v** command in command-line interface, If you set **visual**, but do not specify the name of an editor, Z-Mail uses the value of the *VISUAL* environment variable or *vi*.

warning

BOOLEAN

If you set the **warning** variable, Z-Mail displays warning messages under a variety of situations (detailed below). By default, **warning** is unset.

The purpose of the **warning** variable is to provide users who are comfortable with their own environments a way of detecting changes in other environments, should they be forced to use them. In an integrated environment, you can also receive messages about other processes failing that are not severe enough to stop Z-Mail from functioning properly.

Warn when a command line alias (cmd) looks like a command.
For example:

```
cmd mail 'set fortune; \mail'
cmd respond 'unset fortune; \respond;'
```

Z-Mail displays a warning when you use the **cmd** alias, not when you define it.

Warn when the date format of a message is unknown. Z-Mail takes the date of a message from the *Date:* header. If the date on that header is unknown, Z-Mail searches other headers for a valid date format until a legal one is found. Z-Mail displays a warning when it finds no date or an incomprehensible date.

Warn when a variable is **unset** ***without first being*** **set.** For example, if you give the command:

unset metoo

and the **metoo** variable is not set, you are notified that the variable is not defined.

Warn when no header can be found for a digest article. This occurs when the **undigest** command encounters what appears to be an article separator but cannot find a *From:* or *Date:* header in the subsequent text.

window_shell

STRING GUI ONLY

Set the **window_shell** variable to the name of the program to run when you create shell processes in GUI mode. If **winterm** is set, the default value for **window_shell** is **winterm.** If **winterm** is undefined, Z-Mail will use *xterm -e*.

If **wineditor** is not set, Z-Mail starts an external editor in the GUI interface by appending the **visual** variable to **window_shell** and executing the result as a UNIX command. You should set **window_shell** to an empty string if the **visual** editor creates its own window.

wineditor

STRING

By default, the **wineditor** variable is set to the value of the WINEDITOR environment variable. When you invoke an external editor in GUI mode, the program specified by **wineditor** is executed in favor of the program specified by the **visual** or **editor** variables.

To make Z-Mail start the **wineditor** editor every time you compose or reply to a message, set the **autoedit** variable.

If **wineditor** is not set, Z-Mail invokes the external editor in GUI mode by appending the **visual** variable to **window_shell** and executing the result as a UNIX command.

winterm

STRING

Set the **winterm** variable to the name of the program to run when you create shell processes in GUI mode. If the WINTERM environment variable is set, **winterm** is set by default to its value. If **winterm** is unset, the value of **window_shell** is used instead.

wrap

BOOLEAN

If you set the **wrap** variable, when you move beyond the last message in the active folder using the **next** or **delete** commands, Z-Mail moves the pointer to the first message in the folder that isn't deleted or saved.

wrapcolumn

INTEGER

Set **wrapcolumn** to a column number where you want Z-Mail to automatically wrap lines as you compose messages. If you set **wrapcolumn** on, but specify no column number, Z-Mail wraps at column 78.

If Z-Mail can determine the number of columns on your screen, it enforces a maximum value for **wrapcolumn** of two less than that number of columns. You can disable line wrapping either by unsetting **wrapcolumn** or by setting it with a value of 0 (zero).

Line wrapping occurs only at whitespace (spaces or tabs). Z-Mail will not wrap lines that contain no whitespace and more characters than the value of **wrapcolumn**.

Note: If you start Z-Mail in command-line mode with the **–e** (echo) option, Z-Mail cannot perform line wrapping.

$$

READ-ONLY STRING

The **$$** variable holds the process id (*PID*) of the current Z-Mail process. Z-Mail does not recognize colon modifiers for this special variable.

The PID is a useful pseudo-random string that is commonly used in Z-Script functions as part of the name of temporary files.

EXAMPLE The following command displays the process id of the Z-Mail process:

```
echo $$
10326
```

$*

READ-ONLY STRING

When Z-Mail executes a user-defined function, it sets the **$*** variable to the entire list of arguments specified by the command that called the function, except for the name of the function itself.

Arguments to the **zmail** UNIX command are available in initialization files via **$***. See *Starting Z-Mail* on page 34 for information about passing arguments into initialization files.

You can use this variable in conjunction with the **shift** command to pick off arguments as you need them. If you know the order in which the arguments were specified, you can use the variables **$1**, **$2**, etc.

EXAMPLE Assume you invoked the function **treport** with the following command indicating the recipients of the report as arguments:

```
treport kwilson pramirez dfrank
```

Z-Mail assigns all the arguments you specified to the **$*** variable as one character string.

```
echo $*
kwilson pramirez dfrank
```

For more information, see the **shift** command on page 206 and the **$#** and **$1**, **$2**, ... variables below.

$#

READ-ONLY INTEGER   FUNCTIONS ONLY

When Z-Mail processes a script, it sets the **$#** variable to the number of arguments you specified on the command line that called the function, counting from zero and including the name of the function itself. You can use this variable in conjunction with the **shift** command or other variables such as **$*** and **$1**, **$2**, and so on.

EXAMPLE Assume you invoked the function **treport** with the following command:

```
treport kwilson pramirez dfrank
```

Z-Mail assigns the number of arguments to the **$#** variable.

```
echo $# $*
3 kwilson pramirez dfrank
```

For more information, see the **shift** command on page 206 and the **$*** and **$1**, **$2**, ... variables.

$0

READ-ONLY STRING

The **$0** variable contains the name of the called function.

EXAMPLE Assume you invoked the function **treport** with the following command:

```
treport kwilson pramirez dfrank
```

Z-Mail assigns the name of the function to the **$0** variable.

```
echo $0
treport
```

$1, $2, . . .

READ-ONLY STRINGS

When Z-Mail processes a script, it sets a series of variables named **$1**, **$2**, ... **$n** equal to the individual arguments specified in the command that called the function. These variables are also called *positional parameters*.

You can use these variables in conjunction with the **$#** variable, whose value is set to the number of arguments to the function.

Arguments to the **zmail** UNIX command are available in initialization files via positional parameters. See *Starting Z-Mail* on page 34 for information about passing arguments into initialization files.

EXAMPLE

Assume you invoked the function **treport** with the following command indicating the recipients of the report as arguments:

treport kwilson pramirez dfrank

Z-Mail assigns all the arguments you specified to the **$1**, **$2** and **$3** variables, and the number of arguments (three) to the **$#** variable.

echo $3 $#
dfrank 3

For more information, see the **$#** variable.

CHAPTER 7

Integrating Z-Mail with Other Products

Z-Mail is designed to use standard, nonproprietary UNIX mechanisms to exchange data with other products. By integrating other products into the Z-Mail system, you can greatly increase the value of your e-mail infrastructure. With proper configuration, e-mail can be used as a flexible, reliable data transfer medium as well as a communications medium.

Z-Mail supports the following kinds of data interchange and integration:

- Plain text (ASCII) created in other programs can be imported into and exported from e-mail messages. See the following section *Importing Text Files into Messages*, the **save**, **copy**, **write**, and **pipe** commands, and the *Z-Mail User Guide* for more information.
- Other e-mail programs' folders can be read and written by Z-Mail. See *Mail Folders* on page 17 for more information.
- Other applications can be launched from Z-Mail. See *Launching Applications from Z-Mail* on page 325 for more information.
- Any type of document or program can be attached to a message. Your site's *attach.types* file defines the mechanisms Z-Mail will use when users create, view, and detach attachments. See *Defining Attachment Types* on page 328

for more information.

By combining these tools, you can use Z-Mail to build a desktop environment that integrates the office automation, database, and productivity applications you use regularly. See *Automatically Processing Attachments* on page 327 for a brief example of an integrated system.

Importing Text Files into Messages

Z-Mail can directly import text files created by text editors, word processors, desktop publishing applications, or other products that can export text files.

Including the text of a file is not the same as attaching a document to an outgoing mail message. When you include a text file, Z-Mail treats the contents of the file as if it were text typed by you into the body of the message. Once you include a text file, you can edit its contents. An attachment, on the other hand, is a self-contained document that cannot be directly edited by Z-Mail. See the **~A** command on page 97 and the *Z-Mail User Guide* for instructions on creating and using attachments. See *Defining Attachment Types* on page 328 for information on administrating your site's attachment mechanisms.

The following paragraphs explain how to include text from an external file using Z-Mail's various interfaces.

All interfaces

In all Z-Mail interfaces, you may use any of the following options to the **mail** command to include a text file in a message:

- **mail –H** reads a file into the body of the message
- **mail –h** reads a file into the headers and body of a message.
- **mail –p** reads a template file from the directory specified by the **templates** variable.

GUI and Lite

When you are composing a message using the GUI or Lite interface, pull down the *File* menu and select the *Open / Insert Text* or *Open / Replace Text* item to include a file in the message.

The *Insert Text* item inserts the file at the cursor position, while *Replace Text* replaces the body of the message with the file. If the composition is in *edit headers* mode, the file replaces the headers as well.

See the *Z-Mail User Guide* for more information about including text files in a message composition.

Command-line

When you are composing a message using the command-line interface, if you type **~r** ***filename*** at the beginning of a line, Z-Mail reads the contents of ***filename*** into your outgoing message. See the **~r** tilde-escape command on page 96 for more information.

You may *attach* files to an outgoing message with the **~A** tilde-escape command; see page 97 for more information.

Send-Only Mode

If you use the **zmail** command in send-only mode, you can specify any one of the following options to include a file with a message:

- the **–draft** option to send an entire file as a message. For more information, see the **–draft** option on page 43.
- the **–direct** option passes the standard input of the zmail command directly to the MTA for delivery. For more information, see the **–direct** option on page 43.
- the **–attach** option attaches a file to an outgoing message. For more information, see the **–attach** option on page 40.

Launching Applications from Z-Mail

You can launch other applications and run UNIX commands from within Z-Mail. Some reasons to take advantage of this capability are:

- You want to create a document, with another application, that you want to send using Z-Mail.
- You receive an application's document as an attachment, and you want to start that application to display or work with the document.
- You want to run a UNIX command or utility.

Entering UNIX Commands from Z-Mail

You can issue a command to start another application or a UNIX program from within Z-Mail using a variety of methods that vary depending on your Z-Mail interface.

All interfaces

When the **unix** variable is set, all commands that are not recognized as Z-Script commands will be run as UNIX commands. See the **unix** variable on page 312 for more information.

GUI

Type **sh** followed by the UNIX command in the Command type-in field at the bottom of Z-Mail's main window (e.g., **sh maker**).

Lite

Applications that run in their own window (on a window-system equipped workstation) can be started in the same way as in GUI mode, above. Applications that want to take control of the terminal screen, such as **vi** or **emacs**, should be prefixed with the **screencmd** command. For example, you'd use the command **sh maker** to start FrameMaker, which runs in its own window, or **screencmd –p sh more** to run the **more** program, which needs to take over the terminal from Lite.

Command Line

Type **sh** followed by the command at Z-Mail's command prompt (e.g., **sh maker**).

Functions You can include UNIX commands in Z-Mail functions by preceding them with the **sh** command. In the Lite interface, you may also need to use **screencmd**; see ***Lite*** above.

Launching Applications from Buttons

The information in this section is also available in greater detail in the *Z-Mail User Guide*.

GUI and Lite only To create a function and button that launches applications, follow these steps:

1. Select the *Buttons* item from the *Options* pull-down menu, and the *Buttons and Functions* dialog box appears.
2. Click the *Clear* button at the bottom of the dialog box.
3. Enter the name of the button in the *Button:* type-in field. Make sure you enter the name exactly as you want it to appear in Z-Mail's main window.
4. Enter a name for the function you are about to create in the *Function:* type-in field. Choose any unique name you want, but avoid the names of Z-Mail and UNIX commands.
5. In the large type-in box labeled *Function Text:* in the lower part of the dialog box, enter **sh** followed by a space and the command that starts the application you want to launch (e.g., **sh maker**). See the previous section for more information about the Z-Script commands that start applications.
6. Click the *Install* button and Z-Mail creates a button in its main window.
7. If you want to save this button so it appears every time you start Z-Mail, click the *Save* button at the bottom of the dialog box to save your current settings in your personal initialization file. Z-Mail displays a dialog box that prompts you for a filename. To indicate your default personal initialization file, *~/.zmailrc*, just click *OK* without typing in another name. Z-Mail may ask you to confirm that you want to overwrite your *~/.zmailrc* file.
8. Click the *Done* button, and Z-Mail returns you to its main window.

9. Click the button you just created to start the application.

Automatically Processing Attachments

Integrated User Desktops

By specifying the appropriate program entries in the **TYPE** statements of your *attach.types* file, you build your own integrated user desktops. For example, you can create:

- A purchase requisition form in a Wingz spreadsheet,
- A function called **purchase_req** that starts Wingz, loads the "purchase req" spreadsheet, and waits for the user to press a button indicating they are finished with the form.
- A button labeled *Purchase* that runs the **purchase_req** function.
- An *attach.types* entry that automatically starts Wingz when Z-Mail receives a Wingz document.

Using these system components, you can, for example, construct a purchasing system that all the e-mail users in your organization can use. They fill out purchase orders electronically and e-mail them to appropriate addresses for approval. When the final approval is in place, Z-Mail can submit the purchase orders to a central purchasing agent or even a special userid that periodically uses *cron* to process the forms and update the purchasing database. For more information about *cron*, see the *cron* UNIX manual page.

Defining Attachment Types

Z-Mail allows you to attach files to your messages that can be easily displayed or detached by the messages' recipients. You can specify any type of attachment, including binary, text, video, audio, graphics, and PostScript; or even application files such as FrameMaker and Wingz documents.

This section describes how you can define new attachment types that will automatically be correctly attached to your outgoing mail and detached from your incoming mail.

To learn more about attaching files and documents using Z-Mail's graphical user interface, see the *Z-Mail User Guide*. In the command-line interface, use the **–attach** option (**–A**) of Z-Mail's **mail** command.

NOTE: Much of the information in this section is intended for system administrators and users who have comparable UNIX experience. Facility with UNIX shells, pipelines, and input/output redirection is assumed.

The Attachment Types File

Before you can attach a file or document to your message, you must first tell Z-Mail how to recognize and handle that type of document. Z-Mail comes pre-configured to recognize a variety of documents including text, graphics, bitmaps, Wingz, FrameMaker, PostScript and other files.

You may want to define your own types of attachments so Z-Mail can recognize, display and send those attachments automatically. You define attachments by making new entries in the *attach.types* file located in the *$ZMLIB* (usually */usr/lib/Zmail)* directory.

NOTE: Z-Mail obtains attachment type information at start-up. If you modify the *attach.types* file(s), you can use the **attach** command (see page 122) to cause Z-Mail to reread the *attach.types* file(s).

Normally, one system-wide *attach.types* file contains definitions for all attachment types. You may, however, choose to create several *attach.types* files that you use in different Z-Mail applications or user environments. You tell Z-Mail which attachment definitions file(s) to use by setting the value of the **attach_types** variable, or with the **attach** Z-Script command.

attach.types Format

The *attach.types* file contains a series of statements that define:

- the attachment types that Z-Mail recognizes,
- the programs Z-Mail uses to display the attachments,
- the programs Z-Mail uses to encode and compress attachments before sending them,
- the programs Z-Mail uses to decode and uncompress attachments it receives, and
- the directories Z-Mail will search to find the above programs.

The format of the *attach.types* file is:

PATH ***directory1*:*directory2*:...**
. . .
TYPE ***attachtype1*** **"*viewercmd1*"** **"*editorcmd1*"** **"*comment*"**
TYPE ***attachtype2*** **"*viewercmd2*"** **"*editorcmd2*"** **"*comment*"**
. . .
CODE ***codetype1*** **"*encodecmd1*"** **"*decodecmd1*"** **"*comment*"**
CODE ***codetype2*** **"*encodecmd2*"** **"*decodecmd2*"** **"*comment*"**
. . .
ENCODE ***attachtype1*** ***codetype1***
ENCODE ***attachtype2*** ***codetype2***
. . .
DEFAULT ***codetype***

You may continue an entry on the next line by placing a backslash (****) at the end of the preceding line. The ordering of entries in *attach.types* is not significant.

EXAMPLES

Here is a typical *attach.types* file:

```
PATH      /usr/lib/Zmail/bin:/bin:/usr/bin:/usr/local/bin
TYPE      Wingz      "Wingz %s"       None           "Wingz"
TYPE      bitmap     "bitmap %s"      None           "X11 bit-
     map"
TYPE      xwd        "xwud –in %s"    "xwd –out %s"  "Xwd File"
CODE      compress   "| compress | btoa |"  "| atob | uncompress |"  \
          "Compressed data”
ENCODE    Binary     compress
```

You can place a *comment* on a line by itself by preceding the comment with a **#** character:

```
#   The following lines define three Z-Mail attachment types:
TYPE    Wingz     "Wingz %s"     None   "Wingz"
TYPE    bitmap    "bitmap %s"    None   "X11 bitmap"
TYPE    jpeg      "djpeg < %s | xv –"    None   "JPEG image"
```

PATH Statement

The **PATH** statement designates directories where you want Z-Mail to look to find the programs used for processing attachments.

If no **PATH** statement is specified, Z-Mail uses the *PATH* environment variable’s setting. The **PATH** statement, if specified, overrides the *PATH* environment variable for attachment processing. Z-Mail always prepends *$ZMLIB/bin* to the path, regardless of whether it was obtained from a **PATH** statement or the *PATH* environment variable.

WARNING: Some otherwise useful decoding and viewing programs that are capable of running UNIX shell commands are potential security risks. For example, some PostScript viewers can execute UNIX commands that are embedded in the PostScript file, possibly causing damage (intentional or unintentional) to your system. For this reason, you may wish to specify a **PATH** statement that only allows the use of attachment-related programs that you consider to be safe.

SYNTAX

PATH *directory1:directory2:directory3 . . .*

If more than one **PATH** statement is present, Z-Mail appends the directories you specify in each **PATH** statement to the values specified in the previous statements.

Note: Z-Mail always searches the path */usr/lib/Zmail/bin*, so you do not need to specify it in your **PATH** statement.

CODE Statement

The **CODE** statement tells Z-Mail the encoding and decoding commands to use when processing attachments.

SYNTAX

CODE ***"codetype"*** **"*encodecmd*"** **"*decodecmd*"** **"*comment*"**

Coding Type ***(codetype)***
Indicate the name of the coding type as it appears in the **TYPE** statement. The ***codetype*** may include spaces and non-alphabetic characters if it is enclosed in quotes.

Encoding Program ***(encodecmd)***
Indicate the command Z-Mail runs to encode attachments of type ***codetype***. If there is no encoding command, enter NONE.

Decoding Program ***(decodecmd)***
Indicate the command Z-Mail runs to decode attachments of type ***codetype***. If there is no decoding command, enter NONE.

Comment ***(comment)***
Add a comment explaining the coding type as a quoted text string. The ***comment*** may be omitted; it does not appear in the user interface.

The text strings you specify for ***encodecmd*** and ***decodecmd*** are treated by Z-Mail as Bourne shell (*/bin/sh*) commands. If they include spaces or nonalphabetic characters, they should be quoted.

You must specify the **CODE** statement with one of the following:

- leading and trailing pipe characters,
- a **%s** and one pipe character, or
- a single **%s**.

If ***encodecmd*** or ***decodecmd*** begin with a pipe character (**|**), Z-Mail sends the attachment data to the command as input. If ***encodecmd*** or ***decodecmd*** ends with a pipe character, Z-Mail reads the output of the command from the UNIX standard output pipe.

If ***encodecmd*** or ***decodecmd*** contains **%s**, Z-Mail inserts a filename at that point so ***encodecmd*** or ***decodecmd*** may read data from that file. If your system supports filenames with spaces in them, **%s** must be enclosed in quotes to protect the spaces.

Encoding programs must produce seven-bit ASCII text as output for proper transmission of the data through the mail system.

You can specify as many **CODE** statements in your *attach.types* file as you need.

TYPE Statement

The **TYPE** statement defines the attachment types that Z-Mail recognizes.

SYNTAX **TYPE** ***"attachtype"*** **"*viewercmd*"** **"*editorcmd*"** **"*comment*"**

Attachment Type ***(attachtype)***
Indicate the name of the attachment type as a quoted text string. The value of ***attachtype*** appears as a menu choice in the *Attachments* dialog box in the GUI and Lite interfaces or as a menu selection in the command-line interface. ***attachtype*** may include spaces or nonalphabetic characters if it is quoted.

Viewer Command ***(viewercmd)***
Enter a quoted text string indicating the name of the command Z-Mail runs to display the attachment. If there is no viewing command, enter NONE.

Editor Command ***(editorcmd)***
Enter a quoted text string indicating the name of the command Z-Mail runs to start an program where you can view and modify the attachment. If there is no editor command, enter NONE.

Comment ***(comment)***
The ***comment*** is a quoted text string that appears as the default attachment comment for ***attachtype*** in the *Attachments* dialog. Typically, ***comment*** briefly describes the ***attachtype***. For example, a **jpeg** ***attachtype*** may have a ***comment*** of "JPEG file – compressed photographic image".

The text strings you specify for ***viewercmd*** and ***editorcmd*** are treated by Z-Mail as Bourne shell (*/bin/sh*) commands. By enclosing them in quotes, you protect spaces, pipe symbols, etc.

If ***viewercmd*** or ***editorcmd*** begins with a pipe character (|), Z-Mail sends the attachment data to the standard input of the viewer or editor. You should never specify ***viewercmd*** or ***editorcmd*** with a trailing pipe character, since there is no succeeding command to receive the data. You must specify one of the following:

- a leading pipe character (|),
- one **%s** somewhere in the string, or
- neither of the above.

If you specify ***viewercmd*** or ***editorcmd*** without a leading pipe character or **%s** formatting string, Z-Mail cannot send the command the attachment data.

You may specify an ampersand character (**&**) at the end of ***viewercmd*** or ***editorcmd*** to tell Z-Mail to run the editor or viewer in the background. This is useful for programs that run in their own window, such as FrameMaker and Wingz.

ENCODE Statement

The **ENCODE** statement tell Z-Mail to use specific encoding methods for specific attachment types.

SYNTAX

ENCODE *attachtype codetype*

When you are sending mail, the **ENCODE** statement tells Z-Mail to automatically encode a document of type ***attachtype*** using the program specified by ***encodetype***. You can manually override the encoding indicated in the **ENCODE** statement during your Z-Mail session by changing the setting of the *Encoding* popup of the *Attachments* dialog. When Z-Mail encodes outgoing messages, it adds two headers to them:

- *X-Zm-Data-Type:* holds the value of ***attachtype***.
- *X-Zm-Encoding-Algorithm:* holds the value of ***encodetype***.

You can specify as many **ENCODE** statements in your *attach.types* file as you need.

DEFAULT Statement

The **DEFAULT** statement tells Z-Mail which attachment type it should assign by default to attachments that contain binary (8-bit) data.

SYNTAX

DEFAULT *codetype*

If Z-Mail determines that an attachment is binary (i.e., the attachment contains one or more 8-bit values), it will use the ***codetype*** specified in the **DEFAULT** entry to encode the attachment. Z-Mail examines the first 4 to 8KB of the attachment for 8-bit values to determine if it is binary.

Z-Mail will not permit you to specify an encoding of NONE for binary attachments when you send them, since 8-bit data cannot, in general, be handled by e-mail transport systems.

CHAPTER 7

Using Z-Mail X Resource Variables

Set these X-resource variables in app-defaults or personal X-resources files:

Zmail*maxTextLength Maximum size of all text displayed in text pager dialogs, including the Message Display Window. Default 50000.

Zmail*scrollLength Maximum size of the text in the Output Pane on the Main Window. Text is deleted from the *top* when this size is exceeded, so this is the maximum you can scroll back in that text. Default 4096.

Zmail*finderFullPaths Instructs all File Finder objects to display full path names in the File: typein field. Default FALSE (only basenames shown).

Zmail*showListAll Tells attachment icon areas to display the "List All" icon, which when clicked, brings up the attachments dialog. Default FALSE (don't show the List All icon).

Zmail*offerFolders Tells the command:

zmail -gui -folder foldername

to look for a folder server (another Z-Mail invocation) on the local display, and to attempt to hand off the named folder to that server. Default FALSE. Setting this to TRUE will crash some X servers, notably AIX 3.2.5.

Zmail*acceptFolders Tells **zmail -gui** to become a folder server (refer to **offerFolders**). Default FALSE.

Warning: Setting this to TRUE crashes some X servers, notably AIX 3.2.5, at the time a second Z-Mail starts up.

Zmail*sgiMode Applies to MediaMail and Z-Mail on IRIX 5 only. Tells Z-Mail to look like MediaMail. Might change appearance slightly on non-SGI platforms. Note that the variable **Zmail*sgiMode** is slightly different, and will tell Z-Mail/MediaMail on IRIX 5 to use (or not) the IRIX 5 custom Motif library and other specialized files. Default FALSE, except MediaMail where default is TRUE.

Zmail*iconGeometry Specifies where the Z-Mail main window icon should be placed when zmail is iconified. Default undefined (the window manager places the icon wherever it likes).

Zmail*addressAutoEdit Default TRUE, which means that selecting an address from the list copies it immediately to the text widget.

Zmail*addressAutoReplace Default FALSE, which means that pressing **Return** in the text widget adds the selected items in the list with the contents of the text.

TRUE means that pressing **Return** in the text widget replaces the selected items in the list with the contents of the text.

INDEX

Symbols

! character, in addresses 30
$ character
    in message lists 24
$# variable 318
$$ variable 317
*$** variable 317
$0 variable 318
$1, $2, ... variables 318
% character
    in addresses 29
    in information variables 229
( ) characters
    in addresses 30
    in variable notation 231
* character
    as filename metacharacter 28
    as text metacharacter 26
    in message lists 24
+ character
    as filename metacharacter 27
    defined by *folder* variable 277
    on suspended messages 169
. character
    as text metacharacter 26
    in message lists 24
:c command – see *headers* command
; character, as command terminator 89
? character
    as filename metacharacter 27
    as help command 22
@ character, in addresses 29
[ ] characters
    as filename metacharacters 28
    as text metacharacters 26
    as variable notation 229
\ character
    as indicator for control keys 104
    as text metacharacter 26
^ character
    in message lists 24
{ } characters
    in filenames 28
    in message lists 24
~ character
    as command prefix 91
    as filename metacharacter 27

A

about command 117
active folder
    described 19
    thisfolder variable 311
address book variable 248
address book, variables 241
address_cache variable 249
address_check variable 249
address_filter variable 250
address_sort variable 250
addresses
    case insensitivity in 31
    comments in 30, 32
    described 14
    domain 29
    domain suffix meanings 31
    Internet domain 29
    local 29
    removing redundant 256
    RFC822 29
    sorting 250

UUCP – see also UUCP addresses 30
addressing-related variables 242
alias command 117
aliases
command-line 131
using 87
command-line – see *cmd*
defining 117
disabling expansion of 296
expanding 142
expanding automatically 251
expanding circular definitions 142
in command-line interface 84
removing
address – see *unalias*
command-line – see *uncmd*
using in addresses 32
alternate logins
removing from outgoing mail 119
alternates command 119
alternates variable 250
alts command – see *alternates*
alwaysexpand variable 251
alwaysignore variable 251
AMS folders 18
Andrew Message System folders
see AMS folders
applications, launching 326
arguments
echoing 139
getting all (*$**) 317
getting individual (*$1*, *$2*, . . .) 318
getting name of called function (*$0*) 318
getting number of (*$#*) 318
parsing 141
parsing in function 206
passing into Z-Mail (*–shell!* option) 56
shifting 206
ask command 120
ask variable 252
askcc variable 253
asterisk
as filename metacharacter 28
as text metacharacter 26
in message lists 24
atob binary decoding 330
at-sign (@), in addresses 29
attach.types file 328
attach_types variable 253
attachments
compressing 329
creating 174
decoding 135
detaching to files 135
displaying 135
encoding 329
in command-line interface 81
location of *attach.types* file 253
receiving with other MUAs 17, 81
specifying types of 328
auto_route variable 256
autoclear variable 253
autodismiss variable 254
autoedit variable 254
autoiconify variable 255
autoinclude variable 255
autoprint 256
autosign variable 259
autosign2 variable 260
await command 123

B

Babyl folders 18
backslash character
as indicator for control keys 104
as text metacharacter 26
Bcc - see blind carbon copies
bell, suppressing 303
blank lines, squeezing out 308
blind carbon copies 172
body of message, described 16
Bourne shell, invoking the 205
braces in message lists 24
brackets
as filename metacharacter 28
as text metacharacter 26
as variable notation 229
btoa binary encoding 330
builtin command 124
button command 124
buttons

creating 124, 326
deleting 214
displaying dialog boxes using 136

C

canceling messages
in command-line interface 78
carbon copies
prompting for recipients of 253
sending to copies to files 274
sending with *mail* command 172
caret character
in message lists 24
case insensitivity in addresses 31
Cc - see carbon copies
cd command 129
cdpath variable 264
chroot command 129
circumflex
in message lists 24
close command 130
closing folders 130
closing Z-Mail window 161
cmd command 131
cmd_help variable 264
co command - see *copy*
CODE statement 331
command aliases
deleting 214
using in command-line interface
command history, explained 85
command.hlp file 22
command-line interface
appearance of 73
canceling messages 78
command aliases 131
command history 85, 160
command pipelines 88
customizing your prompt 85
deleting messages 82
displaying headers 75
displaying messages 75
forwarding messages 77
including text in messages 78
learning the 10
macros 175
printing messages 83
replying to messages 76
saving messages 83
searching messages 84
sending attachments in 81
sending messages 76
similarities to *csh, Mail, mailx* 71
sorting messages 84
starting the 20, 72
using aliases 84
zmail.small version 34, 72
command-line macros 106
command-line-related variables 245
commands
echoing 139
entering multiple 89
how Z-Mail processes 90
pipelines 88
reference 117–221
referencing history of 281
referring to previous 86
running UNIX 312
semicolon as terminator for 89
summary of 110–116
tilde-escape 91–101
using with loops 153
comments
in addresses 30, 32
in initialization files 67
compcmd command 131
complete variable
described 265
use of 27
completion, filename 27, 265, 275
Compose Window
setting number of lines in 295
compose_panes variable 267
compose-related variables 239
composing messages
clearing window after sending 253
displaying window 267
ending a compose with a dot 271
quitting while 191
suspending messages 169
wrapping at end of column 316

compositions, performing commands on 131
compressing attachments 329
conditionals
  if...else...endif command 161–166
  in initialization files 68
configuration files - see initialization files
configuration-related variables 233
confirming actions with *verify* 313
control characters in macros 104
control-D, exiting Z-Mail with 283
copy command 133
copying messages to files 133
crt variable 268
crt_win variable 269
csh - see *C-shell*
C-shell 71
curly braces in message lists 24
cwd variable 269

D

d command - see *delete*
date
  listing messages by receive 269
  searching message by 201
date_received variable 269
DBFILE environment var 42, 134
dead letter file 94, 297
dead variable 269
dead.letter file 269
debug command 133
–debug option 42
DEFAULT statement 333
delete command 134
delete_save variable 269
deleted messages, displaying 307
deleting messages
  after saving 269
  in command-line interface 82
  with *delete* command 134
desktops, integrated user 327
detach command 135
detaching attachments 135
dialog boxes
  automatically closing 254
  displaying 136
  error 141
dialog command 136
dialogs
  creating 120
digests
  bursting into messages 215
  merging into active folder 215
directory
  changing root 129
  changing your current working 269
  changing your working 129
  defining your home 281
  displaying working 190
  for templates 310
  for temporary files 311
directory-related variables 237
disable command 136
display command 137
displaying headers
  in command-line interface 75
displaying messages
  after deleting messages 256
  displaying deleted 307
  in command-line interface 75
  number of lines 268, 269
  top lines 212, 312
display-related variables 235
distribution lists - see aliases
documentation
  list of available Z-Mail 7
  obtaining 7
dollar sign
  in message lists 24
domain addresses 29
domain_route variable 270
dot variable 271
draft files 174

E

each command 138
echo command 139
echo, setting local 45
edit command 140
edit_hdrs variable 273
editor variable 274

editor-related variables 238
editors
    editing headers with 273
    invoking with *edit* command 140
    invoking with tilde-escape 93
    setting visual 314
    using external 254, 274
electronic mail, described 5
emacs editor
    starting with a macro 107
e-mail, see electronic mail
empty folders, opening 56
enable command 140
ENCODE statement 333
encoding attachments 329
environment variables 228
error bells, suppressing 303
error command 141
error status codes 309
escape characters, in messages 274
escape commands - see tilde-escape commands
escape variable 274
eval command 141
exclamation point
    as history reference 283
    in addresses 30
exit command 142
exiting
    from Z-Mail without updating 142
expand command 142
expanding aliases 296
expansions in macros, described 102
expire_cache variable 275

F

f command - see *from*
Fcc - see file carbon copy 274
fignore variable 275
file carbon copies, sending 274
file finder dialog
    formatting entires in 276
filelist_fmt variable 276
filename completion 27, 275
file-related variables 237
files
    copying messages to 133
    listing 171
    paging 186
    reading commands from 210
    removing 192
    renaming 193
    saving messages to 221
    writing messages to 221
filter command 143
filters
    avoiding filters at startup 144
    avoiding when opening folder 144
    creating 143
    deleting 216
    restricted commands in 144
    running at startup 45
    suppressing running of 52
flags command 144
fo command - see *folder*
folder command 145
folder information variables 231
folder server
    invoking not as 55
    invoking Z-Mail as 40
folder variable 277
folder_type variable 277
folder-related variables 236
folders
    AMS format 18
    Babyl format 18
    changing 145
    closing 130
    deleting when empty 307
    described 17
    handing off folders to folder server on startup 53
    listing all in a directory 152
    listing open 145
    merging 176
    MH format 18
    MMDF format 17, 18
    name of active 311
    opening 145, 186
    opening empty system 56
    opening multiple 18
    opening read-only at startup 53

overwriting when saving messages 197
read-only 146
removing 192, 196
renaming 193
Rmail format 18
saving empty 307
setting path for 277
setting type of new - see *folder_type* variable 277
specifying active at startup 46
updating 220
V7 format 17
folders command 152
foreach command 153, 298
fortunates variable 279
fortune variable 278
fortunes
adding to messages 172
appending to messages 278
specifying recipients of 279
forwarding messages 173
in command-line interface 77
FrameMaker files 81, 97, 328, 333
from command 154
function command 155
function variables 317, 318
function variables, described 224
functions
creating 155, 326
deleting 216
example of 156
listing 156
parsing arguments in 206
putting comments in 156
recursive 124, 306
saveit, as an example 156
using tabs in 156
using with loops 153
with same name as commands 124

G

gateway 15
effect on addressing 29
geometry, specifying for icons 49
getline function 105
getstr function 105
GUI buttons, creating 326
GUI interface
buttons 124, 214
closing Z-Mail windows 161
dialog boxes 136
error dialog boxes 141
icon geometry 37, 49
iconifying Z-Mail in 161, 211
is_gui variable 286
starting the 20
gui_help variable 279
GUI-related variables 243

H

:h variable modifier 228
h command - see *headers*
h+ and *h*- commands 159
hangup variable 280
hdrs_only variable 280
head of path variable modifier 228
header-related variables 234
headers
avoiding editing 140
creating 184
creating personalized 184
deleting personalized 216
described 16
disabling personalized 297
displaying 154, 157, 195
displaying in command-line 75
editing 273
ignoring 168, 217, 251
in send-only mode 48
in_reply_to 285
logging outgoing 289
modifying with tilde-escapes 91
retaining 195
searching 154
headers command 157
help
requesting 22
help command 160
help files
customizing 22

locating 22, 160
name of general 264
name of GUI 279
help-related variables 233
hide command 160
history command 85, 160
history of commands entered 85
history references, expanding 297
history variable 281
hold variable 281
home directory, defining your 281
home variable 281
hostname variable 282

I

icon geometry, specifying on startup 49
icon, specifying location on startup 49
iconify command 161
iconifying dialog boxes automatically 255
icons
new mail 296
specifying placement 37, 49
Z-Mail GUI 290
if statements
in initialization files 68
if...else...endif command 161–166
ignore command 168
ignore_bang variable 283
ignoreeof variable 283
ignoring headers 217
always ignoring selected 251
when displaying messages 137
importing text files into messages 174, 323
in_reply_to variable 285
including messages 283, 300
incomplete messages
described 269
resuming 195
indent_str variable 283
indenting included messages
described 173
indent string for 283
initialization files
adding commands to 67
adding comments to 67
bypassing 280
conditional statements in 68
creating 66
creating with *saveopts* command 198
editing 67
-init option at startup 49
loading manually 64
order of loading 63
sourcing at startup 57
storing variable values in 227
suppressing loading of 49, 52
initialization variables
hdrs_only 280
is_gui 286
is_sending 287
is_shell 288
redirect 306
initializing Z-Mail 63
input variable 285
integrated user desktops 327
interactive spelling checker, specifying 288
interfaces
starting Z-Mail 20
internal variables, described 224
Internet 14
Internet domain addresses 29
interpose command 166
read receipt example 168
is_gui variable 286
is_sending variable 287
is_shell variable 288
ispeller variable 288

J

jobs command 169

K

keepsave variable 288
key sequences in macros, described 102
known_hosts variable 288

L

launching applications 326

library, location of Z-Mail program 51
license
obtaining a network 60
registering your Z-Mail local 55
lines
number displayed 268
specifying width of 316
listing UNIX files 171
Lite 5
lite.hlp file 23
literal-next character
backslash as 26
in macros 103
logfile variable 289
logging messages
by user 305
entire message 304
headers only 289
logins, alternate
removing from outgoing mail 119
lookup_max variable 290
lookup_sep variable 290
loops - see *each* and *foreach*
lpr command 170
ls command 171

M

m command - see *mail*
macros
command-line 106
creating 175
deleting 108
disabled by *-echo* option 102
errors in 104
expanding 102
expansions 102
invoking tilde-escapes with 107
key sequences 102
listing 175
map command 102, 175
mixed mode 107
prompting users from 105
recursive properties 103
specify control characters in 104
mail command 171
mail transport agent
described 14
passing *From* and *Date* headers to 53, 299
passing verbose option to 60
sending files directly to 43
sending message to 307
verbose option 313
mail user agent 14
mail_icon variable 290
mailing lists - see aliases
MAILRC environmental variable 63
Main Mailbox
described 18
filename of (*mbox*) 292
map command 102, 106, 175
mark command 175
mbox variable 292
menubars, disabling and hiding 297
merge command 176
merging folders 176
message information variables 229
message lists 73
as input to current command 285
from most recent command 298
parsing 177
piping to UNIX commands 187
specifying 24
using metacharacters to specify 24
message summaries
hiding messages in 160
number in character interface 307
number in GUI interface 307
redrawing with *redraw* 191
message templates, directories for 310
message window, specifying display of 292
message_panes variable 292
messages
appending fortunes to 278
assigning priorities 175
attachments 174
clearing temporary priorities 218
copying to files 133
creating headers 184
deleting 134
displaying 137
displaying in command-line 75

displaying next 185
displaying previous 189
displaying top lines of 212
editing 140
forwarding 173
importing text files 174
incomplete 269
indenting included 173, 283
merging 176
overriding signatures 174
paging in character mode 298
pin-up 187
printing 170
replying to all recipients 194
replying to sender 193
resuming incomplete 195
saving 196
searching 199
sending draft files as 174
sending in command-line 76
sorting 208
status flags 144, 158
suspending composed 169
undeleting 214
unmarking 218
unpreserving 219
using templates to create 174
metacharacters 26
filename pattern matching using 27
specifying message lists with 24
text pattern matching using 26
metoo variable 294
MH folders 18
mil_time variable 294
military time 294
MIME 15
MMDF folders 17, 18
modifiers, variable 228
motif.hlp file 22
mpcmd 131
msg_list command 177
msg_win variable 295
MTA - see mail transport agent
MUA - see mail user agent
multivalue variables, how to test fields of 232
Mush 159
.mushrc file 63
my_hdr command 184

N

n command - see *next*
name, setting your real 304
negation in message lists 24
Network License Server – see *NLS*
new mail
checking for 59
filtering 143
waiting for 123
new mail, time between checks for 311
newline variable 295
newmail_icon variable 296
next command 185
NLS 39, 60
no_expand variable 296
no_hdrs variable 297
no_menubars variables 297
nonobang variable 297
nosave variable 297

O

O'Reilly & Associates 31
open command 186
openfolders variable 298
openlook.hlp file 23
output variable 298

P

p command - see *print* and *display*
page command 186
pager variable 186, 298
pager, lines displayed by 268
paging messages 298
parentheses as variable notation 231
parsing
arguments 141
message lists 177
PATH statement 330
pattern matching
in filenames 27

in text 26
percent sign
in addresses 29
in information variables 229
period
as text metacharacter 26
in message lists 24
pick command - see *search*
picky_mta variable 299
pinup command 187
pipe command 187
pipelines
command, described 88
UNIX commands in 89
piping
message lists to UNIX commands 187
only message text 188
plus character
as filename metacharacter 27
POP, variables relating to 299
pop_options variable 299
pop_timeout variable 299
positional parameters – see function variables 318
post_indent_str variable 300
postmaster, reading mail for 59
PostScript files 328
pre command - see *preserve*
pre_indent_str variable 300
preserve command 189
previous command 189
print command - see *display*
print program, specifying name of 302
print_cmd variable 302
printer variable 301
printer, setting the 301
printing messages
in command-line interface 83
specifying print program 302
using *lpr* command 170
print-related variables 238
priorities
assigning 175
removing 176
process id of Z-Mail 317
program library, location of Z-Mail 51
programming-related variables 245
prompt
customizing your command-line 85
prompt variable 303
prompt, customizing your command-line 303
prompting
with a question 120
within macros 105
pwd command 190

Q

q command - see *quit*
question mark
as filename metacharacter 27
as help command 22
question, asking a 120
quiet variable 303
quit command 190
quitting Z-Mail 190

R

r command - see reply
R command - see *replyall*
read command 191
read receipt, example 168
Read Window, size of 269
read-only folders
opening at startup 53
opening with *folder* command 146
real name, setting your 304
realname variable 304
reciept, automatic return after a message is read 168
record variable 304
record_users variable 305
recording messages
by user 305
entire messages 304
headers only 289
recursive functions 124, 306
recursive variable 306
redirect variable 306
redraw command 191
registering users 55

remove command 192
removing folders 196
rename command 193
reply command 193
reply_to_hdr variable 306
replyall command 194
replying
    return paths 306
    routing replies 288
    to all recipients 194
    to sender 193
    to yourself accidently 294
    with inclusion 255
replying to messages
    in command-line interface 76
replysender command - see replying
resume command 195
resuming suspended messages 195
retain command 195
return command 196
return paths on replies 306
RFC 1327 118
RFC822 14
RFC822 addresses 29
Rmail folders 18
rmfolder command 196
root 129
root directory, changing 129
routing messages
    automatically 256
    avoiding unnecessary hosts 270
    via known hosts 288
routing-related variables 242

S

s command - see *save*
save command
    described 196
    differences from *write* and *copy* 198
save_empty variable 307
saveit function 156
saveopts command 198
saving initialization files 198
saving messages 196
    deleting after saving 288
    in command-line interface 83
saving messages to files 221
screen size
    setting in character mode 307
    setting in GUI mode 307
screen variable 307
screen_win variable 307
scripts
    terminating 196
    using variables in 228
    writing functions in GUI mode 326
search command 199
    described 199
    linking multiple 89
searching messages 199
    by date 201
    in command-line interface 84
semicolon, as command terminator 89
sending messages
    in command-line mode 76
    with *mail* command 171
sendmail variable 307
send-only mode 10
    attaching files in 41
    defined 20
    displaying headers in 48
    sending carbon copies in 42
    sending drafts in 43
    specifying subject in 58
    starting 34
services (*/etc/services*) file 61
set command 202
setting variables 202
sh command 205, 325
shell
    invoking UNIX 205
    using to launch commands 325
shift command 206
show_deleted variable 307
signal handling
    with *trap* 213
signature-related variables 243
signatures
    adding with tilde-escapes 95
    alternate 260
    automatic 259

described 15
local and remote 262
overriding 174
signing messages automatically 259
sign-on screen, suppressing 303
sort command 208
sort variable 308
sorting addresses 250
sorting messages
at startup 308
in command-line interface 84
with *sort* command 208
source command 210
sourcing initialization files at startup 57
spelling checker, using interactive 288
spool file - See System Folder 189
squeeze variable 308
starting Z-Mail 34
in command-line mode 34
in GUI mode 34
in Lite mode 34
in send-only mode 34
status variable 309
stop command 211
stty command 211
subject
prompting for a 252
setting with *-s* option 173
suppressing headers - see ignoring headers
suspended messages
listing with *jobs* command 169
resuming 195
suspending Z-Mail 211
System Folder
described 17
holding messages in 281
opening an empty 56
preserving messages in 189
specifying location of 51
starting up with other than 46
System V 94
system.zmailrc file 63, 199

T

:t variable modifier 228
t 129
t command - see *type* and *display*
tail of path variable modifier 228
task_meter command 211
templates
directory for message 310
message 174
using from the *zmail* command 58
templates variable 310
temporary files
directory for 311
testing multivalue variable fields 232
thisfolder variable 311
thwart command 212
tilde character
as command prefix 91
as filename metacharacter 27
tilde-escape commands
aborting messages with 99
adding signatures with 95
attaching files with 97
editing text with 93
including text files with 95
invoking with macros 107
list of 91–101
modifying headers with 91
referring to variables with 101
running commands with 100
starting text editor with 93
storing messages with 98
suspending messages with 99
timeout
checking for new mail 59
in macro keystrokes 103
timeout variable 311
tmpdir variable 311
top command 212
toplines variable 312
transport-related variables 242
trap command 213
twenty-four hour clock 294
type command - see *display*
TYPE statement 332

U

unalias command 213
unbutton command 214
uncmd command 214
undelete command 214
undeleting messages 214
undigest command 215
unfilter command 216
unfunction command 216
un_hdr command 216
unhide command 217
uniconify command 217
unignore command 217
UNIX commands
    entering 325
    invoking shell for 205
    pipelining 89
    piping message lists to 187
    piping messages to 187
    running 312
UNIX *mail* substitute, Z-Mail as a 10
unix variable 312
unmark command 218
unmarking messages 218
unpreserve command 219
unset command 219
update command 220
updating folders
    before hangup or shutdown 280
    with *quit* command 190
    with *update command* 220
user interfaces, Z-Mail's 20
UUCP addresses 30, 87, 119, 120, 185, 256, 257, 271, 289, 297

V

V7 folders 17
variables
    addressing-related 242
    command-line-related 245
    compose-related 239
    configuration-related 233
    default value sources 226
    directory-related 237
    display-related 235
    editor-related 238
    environment 228
    example of user-defined 204
    expanding 141
    file-related 237
    folder information 231
    folder-related 236
    function 224
    GUI-related 243
    header-related 234
    help-related 233
    initialization files and 227
    internal, described 224
    list of Z-Mail 248–319
    message information 229
    modifiers 228
    multivalue 232
    print-related 238
    programming-related 245
    routing-related 242
    setting 202, 225
    signature-related 243
    storing values of 227
    summary of Z-Mail 233–247
    system 224
    transport-related 242
    types of 224
    types of values for 224
    UNIX system-related 247
    unsetting values of 219
    user-defined 224
    using in scripts 228
variables file 23
verbose option, passing to MTA 60
verbose variable 313
verify variable 313
version command 221
version number, displaying Z-Mail 221, 314
version variable 314
visual editor 314
    invoking with tilde-escape 93
visual variable 314

W

w command - see *write*
warning variable 314
warnings
    displaying 314
    suppressing 303
window
    closing Z-Mail 161
    shelling a new 315
window_shell variable 315
Wingz, integrating Z-Mail and 327
working directory
    changing 129
    changing your 129
    displaying name of 190
wrap variable 316
wrapcolumn variable 316
write command
    described 221
    differences from *save* and *copy* 198
writing messages to files 221

X

x command - see *exit*
X Window System 5, 267, 280, 290, 296
X.400 addresses 118
xit command - see *exit*

Z

z command - see headers
z+ and *z*- commands 159
ZCNLSERV environment variable 60
–zcnlserv option 60
zmail command
    as substitute for UNIX *mail* 10
    option summary 36–39
    reference 40–60
Z-Mail Installation Guide 7, 55
Z-Mail User's Guide 7
zmail.small program 34, 72
.zmailrc file 63, 199, 210
ZMAILRC environment variable 63
Z-Script
    command reference 117–221
    command summary 110–116
    debugging 42, 133
    described 6