1 User's Guide to Symbolics Computers

symbolics

1 User's Guide to Symbolics Computers

symbolics™

Cambridge, Massachusetts

User's Guide to Symbolics Computers

# 999017

July 1986

This document corresponds to Genera 7.0 and later releases.

Symbolics, Inc.
4 New England Tech Center
555 Virginia Road
Concord, MA 01742

Text written and produced on Symbolics 3600-family computers by the Documentation Group of Symbolics, Inc.
Text masters produced on Symbolics 3600-family computers and printed on Symbolics LGP2 Laser Graphics Printers.
Cover Design: Schafer/LaCasse
Printer: CSA Press
Printed in the United States of America.

Printing year and number: 88 87 86 9 8 7 6 5 4 3 2 1

Table of Contents

List of Figures

1. Overview of Symbolics Computers

1.1 Documentation Notation Conventions

1.1.1 Notation Conventions Quick Reference

Modifier Key Conventions

Modifier keys are designed to be held down while pressing other keys. They do not themselves transmit characters. A combined keystroke like `META-X` is pronounced "meta x" and written as `m-X`. This notation means that you press the `META` key and, while holding it down, press the `X` key.

Modifier keys are abbreviated as follows:

Key	*Abbreviation*
CONTROL	c-
META	m-
SUPER	s-
HYPER	h-
SHIFT	sh-
SYMBOL	sy-

Modifier keys can be used in combination, as well as singly. For example, the notation `c-m-Y` indicates that you should hold down both the `CONTROL` and the `META` keys while pressing `Y`.

Modifier keys can also be used, both singly and in combination, to modify mouse commands. For example, the notation `sh-`(L) means hold down the `SHIFT` key while clicking left on the mouse and `c-m-`(M) means hold down `CONTROL` and `META` while clicking middle.

The keys with white lettering (like `X` or `SELECT`) all transmit characters. Combinations of these keys should be pressed in sequence, one after the other (for example, `SELECT L`). This notation means that you press the `SELECT` key, release it, and then press the `L` key.

`LOCAL` is an exception to this rule. Despite its white lettering, you must hold it down while pressing another key, or it has no effect. For example, to brighten the image on your monitor, you would hold down `LOCAL` while pressing `B`.

Documentation Conventions

This documentation uses the following notation conventions:

Appearance	*Representing*
cond, zl:hostat	Printed representation of Lisp objects in running text.
`RETURN`, `ABORT`, `c-F`	Keyboard keys.
`SPACE`	Space bar.
`login`	Literal typein.
`(make-symbol "foo")`	Lisp code examples.
(**function-name** *arg1* &optional *arg2*)	Syntax description of the invocation of **function-name**.
arg1	Argument to the function **function-name**, usually expressed as a word that reflects the type of argument (for example, *string*).
&optional	Introduces optional argument(s)
Show File, Start	Command Processor command names and command names in Zmacs, Zmail, and the front-end processor (FEP) appear with the initial letter of each word capitalized.
Insert File (`m-X`)	Extended command names in Zmacs and Zmail. Use `m-X` to invoke one.
[Map Over]	Menu items.
(L), (R2)	Mouse clicks: L=left, L2=`sh`-left, M=middle, M2=`sh`-middle, R=right, R2=`sh`-right. (`sh`-left means that you press the `SHIFT` key while holding down the left mouse button. You can achieve the same result by clicking the button quickly twice.)

Mouse Command Conventions

The following conventions are used to represent mouse actions:

1. Square brackets delimit a menu item.

2. Slashes (/) separate the members of a compound mouse command.

3. The standard clicking pattern is as follows:

   - For a single menu item, always click left. For example, the following two commands are identical:

     [Previous]
     [Previous (L)]

- For a compound command, always click right on each menu item (to display a submenu) except the last, where you click left (to cause an action to be performed). For example, the following two compound commands are equivalent:

  [Map Over / Move / Hardcopy]
  [Map Over (R) / Move (R) / Hardcopy (L)]

4. When a command does not follow the standard clicking order, the notation for the command shows explicitly which button to click. For example:

   [Map Over / Move (M)]
   [Previous (R)]

1.2 Introduction to the Symbolics 3600 Family of Computers

1.2.1 Introduction

The Symbolics family of advanced symbolic processing machines covers a full range of symbolic processing power and functionality. The unique design of Symbolics machines allows them to implement LISP and other symbolic programming languages with both speed and efficiency. The machines are faster and more efficient than other language specific machines and are superior to conventional computers for applications ranging from artificial intelligence, CAD/CAM, high resolution graphics, and expert system research and development as well as many general-purpose applications. The power, speed, and flexibility of Symbolics processing machines result from optimizing the hardware design to match the the software environment. Some of the special architecture features include :

Tagged architecture
Multiple caches
Hardware stack management
Pipelined instruction cycles
Parallel processing
Hardware assisted garbage collected
Fully ECC'ed system memory

1.2.2 Packaging

Members of the Symbolics family of advanced symbolic processing machines are single-user machines. The larger machines can be powerful stand-alone work stations or file or knowledge servers in networks, while the smaller versions are suitable as delivery vehicles for previously developed applications. All of the larger machines and some of the smaller machines are packaged as single-cabinet main units plus a console. Many of the delivery vehicles are complete in one desktop unit. Main units for the larger machines are desk-side size (30-inch and 55-inch cabinets) and the delivery vehicle machines fit under the average desk. The console for all models is a desktop unit with a full size black and white CRT and an extended keyboard.

1.2.3 Console

The high resolution, fully bit-mapped, black and white screen, the keyboard, and the mouse together form the console.

The three button mouse is a graphic input device with an on screen pointer and one or more screen text lines devoted to the mouse status. The mouse is used to point to and select menu items, to mark regions of the screen, and in some applications, to draw screen graphics.

The extended keyboard has eighty-eight keys divided into three sections: the typewriter section, the modify section, and the function section. The standard typewriter section of the keyboard contains the character keys. These keys are the primary user interface the system. The action of any of the character keys is changed by simultaneously holding down any of the modifying keys and the character key. The third section of the keyboard contains the function keys. Typing a function before typing some key combinations modifies the action of the other keys.

The high resolution black and white screen can be replaced with any of a wide selection of R-G-B color screens suited to particular applications.

1.2.4 Main Unit

The main unit of a Symbolics machine contains all of the electronics except the audio and video circuits of the console. The major parts of the main unit are the central processor and caches (CPU), the optional Floating Point Unit (FPU), the main memory, the Input/Output controller, and the Front End Processor (FEP).

1.2.5 CPU

The proprietary central processor unit is the heart of all Symbolics machines. Features that contribute to its power include : tagged architecture, multiple caches, hardware stack pointers, pipelined instruction cycles, and parallel processing.

Tagged architecture allows run-time data checking to catch invalid operations before they occur. Data type checking is performed in parallel with instruction execution thus eliminating the need for extra microinstructions and other software overhead.

The multiple caches used in the processor provide high speed access to the most current data and instructions, provide fast translation of virtual addresses, and allow efficient garbage collection.

Hardware stack pointers support high speed access to the stack and eliminate the need to execute microinstructions for managing the stack. Special hardware registers maintain top-of-stack and other stack addresses at all times to allow speedy access.

Symbolics machines that have the Enhanced Performance Option (EPO), perform three stage pipelined instruction cycles. An instruction fetched (first stage) from the instruction cache is dispatched (second stage) to the microsequencer. In the execute stage (third stage), the Instruction Fetch Unit supplies any necessary immediate argument to the processor.

The processor supports garbage collecting, run-time data-type checking, and instruction fetching, decoding, and executing in parallel.

1.2.6 Caches

Symbolics processing machines include multiple caches. Major caches are the Stack cache, the Memory Map cache, and the Instruction cache.

Symbolics processing machines are stack oriented machines with no general registers. This means that instructions are executed out of the control stack. The high-speed stack cache contains the top portion of the control stack and several pages surrounding the current stack pointer. These pages are most likely to contain the next referenced data object.

The Memory Map cache is a high speed 8K RAM located within the processor (CPU). It cross references the virtual page number and the physical page number.

The Instruction cache is part of the optional EPO. It is a 1 Kword cache that stores 2048 instructions loaded from the prefetch part of the Instruction Fetch Unit.

1.2.7 FPU

The Floating Point Unit is implemented in with a combination of NMOS VLSI and Schottky TTL technology. It is compatible with IEEE standard 754 for binary floating-point arithmetic. Because it works in parallel with the CPU, there is never a wait state while transferring a numerical operand to the FPU.

1.2.8 Memory

The memory word on Symbolics machines includes at least 36 bits for data plus Error Correction Code bits. The memory controller automatically corrects single-bit errors and detects and reports double-bit errors.

The physical (main) memory can be as large as 28 MB and the full-paging virtual memory is a maximum of 1 GBytes. The memory transfer rate can be as high as 5 Mwords /s. Some virtual memory is main memory and some resides on disk. The hardware automatically swaps between main memory and the pages on disk. A virtual address translates into a physical address through a hierarchy of mapping tables, some of which are cached in high-speed memory on the processor.

Disk storage is on either fixed Winchester or removable disks with capacities of up to 474 MB. The disk storage is expandable by adding expansion cabinets to hold eight 474 MB disk drives increasing the total capacity to the maximum 3.8 GBytes.

1.2.9 Input/Output

Symbolics processing machines support high and low speed I/O. Low speed devices such as the mouse, keyboard, cartridge tape, and serial lines are connected through the FEP.

One of the serial lines may be run synchronously and all of them can operate asynchronously. There is always one serial line from the console. The transmission rate of all serial lines is programmable up to 19.2 K baud.

Winchester disk drives are the mass storage medium for files and virtual memory paging , the disk controller hardware is linked through the system memory bus to the processor for high speed I/O.

1.2.10 FEP

The Front End Processor (FEP) is based on the MC68000 microprocessor chip which some computer manufactures use as the Central Processor Unit for their general purpose computers. Symbolics machines restrict the MC68000 to reducing the workload of the proprietary central processor. The FEP takes care of the initial bootstrapping of the machine and offloads low-speed peripheral I/O from the CPU.

1.3 Introduction to Genera

The Symbolics software environment that runs on the Symbolics Family of Computers is called Genera.

1.3.1 The Console

The devices that are used to talk to Genera are collectively referred to as the *console*. These include one or more bit-raster displays, a specially extended keyboard, and pointing device called a *mouse*.

1.3.2 The Screen

The screen always contains one or more windows. Regardless of which windows are displayed, the screen always contains some information displays, including a *mouse documentation line* and a *status line*. These information displays are helpful in determining whether Genera is operating normally or needs intervention. See the section "Recovering From Errors and Stuck States", page 143.

1.3.3 Mouse Documentation Line

The mouse documentation lines contain information about what different mouse clicks mean. As you move the mouse across different mouse-sensitive areas of the screen, the mouse documentation lines change to reflect the changing commands available.

When no documentation appears, it does not necessarily mean that the mouse clicks are undefined. Not all programs have provided material for the mouse documentation line. When the mouse documentation lines are blank at "top level" in a window, the mouse usually offers some standard commands. Clicking `Mouse-Left` selects a window. Clicking `Mouse-Right` often brings up a menu specific to the application. Clicking `sh-Mouse-Right` brings up the System menu.

The mouse documentation lines are normally displayed enclosed in a box in reverse video. Pressing `FUNCTION m-C` complements the video state of the mouse documentation line.

1.3.4 Status Line

The status line is the line of text at the bottom of the screen. It contains the following information:

- Date and time
- Login name
- Current package
- Process state
- Run bars
- Other context-dependent information, such as
  - Console idle time
  - Network service indicators

1.3.5 Process State

The process state refers to the processes associated with the selected window. See the section "Selecting and Creating Windows", page 13. The following list shows some common states:

State	*Meaning*
`Mouse Out`	Waiting for the mouse process to notice a change of windows.
`Net In`	Waiting for data from another machine on the network.
`Net Out`	Waiting to send data to another machine on the network.
`Open`	Waiting to open a file on another machine on the network.
`Run`	Process is running.
`User Input`	Waiting for input from keyboard or mouse.

1.3.6 Run Bars

The run bars are thin horizontal lines near the process state in the status line. A description of each one follows:

GC bar (under the package name)
: Left half is visible when the scavenger is looking for references to objects that are candidates to become garbage. Right half is visible when the transporter is copying an object.

Disk bar
: Visible when the processor is waiting for the disk, typically because of paging. Nonpaging disk I/O usually waits via **process-wait**, in which case this bar does not appear.

Run bar (under the run/wait state)
: Visible when a process is running and not waiting for the disk. Not visible when the scheduler is looking for something to do.

Disk-save bar
: Visible when **zl:disk-save** is reading from the disk; **zl:disk-save** alternatively reads and writes large batches of pages. The alternating state of this bar tells you that **zl:disk-save** is working while you wait for it.

1.3.7 The Keyboard

There are 88 keys on the keyboard. The keyboard has unlimited rollover, meaning that a keystroke is sensed when the key is pressed, no matter what other keys are depressed at the time.

The keys are divided into three groups: special function keys, character keys, and modifier keys. Special function keys and character keys transmit something. They have white labels on the tops and are typed in sequence. Modifier keys are intended to be held down while a function or character key is typed, to alter the effect of the key. They have dark labels on the tops.

Function Keys

FUNCTION, ESCAPE, REFRESH, CLEAR INPUT, SUSPEND, RESUME, ABORT, NETWORK, HELP, TAB, BACKSPACE, PAGE, COMPLETE, SELECT, RUBOUT, RETURN, LINE, END and SCROLL

Character Keys

a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 : - = ' \ | ( ) ; ' , . / and the space bar.

Modifier Keys

LOCAL, CAPS LOCK, SYMBOL, SHIFT, REPEAT, MODE LOCK, HYPER, SUPER, META, and CONTROL

The following keys are reserved for use by the user (for example, for custom editor commands or keyboard macros):

CIRCLE
SQUARE
TRIANGLE
HYPER

1.3.7.1 Auto-repeat

You can have keys repeat if they are held down. This feature is disabled by default, but you can enable it by setting **si:*kbd-auto-repeat-enabled-p*** to t.

```
(setf si:*kbd-auto-repeat-enabled-p* t)
```

The speed of repetition is controlled by **si:*kbd-repetition-interval***. See the variable **si:*kbd-repetition-interval***, page 223.

You can exempt certain keys from auto-repetition using the function **si:set-auto-repeat-p**. For example, to make SQUARE one of the keys that do not auto-repeat, you would type:

```
(si:set-auto-repeat-p #\Square nil)
```

See the function **si:set-auto-repeat-p**, page 97.

1.3.8 The Mouse

The mouse is a pointing device that can be moved around on a flat surface. These motions are sensed by Genera, which usually responds by moving a cursor around on the screen in a corresponding manner. The shape of the cursor varies, depending on context.

There are three buttons on the mouse, called left, middle, and right. They are used to specify operations to be performed. Typically you point at something with the mouse and specify an operation by clicking the mouse buttons. "Shift clicks", indicated by sh-, are conventionally distinguished from single clicks. Holding down the SHIFT key while clicking a button is the same as clicking that button twice quickly. In any specific context, there are up to six operations that can be performed with the mouse, invoked by Left, sh-Left, Middle, sh-Middle, Right, and sh-Right clicks. Some of these operations are local to particular programs such as the editor, and some are defined more widely across the system.

Typically the operations available by clicking the mouse buttons are listed at the bottom of the screen. This display, called the mouse documentation line, changes as you move the mouse around or run different programs.

Sometimes holding a mouse button down continuously for a period of time may also be defined to perform some operation, for example, drawing a curve on the screen. This will be indicated by the word "Hold". For example, "Middle Hold" means to click the middle mouse button down and hold it down, releasing it only when the operation is complete. "sh-Left Hold" means hold down the SHIFT key and click left, then release the SHIFT key but hold the left button down until the operation is complete.

1.3.9 The Mouse and Menus

1.3.9.1 Mouse-sensitivity

Parts of the screen can be *mouse-sensitive;* that is, clicking one of the mouse buttons on these parts causes some action to occur. When the mouse cursor moves over a portion of the screen that is mouse-sensitive, an outline box appears around the item. Clicking on the boxed item in the manner specified in the mouse documentation line causes the desired action to occur.

1.3.9.2 Scrolling

Many windows in the system respond to scrolling commands. Bump the mouse against the left side of the pane until a scroll bar and double-headed pointer appear.

The scroll bar, by its size and placement on the left side of the pane, indicates the percentage of the pane or buffer contents that is currently visible on the screen. For example, a very short scroll bar at the bottom of the pane indicates that you are seeing the last part and only a small percentage of the contents of that pane or buffer. A very long scroll bar in the center of the pane indicates that you are seeing a large proportion and are approximately half-way through the pane's entire contents.

To scroll using the scroll bar and the double-headed pointer, use one of the following mouse buttons:

R	Moves the line currently at the top of the screen to the position indicated by the pointer.
M	Displays the percentage of the pane contents that approximately corresponds to the position indicated by the pointer.
L	Moves the line indicated by the pointer to the top of the screen.
L2	Moves the line indicated by the pointer to the bottom of the screen.

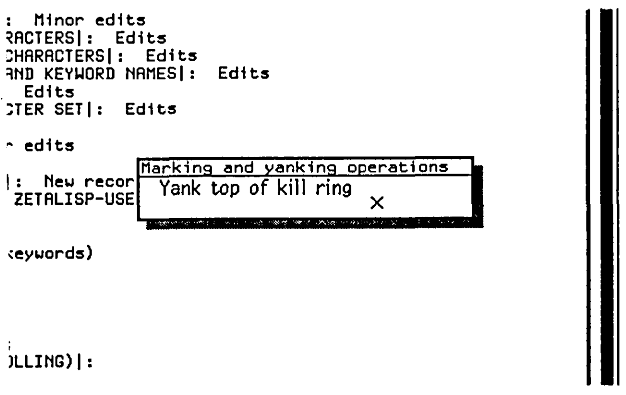
1.3.9.3 Menus

One common application of a mouse button is to call up a *menu* of options, containing mouse-sensitive choices. Menus are lists of mouse-sensitive choices, surrounded by a border. They normally appear in the part of the screen where the mouse cursor was positioned when you clicked the button.

The 3600 has several styles of menus, including the following common ones:

- Momentary menu

  Each item is a possible choice. Positioning the mouse cursor over an item and then clicking the appropriate button makes the choice. The System Menu is a momentary menu.

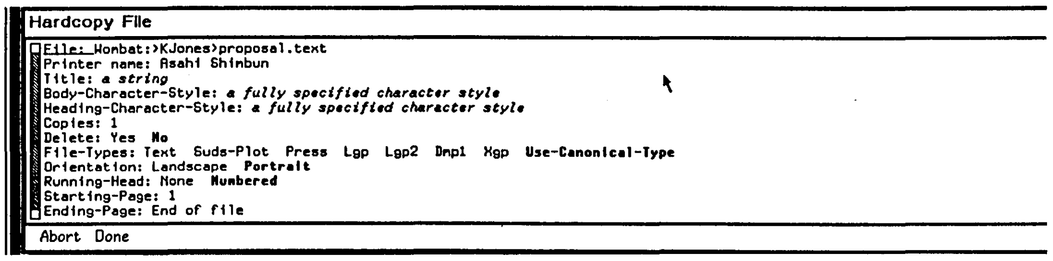


```
:  Minor edits
RACTERS|:  Edits
CHARACTERS|:  Edits
AND KEYWORD NAMES|:  Edits
  Edits
CTER SET|:  Edits

- edits

|:  New recor
ZETALISP-USE

keywords)


DLLING)|:
```

Figure 1.   A Momentary Menu

- Choose-variable-values menu

  Each line presents one or more possible values of a particular variable. The



```
Hardcopy File
File: Wombat:>KJones>proposal.text
Printer name: Asahi Shimbun
Title: a string
Body-Character-Style: a fully specified character style
Heading-Character-Style: a fully specified character style
Copies: 1
Delete: Yes  No
File-Types: Text  Suds-Plot  Press  Lgp  Lgp2  Dmp1  Xgp  Use-Canonical-Type
Orientation: Landscape  Portrait
Running-Head: None  Numbered
Starting-Page: 1
Ending-Page: End of file
 Abort  Done
```

Figure 2. A Choose-variable-values Menu

Window Attributes Menu is a choose-variable-values menu. Each variable has a type that controls what values it can take on. The way in which the possible values are presented and the way in which you choose a value depend upon the type. Variables can have one of two types.

° A type with a small number of legal values. Each line in the menu presents the possible legal values of a particular parameter. The current value appears in bold face. Each of the values is mouse-sensitive. Clicking on a value selects it.

° A type with a large or infinite number of legal values. Each line in the menu presents only the current value of a particular parameter. A numerical value is of this type. To change a value, select the current value by clicking on it, type in a new value, and press RETURN. Rubbing out more characters than have been typed in restores the original value instead of changing it.

You exit menus in a variety of ways. For some menus, like the System Menu, making the choice causes the menu to disappear. Moving the mouse cursor off this kind of menu also causes the menu to disappear. Other menus have explicit commands, such as [Do It], [Exit], or [Abort], which you must click on to make the menu disappear. Other menus are displayed in the frame permanently, such as the Zmail Command Menu.

1.3.9.4 System Menu

The System Menu is a momentary menu that lists several choices for acting upon windows and calling programs (for example, Lisp Listener, Zmacs, or the Inspector). You can always call the system menu by clicking `sh-right` (or the right mouse button twice). Use the System Menu to do many things, among them:

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

For more information about the mouse and menus: See the section "Using the Window System". See the section "Window System Choice Facilities".
7B s. 83 7B s. 239

1.3.10 Selecting and Creating Windows

1.3.10.1 Introduction

All user interaction with the 3600, except with the FEP, occurs in *windows.* The screen always contains one or more windows. The window that you are interacting with is called the *selected window.* You select a window via the mouse, a menu, or a keyboard key. If the window is not already exposed, it appears on the screen. See *Introduction to Using the Window System* for more information about windows.

1.3.10.2 Default Windows

The 3600 has a default set of windows available, some of which are available via a System Menu and some via the `SELECT` key as well.

Use [Select] in the Windows column of the system menu to see a menu of currently available windows. Some default windows are

`Main Zmail Window`
`Lisp Listener 1`
`Edit:` *pathname*

1.3.10.3 Moving

On the 3600, you do not "leave" a window with an explicit terminating command; instead, you select a different window. You can return to the most recently used window by pressing `FUNCTION S`.

1.3.10.4 `SELECT` Key

This key is a prefix for a family of commands, generally used to select a window of a specified type, such as a Lisp Listener or Zmail. The current list is:

C	Converse
D	Document Examiner
E	Editor
F	File system maintenance
I	Inspector
L	Lisp
M	Zmail
N	Notifications
P	Peek
Q	Frame-Up
T	Terminal
X	Flavor Examiner

`SELECT c-` creates a new window of the specified type.

2. Starting up

This section provides information about how to start, cold boot, log in to, and log out of the 3600 family of machines. It assumes that the software is installed and your site has been configured. If you are not sure that this has been done, check with your site manager. The software must be installed and the site configured before you attempt to use the system. For information on installation and site configuration:

See the *Software Installation Guide*
See the section "Installation Procedures" in *Site Operations.*

2.1 Powering up

To power up and start using the your Symbolics Computer, use the following procedure:

1. If you have a 3600:

   a. Plug in the 3600. The front panel lights on the processor cabinet display "3600" when the machine is plugged in. If they are not lit, check that the main circuit breaker at the lower rear of the cabinet is turned on.

   b. Turn the key on the front panel to the vertical position, marked LOCAL.

   c. After the front panel lights display "Power up?", push the spring-loaded switch marked YES. The front panel lights then display "3600 on".

2. If you have a 3670 or 3640:

   a. Plug in the machine.

   b. Press the Power button on the front panel.

See the section "Cold Booting After Powering up".

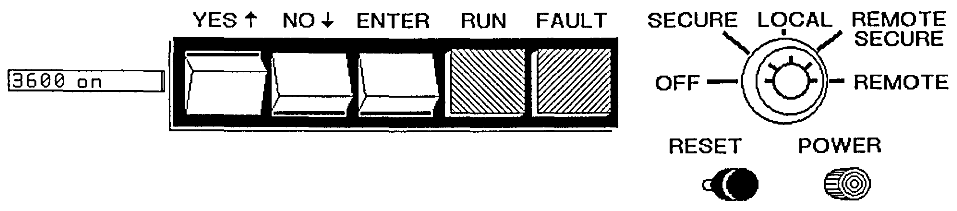


Figure 3. The Front Panel on a 3600

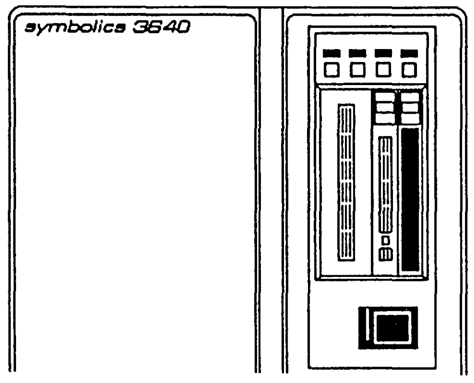


Figure 4. The Front Panel on a 3670 or 3640

2.2 Logging in

After cold booting, you are in a window named Dynamic Lisp Listener 1. You are now ready to log in. If your login name is KJones, you can log in in any of the following ways: (Note that the examples are given in upper and lower case, but the machine is not case sensitive. You can use all upper case, all lower case, or mixed case as you prefer.)

- To log into the default host machine, using your init file, type
  ```
  Login KJones
  ```
- To log into your machine, without your init file, type
  ```
  Login KJones :init file none
  ```
- To log into another machine "sc3", using your init file, type
  ```
  Login KJones :host sc3
  ```

If the host machine you log in to is a timesharing computer system, you must have a directory and account on that host machine.

For more information about logging in: See the section "Login Functions and Variables".

For more information about how to write init files: See the section "Customizing Genera", page 93.

2.3 Logging Out

1. Use a Lisp Listener by pressing `SELECT L`.

2. Log out by typing either the command `Logout` or the function `(logout)`.

   Wait until the Lisp Listener says that you have been logged out before you go to the next step.

3. Cold boot the machine.

   This step is optional. It is not necessary to cold boot if the machine has been used only a short while and if no major changes to the machine state have been made. If the machine has been used for several hours and many files have been loaded or read into it, we recommend that the machine be cold-booted.

   Cold booting frees up virtual memory and puts the machine in a fresh state. In this way your customizations do not affect the next user's environment.

Note: You need not turn the machine off each night; however, it does not hurt the machine to do so.

2.4 Powering Down

To power down your machine:

1. First logout by giving the command:

   ```
   Logout
   ```

2. Next, halt the machine by giving the command:

   ```
   Halt Machine
   ```

3. If you have a 3600, turn the key on the front panel to the off position.

   If you have a 3670 or 3640, give the shutdown command to the FEP:

   ```
   Shutdown
   ```

   or press the power button on the front panel.

Note: it is not necessary to turn off the circuit breaker on the back of the machine unless you are planning to unplug the machine and move it.

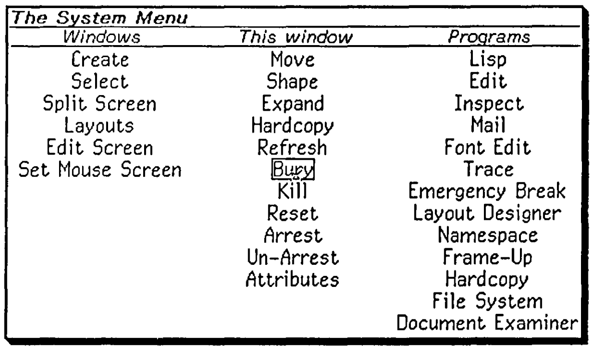
2.5 Getting Acquainted with Genera

2.5.1 Using the System Menu

1. Press down the `SHIFT` key and click the right mouse button. (This is usually denoted in documentation as `sh-Mouse-right`.) You should see the System menu pop up on the screen as in Figure 4. Notice that the mouse cursor has become an x and that there is a box around the word `bury`. The word `bury` is in the column titled "This Window" and its being in a box means that `bury` *mouse sensitive.* If you were to click a mouse button while the box is around `bury`, thereby *selecting* it, the current window ("This window") would be *buried* down at the bottom of the stack of active windows (that is, shuffled to the bottom as you might do to a stack of paper). The mouse documentation line at the bottom of the screen says:

   ```
   Bury the window that the mouse is over, beneath all other active windows.
   ```

   Move your mouse so that it is off this menu. The menu disappears. That is



The System Menu

Windows	This window	Programs
Create	Move	Lisp
Select	Shape	Edit
Split Screen	Expand	Inspect
Layouts	Hardcopy	Mail
Edit Screen	Refresh	Font Edit
Set Mouse Screen	Bury	Trace
	Kill	Emergency Break
	Reset	Layout Designer
	Arrest	Namespace
	Un-Arrest	Frame-Up
	Attributes	Hardcopy
		File System
		Document Examiner

Figure 5. The System Menu

the way *pop-up* menus work: they disappear when the mouse is moved off them.

Click `sh-Mouse-right` again. Notice that the System menu again pops up, but it has followed your mouse. A pop-up menu always pops up where your mouse is, because it can only remain on the screen when the mouse is inside its borders. Try moving the mouse and clicking `sh-Mouse-right` several more times. `sh-Mouse-right` summons the System menu in all contexts in Genera. This is important to remember.

2. Click `sh-Mouse-right` again and now take a more careful look at the information the System menu is providing.

The operations that the System menu provides to you are divided into three categories:

Windows
This Window
Programs

Windows refers to general window operations, as indicated by the items in that column (See Figure ~~4~~): 5

- `Create` - Create a new window.

- `Select` - Select one of the windows already established by Genera or created by you. Another pop-up menu appears, offering all the windows currently available in Genera.

- `Split Screen` - Divide your screen so that two windows are completely visible at the same time.

- `Layouts` - Operate on the geography of the screen display by restoring the previous state or saving the current state. You can do what Layouts does with Edit Screen and the Undo option.

- `Edit Screen` - Modify the geography of the screen.

- `Set Mouse Screen` - Set the screen the mouse is on if you have more than one screen (for example, a color console and a standard one).

This Window refers to operations of the current window, that is, the one that you were looking at and typing to when you clicked `sh-Mouse-right`.

- `Move` - Change the location of the current window on the screen.

- `Shape` - Modify the shape of the current window (square to rectangle, and so on).

- `Expand` - Make the current window larger by occupying unused space on the screen.

- `Hardcopy` - Send an image of the current window to a printer. This is the same as pressing `FUNCTION c-Q`. See the section "`FUNCTION` Key", page 216.

- `Refresh` - Redisplay the current window. This is the same as pressing `FUNCTION REFRESH`. See the section "`FUNCTION` Key", page 216.

- `Bury` - Shuffle the current window down to the bottom of the stack of windows, selecting the previous (next down in the stack) window as current.

- `Kill` - Remove the current window.

- `Reset` - Initialize the current window, that is, restart it in its initial state. This is useful if a window is in a confused state. It might cause loss of information, however. See the section "Recovering From Errors and Stuck States", page 143.

- `Arrest` - Halt any processes running in the current window.

- `Un-Arrest` - Release any processes in the current window, allowing them to continue.

- `Attributes` - Modify the attributes of the current window. This summons a menu of the attributes of the window and allows you to selectively change them. Move the mouse down vertically and click on `Attributes`. Notice that the menu that appears has two choices in its bottom margin: `Abort` and `Done`. These are for exiting from the menu. You can click on unhighlighted entries in the central portion of the menu (the choice section). Highlighted items are the current attributes. Clicking on an unhighlighted entry highlights it and selects it to go into effect, but nothing happens until you exit from the menu. Try clicking on several choices to see how it works. This is called an *Accept Variable Values* menu, or in some cases a *Choose Variable Values* menu. Now, since you do not really want to modify anything just now, click on `Abort`. The menu goes away.

Click `sh-Mouse-right` again to get the System menu back and look at the third column:

Programs refers to available preloaded programs, or activities, in Genera.

You can select an activity from this menu, or you can use the Select Activity command processor command. See the section "Select Activity Command", page 251. For now, just move your mouse off the menu.

3. Click `sh-Mouse-right` once more. This time, move the mouse to the left and position it over the *Windows* column.

   Move it to sit on `Create`.

   Click left on `Create`. Another pop-up menu appears containing a list of window types.

   Click on `Lisp`.

   The menu disappears and ah icon representing the upper left corner of a box appears where your mouse cursor was. Move this icon using the mouse to some convenient location on your screen. (See Figure 5). Press the left mouse button and notice that the left corner now sprouts two more sides and becomes a *rubber band* with four corners. Stretch it around by moving the mouse. Notice what happens if you pull it up to the left of the left corner that you just positioned. If you were to click left at this point, you would create a new window that is the full size of the screen.

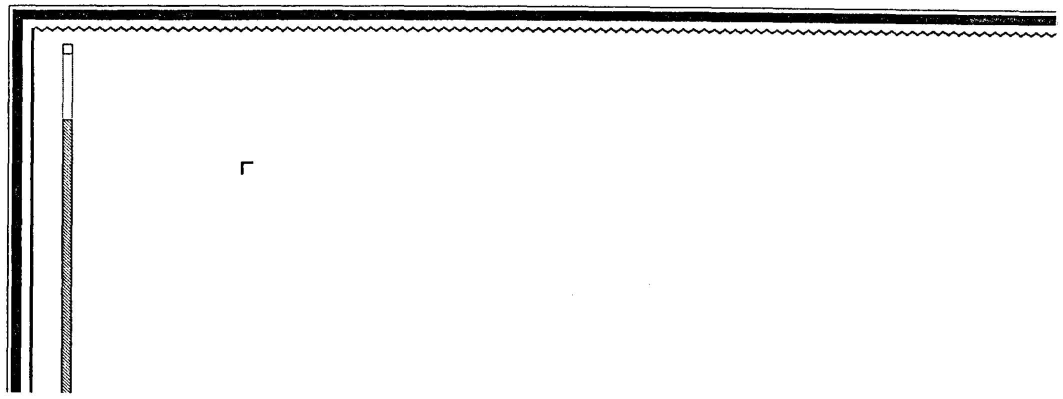


Figure 6. Positioning the Upper Left Corner of a New Window

Now, pull it down to make a reasonable sized window, at least ten inches across and at least 25 lines (about four inches) vertically. Click left if your mouse and the new window are not in the proximity of any other window edge. If the lower right hand corner of your new window is near the edge of another window, click right. Clicking right (as the mouse documentation line says) is *smart*. This means it tries to optimize the position of the edges of windows so that they do not overlap. Windows that overlap by very tiny amounts at the edge can confuse you as to which window is actually selected, especially if you choose not to use the graying option for non-selected windows. (See the section "Set Screen Options Command", page 258.)

You have just created a second Lisp Listener, and it is selected. That means that any typing you do at this point goes into this window.

2.5.2 Trying Out the Command Processor

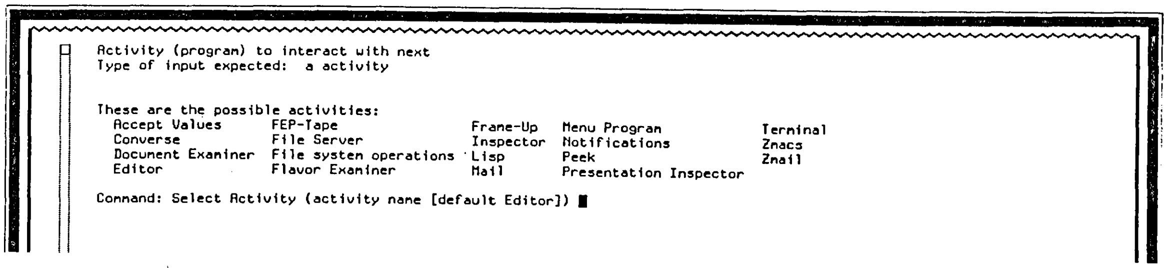
2.5.2.1 Typing Commands

Type `Select Activity SPACE Lisp RETURN`. You are now back in your original Lisp Listener. Type `Select Activity SPACE Lisp RETURN` again to get back to our newly created Lisp Listener. Select Activity *activity* will cycle through all the windows of a given activity. Get back to your original Lisp Listener, typing `Select Activity` if necessary. Notice that there is a key labelled `SELECT`. Press `SELECT` followed by `L`. You are now in your other Lisp Listener. `SELECT L` is the same as Select Activity Lisp. Press `SELECT` followed by `HELP` to see a display of the other single letter abbreviations for activities.

Select Activity is a command that takes one argument, an activity name. Some commands take no arguments. Some take more than one argument. Some commands also take optional arguments called *keywords* that control the behavior of the command.

2.5.2.2 Using the HELP Key

Type `Select Activity SPACE` then press the `HELP` key. You should see a display like that in Figure 6.



```
Activity (program) to interact with next
Type of input expected:  a activity

These are the possible activities:
  Accept Values      FEP-Tape                Frame-Up   Menu Program           Terminal
  Converse           File Server             Inspector  Notifications          Zmacs
  Document Examiner  File system operations  Lisp       Peek                   Zmail
  Editor             Flavor Examiner         Mail       Presentation Inspector

Command: Select Activity (activity name [default Editor]) ▮
```

Figure 7. Select Activity Command

Press `CLEAR INPUT`. The command goes away. `ABORT` would also cancel the command, but since you had not finished typing the *arguments* to the command (that is, the activity you wanted to select), there is nothing to abort so the "smaller hammer", just clearing the input you have typed, can be used.

2.5.2.3 Using Keywords Arguments

Type `Show Herald RETURN`. The Herald (the initial screen display announcing the system version loaded) is printed on your console. Show Herald is a command that takes no arguments.

Type `Show Herald` again and then press `SPACE`. The prompt `(keywords)` pops into your input line. Press `HELP`. You should see:

```
You are being asked to enter a keyword argument


These are the possible keyword arguments:
:Detailed            Whether to print version information in full detail
:Output Destination  Redirects the output of this command to the specified streams.

Command: Show Herald keywords
```

Figure 8. Show Herald and its Keywords

This time add `:detailed` before pressing `RETURN`. Now you should see additional information about the version of the FEP you are running with and other systems that are loaded in your world.

This demonstrates in a very simply way how keywords can affect the behavior of a command.

2.5.2.4 Some Useful Command Processor Commands

1. Type `Show Machine Configuration`. When this command takes no arguments, it displays the information about your particular machine. You can give Show Machine Configuration the name of another machine on your network as an argument and it will show you the information about that machine. This is an important command to know because if you have to call Symbolics Software Support you might be asked for information about your hardware that you can get only from this display. Of particular importance when you call Software Support is the Machine Serial Number, appearing in the third line of the display.

2. Type `Show Command Processor Status`. Again, this command takes no arguments because it is displays information about the setting of the Command Processor. Unless you have changed the Command Processor Mode or your prompt, you should see a display like this:

```
Command: Show Command Processor Status
 The command processor's current mode is
  Command Preferred: Interprets input starting with an alphabetic
                     character as commands; type an initial , to
                     force Lisp interpretation.

 The prompt string is "Command: ".

 The prompt strings for other modes are:
   Form Preferred:          ""
   Form Only:               ""
   Command Only:            "Command: "
```

This means that your Command Processor is in the default mode and any typing you do to it is assumed to be a Command Processor command unless it begins with a left parenthesis or a comma. Type

```
,*package* SPACE
```

This asks for the value of the Lisp variable ***package***. The value returned should match the package shown in the status line at the bottom of the screen.

You can change the mode of the Command Processor. See the section "Set Command Processor Command", page 254.

3. Type `Show FEP Directory`. This displays a list of all the files on the local disk(s) of your machine. The files that Genera is currently using (the *World* load from which Genera was booted, the paging files, and the microcode in use) are displayed in **boldface**. That is to remind you that you should not delete them.

   Type `Show FEP Directory` again and this time press `SPACE` three times. When it prompts for keywords, press `HELP`. Try Show FEP Directory with the :type keyword. After typing `:type` press `HELP` again and select a kind of file, perhaps `boot`. This displays only those files that are boot files, that is, files that contain a *script* of commands for booting a world. See the section "Cold Booting After Powering up".

2.5.2.5 Looking Back Over Your Output (Scrolling)

By now you have had to press `SPACE` at `**More**` breaks several times. Hold down the `META` key and press `SCROLL` (this is usually denoted `m-SCROLL`). You can look back over your interaction with Genera. To get back to the "end" of this output *history* you can press `SCROLL`, press `m-sh->`, or just start typing a new command. Without pressing anything, from where you are in the middle, try typing `Show`

`Namespace Object user` *your user name.* If your user name were KJones, you would see the window scroll forward to the new command line and then display:

```
View in namespace ACME:
USER KJONES
LISPM-NAME KJones
PERSONAL-NAME "Jones, Kingsley"
HOME-HOST ACME
MAIL-ADDRESS kjones ACME
LOGIN-NAME kjones VAX01
NICKNAME King
WORK-ADDRESS "Building 3-701"
WORK-PHONE 5891
BIRTHDAY "19 June"
PROJECT Database
SUPERVISOR "Finklestein"
USER-PROPERTY :USUAL-LOGIN-HOST wombat
```

For more details about scrolling windows: See the section "Scrolling with the Mouse", page 212.

Now you have tried a few command processor commands and it is time to show you how Semanticue and Dynamic Windows can speed up your work by cutting down on your typing. See the section "Getting Acquainted with Dynamic Windows", page 26.

2.5.3 Getting Acquainted with Dynamic Windows

2.5.3.1 Reusing Commands

1. Press `c-m-Y`.

   Press `m-Y`.

   Press `m-Y` again.

   Notice how each successive previous command processor command you typed is placed at the `Command:` prompt. Conveniently, you can reactivate any of these commands by pressing `RETURN` when the one you want appears. For now just press `CLEAR INPUT`.

2. Type `Show Documentation SPACE "Show Documentation Command" RETURN`. (The topic name should be inside quotation marks.)

3. Press `c-m-Y`. Press `m-RUBOUT` three times to erase "Show Documentation Command". Now type `"Reusing Commands" RETURN`. You can use Show

Documentation on any topic in the documentation set. Those topics whose documentation you have displayed in your Lisp Listener are also read into the Document Examiner. See the section "Using the Online Documentation System", page 43.

2.5.3.2 Using Your Output History

1. Type `Show Directory SPACE sys:examples;`. This is a directory of sample programs you can look at and run.

   Move your mouse slowly over the display of the directory. Notice that the individual files (or subdirectories) listed are mouse sensitive, that is, a box appears around them as the mouse passes over them.

2. Now type `Show File` and click the mouse left on one of the files in the display from Show Directory, perhaps on Teach-Zmacs-Info.text. The pathname is inserted in your new command line. Press `RETURN` to activate the command, or `CLEAR INPUT` if you do not want to see the contents of the file displayed. (This file, Teach-Zmacs-Info.text, is a good one to remember about when you are ready to learn about Genera's editor, Zmacs. It tells you how to run a tutorial that explains the editor.)

3. Scroll back, using `m-SCROLL` or `m-V`, over your output history so far. Select a command line you would like to reactivate. Move the mouse over it and when it becomes mouse sensitive, click left. The entire command line is *yanked* down to the current prompt. You can press `RETURN` to reactivate it, or `RUBOUT` or other editing commands to edit it. See the section "Editing Your Input", page 134.

4. Once again scroll back over your history and select a command to reactivate. Click `sh-Mouse-Left`, that is, hold down the shift key while you click the left mouse button. This time your selected command is not only yanked down to the current prompt but also is reactivated without your having to press `RETURN`.

5. When your history is long, scrolling back over it is tedious. Hold down the `SUPER` key and press `R` (`s-R`). Now type a string of characters, `Show` for instance. Notice that your cursor is moved back through to the history to the most recent occurrence of the word `show`. Press `s-R` again. You are moved back to the next most recent occurrence. If you move past the occurrence of `show` that you want, press `s-S` to search forward. Press `END` to terminate the search. You can now click `Mouse-Left` or `sh-Mouse-Left`.

6. Sometimes you know exactly what command you want to yank and do not

want to search for it. Hold down the CONTROL, META, and SHIFT keys and press Y (this is denoted c-m-sh-Y). You are prompted for a character string to use and the most recent command that contains that string is yanked directly. If the most recent command containing the string is not the one you want, press m-sh-Y and the previous most recent command is yanked instead. Successive uses of m-sh-Y go back farther and farther in your history locating commands containing the string.

Now that you know how to move back through your history it is time to learn some more ways to make use of it. See the section "Using the Mouse", page 28.

2.5.3.3 Using the Mouse

1. Move your mouse so it is just off the top of the screen. Look down at the *mouse documentation lines* just above the date and time at the bottom of the screen. Notice that it is blank. Move the mouse back down onto a blank part of the screen. Now the mouse documentation line says "To see other commands, press Shift, Control, Meta-Shift, or Super".

2. Move the mouse so that it is over a previous command. Now notice the mouse documentation line. It now says that Mouse-L is the command, meaning that clicking left yanks the command. You have already discovered this: See the section "Using Your Output History", page 27. Mouse-R says "Menu". Click right. A menu pops up with a list of operations that can be performed on the command line the mouse is over. Move your mouse off the menu to make it disappear.

3. Now position your mouse over the command line again. Press SHIFT. Notice that the documentation changes to indicate what operations can be performed by pressing SHIFT (sh-Mouse-) while clicking Left or Right. sh-Mouse-Left you have already tried: See the section "Using Your Output History", page 27. sh-Mouse-Right pops up the System menu: See the section "Using the System Menu", page 18.

4. Release SHIFT and press CONTROL. The operations offered by pressing CONTROL while clicking the mouse (c-Mouse-) are for marking regions or words for yanking or copying. c-Mouse-Left marks a region. Hold down the CONTROL key and the left mouse button while you move your mouse around. Notice that the text the mouse moves over is underlined. Release CONTROL key and the left mouse button. Now click c-Right. The *Marking and Yanking* menu pops up, offering you four things to do with the text you have marked. Usually you want to push the text on the *kill ring*, the place where text that has been deleted with one of the kill commands (c-K, c-W, and others) in any context (Lisp, Zmacs, or Zmail) is stored for recall with

c-Y. You can push the marked text on the kill ring by clicking on that choice in the menu. Do that now. Because pushing text on the kill ring is such a common operation, it can also be done by holding down SUPER and pressing W (s-W). Now press c-Y. The text you marked is yanked back at the current prompt. In this way you can yank arbitrary pieces of text for editing and turning into a new command. Since the kill ring is common to Lisp and Zmacs, this is a way to transfer something from your Lisp Listener to an editor buffer for editing or for saving in a file.

5. c-Mouse-Middle marks and yanks the word the mouse is over. Try pressing c-Mouse-Middle several times to yank arbitrary words from your output history. Press CLEAR INPUT.

6. Hold down both the CONTROL key and the SHIFT. c-sh-Mouse-Middle allows you to mark (without yanking) words.

7. Holding down the META and SHIFT keys and clicking right pops up the *Window Operation* menu. This menu offers operations to perform on the current window, much like the "This Window" column in the System menu. See the section "Using the System Menu", page 18.

2.5.3.4 Using Menus

Using the Mouse and the Keyboard on Menus

Type Set Window Options and press RETURN. An Accept Variable Values menu is displayed. The items in the menu are the various options that you can set to customize your Lisp Listener. The highlighted (boldface) items are the current settings. Move your mouse over the menu and notice that items become mouse sensitive. You can click on an unhighlighted choice in a list and it will become highlighted. You can click left on a displayed value to replace it with a new value. You can click middle on displayed value to edit the old value. You can click on <abort> or <exit> to cancel or activate the command. You can also use keyboard commands to interact with this kind of menu. The keyboard commands available are:

SPACE	Enter a value for an item.
c-E	Edit the value of an item.
REFRESH	Force complete redisplay.
HELP	Display list of commands.
END	Use these choices.
ABORT	Abort these choices.
c-N	Move down to next item.
c-P	Move up to previous item.
c-F	Move to next choice in a list of choices.
c-B	Move to previous choice in a list of choices.

```
More processing enabled: [✓ Yes]  No
Reverse video: Yes  [✓ No]
Vertical spacing: 2
Deexposed typein action: Wait until exposed  [✓ Notify user]
Deexposed typeout action: [✓ Wait until exposed]  Notify user  Let it happen
                              Signal error  Other
ALU function for drawing: [✓ Ones]  Zeroes  Complement
ALU function for erasing: Ones  [✓ Zeroes]  Complement
Screen manager priority: None
Save bits: [✓ Yes]  No
Default character style: FIX.ROMAN.NORMAL
Echo character style: NIL.NIL.NIL
Typein character style: NIL.NIL.NIL
End of screen action: [✓ Default]  Scroll  Truncate  Wrap
Amount to scroll by: Default
<abort> aborts, <end> uses these values
```

Figure 9. Set Window Options Menu

Press c-N and notice that the first option, "More processing enabled" is underlined. Press c-F. The underlining moves over to the word **Yes**. Press c-F again. Now the underlining is under the work "No". Press SPACE. The line redisplays with the **No** choice in boldface. Press c-N to move to the next option, "Reverse video". Press c-F. Suppose at this point you decide you do not want to change the video after all. Just press c-P (or c-N) to continue. Press c-P now to return to the "More processing enabled" and c-F followed by SPACE to set more processing back on. Now press END to exit from the menu.

Occasionally typing on a menu causes some overwriting of other parts of a menu. REFRESH or FUNCTION REFRESH will redisplay it correctly.

Using m-COMPLETE

Type `Show Directory sys:examples;`. The Show Directory command takes several keywords to control the format of the display. You can type them directly in the command line, but with some commands the interactions among keywords is complex and it is more convenient to see all the options and be able to alter them selectively. Press m-COMPLETE. You should see a menu like Figure 10. The items in the menu are mouse sensitive; you can select keyword values with the mouse or by keyboard commands. Pressing END or clicking on `<end> uses these values` activates the command.

```
Show Directory SYS:EXAMPLES;*.*.NEWEST
Files: SYS:EXAMPLES;*.*.NEWEST
Size: 0
Since: a universal time or a null value
Before: a universal time or a null value
Order: Smallest-First  Largest-First  Oldest-First  Newest-First  Name  Type
Output-Destination: a destination
<abort> aborts, <end> uses these values
```

Figure 10. Show Directory Command Menu

2.5.4 What You Have Learned

If you have followed all the directions given in the sections starting with "Getting Acquainted with Genera" (See the section "Getting Acquainted with Genera", page 18.), you should now be able to do the following:

- Use the System menu
- Use other menus
- Use the command processor to do some simple information gathering tasks
- Use the mouse
- Use facilities of Semanticue provided by Dynamic Windows to create new commands from your previous commands

You are now ready to learn in more detail about the command processor: See the section "Communicating with Genera", page 33.

For detailed descriptions of all the commands available in the command processor: See the section "Dictionary of Command Processor Commands", page 227.

To learn how to allow your application programs to take advantage of the power of dynamic windows and the command processor: See the section "The Command Processor Program Interface".

3. Communicating with Genera

3.1 Overview of the Command Processor

The command processor is a utility program that accepts a command and its arguments and then runs that command for you. The command processor takes care of various chores:

- Prompting for arguments
- Checking arguments for correctness
- Providing completion when possible
- Providing documentation on request

The command processor operates in all Lisp Listeners and **zl:break** loops. The prompt "Command: " indicates that you should enter a command or a Lisp form. By default, command processor is in *command-preferred mode.* This means that input to a Lisp Listener or **zl:break** loop is treated as a command if it begins with an alphabetic character or a colon. Input is treated as a Lisp form if it begins with a nonalphabetic character or is preceded by a comma.

For information on entering a command: See the section "Entering a Command", page 34.

For information on changing the command processor's mode, prompt, and other characteristics: See the section "Customizing the Command Processor", page 98.

For descriptions of predefined commands: See the section "Dictionary of Command Processor Commands", page 227.

For information on the command processor reader and the facility for defining your own commands: See the section "The Command Processor Program Interface".

For information on turning the command processor on and off: See the section "Turning the Command Processor on and Off", page 40.

3.2 Parts of a Command

A command has three logical parts to it, which you specify in this order:

1. *Command name.* This is a word or a series of words separated by spaces.

2. *Positional arguments.* These are arguments that the command processor prompts for directly after the command name. Some commands have several positional arguments; others have none. Commands that have arguments might use default values for the ones that you don't specify.

3. *Keyword arguments.* Some commands have keyword arguments that make it simple to modify the meaning of the commands. Most of these arguments require values. These arguments have default values that the command processor assumes if you specify the command without mentioning the argument name. Some commands have arguments whose values differ according to whether you omit the argument altogether or mention the argument name and omit its value. These argument defaults are called *unmentioned defaults* and *mentioned defaults.*

4. Some keyword arguments do not have values at all.

For information on entering command names and arguments: See the section "Entering a Command", page 34. See the section "Completion in the Command Processor", page 39.

For information on help in the command processor: See the section "Help in the Command Processor", page 38.

3.3 Entering Commands

3.3.1 Entering a Command

In entering a command, you enter the components in order: first the command name, then its positional arguments, then its keyword arguments, then the command terminator (RETURN or END). (When the command processor is in **:form-preferred** mode, you must precede the entire command by a colon: See the section "Setting the Command Processor Mode", page 99.)

The parts of the command can be entered using the keyboard or the mouse. You can click Mouse-Left on previous commands on the screen to *yank* them for reactivation. If you click sh-Mouse-Left on a previous command it is yanked and reactivated in one step.

When you type a command, items from your output history on the screen become mouse sensitive if they are appropriate as arguments to a command. Clicking Mouse-Left on such an object yanks it into the current command line.

c-Mouse-Middle yanks the word the mouse is over for use in composing a new command line. For example, if you have done Show Mail and a message refers to a file that you want to look at, you can yank the file name as an argument to a Show File command.

The command processor can *complete* components of commands. While you are typing a command name or keyword argument name, if you press SPACE the command processor attempts to complete the current word and all previous words in that command name or keyword argument name. If you press COMPLETE, the command processor attempts to complete the entire command name or keyword argument name. The command processor can also complete argument values that are members of a limited set of possibilities. If you press m-COMPLETE the command processor displays a menu of the argument values it has collected so far. You can then select values from the menu using the keyboard or the mouse. When you terminate a command, the command processor completes any command component in progress.

Some arguments have *default* values. If you press SPACE instead of typing an argument, the command processor uses the default for that argument. The command processor also uses the defaults for any arguments you haven't specified at all when you terminate the command.

All this means that you don't have to type an entire command to enter it. Suppose, for example, that you type the following:

```
d e SPACE f SPACE f o o . * SPACE : q SPACE y RETURN
```

You see the following on the screen:

```
Delete File (file [default ACME-BLUE:>joe>foo.lisp]) foo.*
(keywords) :Query (Yes, No, or Ask) Yes
```

While entering a command, pressing HELP or c-? displays documentation appropriate for the current stage of entering the command. See the section "Help in the Command Processor", page 38.

3.3.1.1 Supplying a Command Name

You type the command name, or some portion of it, followed by SPACE. The command processor either recognizes the command from what you have typed or it doesn't.

- When it recognizes the command, it fills in the part of the command name that you didn't type and then prompts you for the first argument. For example, you type:

  ```
  d e SPACE f SPACE
  ```

  The command processor displays:

  ```
  Delete File (file [default ACME-BLUE:>joe>foo.lisp])
  ```

- When it doesn't recognize what you have typed so far as being the possible beginning of a command, the command processor informs you that no such commands are available. You have to edit your input or erase it and start over.

- When it determines that what you have typed matches the beginning of several different commands, it fills in as much of the command as possible and waits for more input. You can use SPACE again to see if there is a default completion for this command, or you can use HELP or c-? to see the set of commands that begin with what you typed.

3.3.1.2 Supplying Positional Arguments to a Command

When the command processor has prompted you for a positional argument, you enter whatever argument is appropriate for the command. The prompt words indicate what the command expects:

```
Delete File (file [default ACME-BLUE:>joe>foo.lisp])
Set Package (A package)
Load Patches (for systems)
```

An argument can be either a single item or, sometimes, a set of items separated by commas. An argument cannot end with a comma, so SPACE can appear after a comma for attractiveness if you want; the command processor just ignores SPACE after a comma.

```
Load Patches (for systems) System, Zmail
```

You end each argument with SPACE. The command processor then checks whatever you have entered and prompts for the next argument (if there is one) or for the keyword arguments. If you haven't typed anything except SPACE, it fills in the default argument when one exists. Otherwise it checks what you typed for validity (for example, if the command wants a number, it makes sure that you didn't enter a string).

```
Delete File (file [default ACME-BLUE:>joe>foo.lisp]) foo.* (keywords)
```

Some arguments can only be members of a limited set of possibilities, displayed in the prompt. In this case the command processor can attempt to complete the argument. If you begin to type the argument and press SPACE, the command processor attempts to complete the current word and all words before that word in the argument. If you begin to type the argument and press COMPLETE, the command processor attempts to complete the entire argument. For example, you type:

```
s e SPACE c SPACE p SPACE f - p SPACE
```

The command processor displays:

```
Set Command Processor (Form-Only, Form-Preferred, Command-Preferred,
or Command-Only) Form-Preferred (prompt string)
```

What if one of the items in the argument list needs to contain one of the special characters (SPACE, comma, leading colon, or RETURN)? Use double quotes to delimit that item:

```
Show Hosts (hosts) Missouri,"Red River"
```

Most arguments have a default, which is usually indicated by the argument's prompt. When you want to use the default for an argument, you can indicate that simply by using SPACE. This terminates the argument, causing the command processor to fill in the default.

Sometimes when you supply a value for argument, the value that the command processor actually uses is a function of both the default and what you type. This is what happens with pathname arguments; the default pathname and the value that you type are *merged* to form the argument value that the command processor gives to the command.

Once you have specified as many of the arguments as you need (even none), you can use RETURN or END to enter the command. The command processor uses the defaults for any arguments you haven't specified.

Suppose you want to use the defaults for the remaining positional arguments, but you want to supply some keyword arguments. You must use SPACE to fill in the default for each of the remaining positional arguments. When you have finished the positional arguments, the command processor prompts for keyword arguments.

3.3.1.3 Supplying Keywords and Values for a Command

The command processor prompts for keyword arguments when you have entered all of the positional arguments for the command.

Suppose you have supplied all of the arguments to the Delete File command and are now being prompted for any keywords for modifying the standard action of the command. You enter keywords and their values in any order, finishing off the command with RETURN or END. The keyword prompt does not appear for every keyword, as that would clutter up your command.

The command processor can attempt to complete keyword argument names and values that are members of a limited set of possibilities. When you are typing a word, if you press SPACE the command processor attempts to complete that word and all previous words in the current keyword argument name or values. If you press COMPLETE, the command processor attempts to complete the entire keyword argument name or value in progress. For example, you type the following:

```
d e SPACE f SPACE f o o . * SPACE : e SPACE n SPACE
: q SPACE a RETURN
```

The command processor displays:

```
Delete File (file [default ACME-BLUE:>joe>foo.lisp]) foo.*
(keywords) :Expunge (Yes, No, or Ask) No :Query (Yes, No, or Ask) Ask
```

You can also press m-COMPLETE and see a menu of the arguments and their values.

Most keyword arguments have several values, but some are *flag* keywords with no value. For these keywords, you do not specify a value. Often, such flag keywords exist as synonyms for some other keyword/value combination. For example, suppose there was a keyword argument called :Expunge that had three values: Yes, No, and Ask. The person defining the command could have decided for convenience to offer :No-Expunge as a flag keyword that is synonymous with ":Expunge No".

Some commands have keyword arguments with interesting defaulting behavior. These arguments have two different kinds of "defaults", one that applies when you mention the keyword without explicitly supplying a value, and one that applies when you omit the keyword altogether. For example, consider the :Expunge argument for Delete File. When you omit :Expunge, the command processor assumes you mean ":Expunge No". When you supply :Expunge and use SPACE to fill in the default, it assumes you mean ":Expunge Yes". This style of argument occurs less often than the one with the conventional defaulting behavior.

Keywords can be specified at most once in a command line. The command processor views a command line in which the same keyword has been specified twice as ambiguous; you have to correct the problem by removing one of the keyword argument pairs.

3.3.2 Editing a Command

The command processor uses the input editor to manage typing, displaying, and editing of a command that you are entering. You can move from field to field within a command, change arguments, delete keywords, even change the command name. See the section "Editing Your Input", page 134.

3.3.3 Help in the Command Processor

Press HELP to the command processor at any time before or during entering a command. (Once you have started to enter a command, you can also use c-?.) It provides documentation that is appropriate for the particular stage you have reached in entering the command.

Before starting	Explains how to enter a command processor command.
Command name	Shows the commands that could be completions of what you have typed so far.
Positional argument	Explains the characteristics of the argument that is required at this position, including possible values.
Keyword argument name	If you have not yet typed a keyword flag character, the command processor lists the remaining arguments and briefly

describes them. If you have typed a keyword flag character, the command processor shows the keywords that could be completions of what you have typed so far.

Keyword argument value

The command processor presents documentation for the meaning of all the possible values of the argument.

3.3.4 Completion in the Command Processor

The command processor offers two kinds of completion: *partial* completion and *token* completion. A token is a command component, such as the command name or a keyword argument name.

- Partial completion: When you are typing a word in a command name or keyword argument name, if you press SPACE the command processor attempts to complete the current word and all previous words in the current command name or keyword argument name.

- Token completion: When you are typing a command name or keyword argument name, if you press COMPLETE the command processor attempts to complete the entire command name or keyword argument name in progress.

Completion is also available for argument values that are members of a limited set of possibilities, and for system and package names.

3.4 Command History

Command processor commands are maintained in the input editor input history, along with other input to the Lisp Listener or **zl:break** loop. c-m-Y yanks the last element of the history. m-Y yanks the next previous element. Thus you can press c-m-Y followed by m-Y m-Y ... to yank successively further back elements in your input history. c-m-0 c-m-Y lists the elements of the history. A numeric argument to c-m-Y yanks the element of the history specified by the argument.

c-m-sh-Y prompts you for a string and yanks the most recent element containing that string. m-sh-Y acts like m-Y, yanking successive previous elements.

Your output history is maintained on the Lisp Listener window. You can scroll back over your history using m-SCROLL or m-V. Scrolling forward is done with SCROLL and c-V, just as in Zmacs and Zmail.

s-R searches back through your history. END terminates the search so that you can yank the element you have found. s-S searches forward.

You can mark sections of your output to be pushed on the kill ring. `c-Mouse-Right` pops up a menu of marking and yanking options, or you can mark elements directly using `c-Mouse-Left`. Hold down the `CONTROL` key and the left mouse button and move the mouse over the area you want to mark. The marked region is underlined. You can push the marked region on the kill ring by clicking on that choice in the Marking and Yanking menu or by pressing `s-W`.

If you have two Lisp Listeners side by side on your screen, the histories of both remain mouse sensitive and you can yank elements from either one.

Clicking `Mouse-Right` on an element of your history pops up a menu of possible operations on that object. For example, clicking right on a pathname offers, among other operations, a choice of Show File.

For a list of the mouse gestures that can be used on to manipulate your history on dynamic windows: See the section "Mouse Gestures on Dynamic Windows", page 211.

You can also copy your output history into a Zmacs buffer for editing or saving in a file. See the section "Copy Output History Into Editor Command", page 231.

You can clear your output history if you want to clean up and do a garbage collection. See the section "Clear Output History Command", page 228.

3.5 Error Handling in the Command Processor

Part of the command processor's contract with the programs it serves is to collect syntactically valid arguments for the command you want to use. Thus if the command wants a numeric argument and you have entered a file spec, the command processor notices the problem, complains about the argument that you typed, moves the cursor there, and requests that you edit what you typed in order to make it appropriate for the command.

The command processor checks for errors of omission as well, warning you when you try to finish a command before specifying some argument that needs to be explicit.

In making its error warnings, the command processor prints out a diagnosis of the problem and asks you to correct your input. It never removes anything from what you have typed, since you are the best judge of how to remedy the problem.

3.6 Turning the Command Processor on and Off

The command processor is on by default in all Lisp Listeners and **zl:break** loops. You can turn the command processor on and off, but normally you should have to do neither. If you want the command processor to treat input differently from the default, or if you want a prompt that is different from the default, you can change these characteristics by using the Set Command Processor command or setting special variables: See the section "Setting the Command Processor Mode", page 99. See the section "Setting the Command Processor Prompt", page 100.

For example, suppose you want the command processor to act as if it weren't there. You can use the Set Command Processor command to set the dispatch mode to **:form-only** and the prompt to the empty string. Alternatively, you can set **cp:*dispatch-mode*** to **:form-only** and **cp:*prompt*** to **nil** or the empty string. If you then want to return the command processor to its default behavior, you can set **cp:*dispatch-mode*** to **:command-preferred** and **cp:*prompt*** to **"command: "**.

If for some reason you need to turn the command processor off completely, you can call **cp:cp-off**.

cp:cp-off *Function*

Turns off the command processor in all Lisp Listeners and **zl:break** loops.

Once you call **cp:cp-off**, you must call **zl:cp-on** to turn the command processor back on.

cp:cp-on &optional (*dispatch-mode* **cp:*dispatch-mode***) (*prompt-string* **nil** *prompt-supplied*) *Function*

Turns on the command processor and sets its mode and prompt in all Lisp Listeners and **zl:break** loops.

dispatch-mode is **:form-only**, **:command-only**, **:form-preferred**, or **:command-preferred**. For the meaning of these keywords: See the section "Setting the Command Processor Mode", page 99. This argument becomes the value of the variable **cp:*dispatch-mode***. The default mode is the current mode (the current value of **cp:*dispatch-mode***). The initial default mode is **:command-preferred**.

prompt is a prompt option for displaying the command processor prompt in Lisp Listeners and **zl:break** loops. This argument becomes the value of the variable **cp:*prompt*** and is passed to the input editor as the value of the **:prompt** option. The value can be **nil**, a string, a function, or a symbol other than **nil** (but not a list): See the section "Displaying Prompts in the Input Editor" in *Reference Guide to Streams, Files, and I/O.*

The default prompt depends on *dispatch-mode*. If *dispatch-mode* is **:command-preferred** or **:command-only**, the default prompt is "Command: ". If *dispatch-mode* is **:form-preferred** or **:form-only**, the default prompt is the empty string, and no prompt is displayed. If you supply a value of **nil** or the empty string, no prompt is displayed.

4. Using the Online Documentation System

4.1 Introduction to the Document Examiner

The *Document Examiner* is a utility for finding and reading documentation.

- Using the Document Examiner is similar to using the printed documentation. Books in the document set are available on a shelf or on your desk; in the Document Examiner, they are available in the *Current Candidates* pane when you first select the Document Examiner. Just as you can open a book to any topic and read through to the end of that topic, the Document Examiner lets you "open" the documentation to any topic and read to the end of that topic.

- When you use the Document Examiner, you do not have to remember how information is arranged; for example, you do not have to remember the section, chapter, or printed book in which a particular function is explained. Each function (and each section, chapter, or other division of printed information – even entire books), is directly accessible.

- In addition to looking up documentation, you can create *private documents* with the Document Examiner by placing *bookmarks* in documentation topics and saving the list of bookmarks for future use.

- The online documentation is kept in a *documentation database.* The documentation database consists of documentation binary files. Loaded into your Lisp world is index information about the documentation database.

- Each documentation topic is stored as a record in the documentation database. Each record contains information on a particular topic and is uniquely identified by a topic name. Records fall into two categories: Object records documenting code objects, such as **make-array** or **tv:menu**, and concept records documenting abstract ideas that are not tied to code, such as "Introduction to the Document Examiner". Records also have a type designation. Examples of object record types are function, flavor, and variable. Concept records have a type of section.

- SELECT D, [Document Examiner] in the System menu, and the command Select Activity Document Examiner select the Document Examiner. Pressing HELP in the Document Examiner displays a listing of its commands.

Note that the Document Examiner offers a command that lets you read complete self-documentation:

Document Examiner Documentation

Provides complete, chapter-length documentation of all Document Examiner features. This command is equivalent to clicking middle on [Help] in the command pane.

4.2 Looking up Documentation

You can look up documentation in the Document Examiner, in an editor (Zmacs, Zmail, Converse), or at a Lisp Listener using a variety of commands. One command looks up and displays documentation by name. Another set of commands pops up a menu of all documentation topic names that satisfy a query request. Such query requests are carried out by matching an initial substring, substrings, or whole words against documentation topic names or their *keywords*. (A keyword is comparable to a word in an index entry.) Clicking on a topic in one of these menus looks up and displays the documentation for that topic. See the section "Documentation Lookup Commands", page 46.

Another set of commands is available for repositioning text in the Document Examiner. See the section "Repositioning Text in the Document Examiner", page 60.

Your lookup request is always made in terms of a documentation topic name. You are prompted for a *type* (section or function, for instance) only when several topics have the same topic name but different types. For instance, suppose there are two topics whose names are "error"; one documents a flavor and the other a function. Requesting a display of "error" causes a menu of the possible types (flavor or function) to pop up. You choose the type you want displayed.

When you look up documentation, the more general the topic you look up, the larger the amount of documentation you see for it. The most general topic names are the names of the books in the printed documentation set. When you first select the Document Examiner, these books appear as items in the Current Candidates pane. If you are unsure what level in the documentation you need, use the command Find Table Of Contents, giving it the name of the printed book or section that interests you.

In the Document Examiner, you can look at an *overview* of a given topic. The overview includes the topic(s) and book(s) in which the topic appears, as well as a list of keywords included in the topic. In addition, the overview includes a graph showing the topic's position in the book or books in which it appears. You can use this graph to look at the topics that precede and follow the current topic in the printed documentation.

You can look at a topic's overview by using one of the following methods:

- Click middle on a mouse-sensitive item in the viewer.
- Click middle on a topic in the list of current candidates or the list of bookmarks.
- Use Show Overview at the command prompt and supply the name of a topic.
- Use [Show (M)] in the command menu and supply the name of a topic.

For more information on the overview facility: See the section "Show Overview", page 52.

In addition, you can find the printed book in which the topic appears by using What Document (m-X) in the editor.

When you use one of the *documentation find commands* in an editor or the Document Examiner, a menu of topic names is displayed. This menu includes all the topic names that fulfill your lookup query. When you click on one of the topic names, the chosen topic is displayed. The find commands are Find Initial Substring Candidates, Find Whole Word Candidates, Find Any Candidates, or Find Table Of Contents.

Recovering From a Stuck Document Examiner

When you look up documentation at a Lisp Listener or an editor, the Document Examiner is updated to include the last topic or menu you looked up. This normally happens within a few seconds. Occasionally, the topic you look up in the editor or Lisp Listener does not show up in a matter of seconds in the Document Examiner. If this occurs, enter Peek. In Peek, press P to see a listing of processes. Notice that a process called "DEX background" is showing. This process appears only if there is a problem. Click on this process and select "Debugger" from the menu. You should see a number of proceed options in the Debugger, one of which offers to skip trying to process the current topic and move on to the next pending one. Choose that proceed option. The background process that feeds queued topics or candidates lists to the Document Examiner should then "unplug" and put everything that it has been saving into the Document Examiner, one thing at a time.

Topics Pruned From the Documentation Database

When the Documentation Database is installed at your site, the installation manager has the option of pruning the database. By selecting from a menu of major sections in the database, the system manager specifies which files are deleted from the documentation database file server. In this way, space can be saved on the file server by pruning sections not needed at your site.

Trying to display a topic that has been pruned from the database causes a "dummy" topic to be displayed. Suppose, for example, the files containing the section "Streams" were pruned from the database at your site. Trying to display the topic ":tyi" produces the following display:

> **:tyi** *message*
> Documentation for **:tyi** as a Message is offline.
> It appears in document: *Reference Guide to Streams, Files, and I/O*
> Reload the file SYS: DOC; STR; STR2.SAB.4 to make this topic accessible online.

If you decide you do want to see the topic online, you can use **sage:load-index-info** to load the file that contains the index information for the topic.

sage:load-index-info *pathname* *Function*

Loads *pathname*, a .sab file, into your world.

4.3 Documentation Lookup Commands

The Document Examiner, the editor, and the command processor all provide various commands for looking up documentation. Some commands are available in all three contexts, while others are available in only one of the contexts. The following are descriptions of the commands provided, categorized according to the context in which the command is available.

Lookup Commands Available in the Document Examiner, Editor, and Command Processor

Show Documentation (an Overview)

Show Documentation looks up a topic and displays it. You can use the command in the Document Examiner, in an editor, and at a command processor.

- In the Document Examiner, Show Documentation prompts for a topic name, with completion, accepting only those topics for which documentation exists in the database. You can use Show Documentation in the Document Examiner any of the following ways:
  - Type the command at the command pane.
  - Use [Show] in the Document Examiner command pane menu.
  - Click left on a mouse-sensitive item in the viewer or an item in the list of candidates or list of bookmarks.

- In an editor, Show Documentation (`m-X`, `m-sh-D`) prompts you for a topic name, with completion, accepting only those topics for which documentation exists in the database. You can direct the display of a documentation topic to a supported printer (LGP1, LGP2, or DMP1) by issuing Show Documentation (`m-X`) with a numeric argument. This prompts for an output device.

- At a command processor, Show Documentation prompts you for a topic name, with completion, accepting only those topics for which documentation exists in the database. When you give the command the keyword argument `:destination`, the command offers to route it to the default text printer.

Note that topic names for methods are of the form (flavor:method *:generic-function-name flavor-name*), for example, (flavor:method :set-edges tv:menu). To look up documentation for methods, use one of the following strategies:

- Use Show Documentation, giving it the topic name of the method in the form (:method *:generic-function-name flavor-name*).

- Use Find Whole Word Candidates, giving it the name of the method in the form *:generic-function-name.* Then click on the item whose documentation you want to see.

Lookup Commands Available in the Document Examiner and Editor

Find Any Candidates

Sometimes you want to know if the documentation database contains any topics about a particular subject. You might have a string or strings in mind dealing with that subject. Using the command Find Any Candidates, you can search the database for any topics whose topic names or keywords contain the string or strings as *substring(s).*

A substring is a string that appears somewhere in another string. A substring can be an initial substring. However, when you use the command Find Any Candidates, the search is for a substring that appears anywhere in another string, not necessarily as the initial substring of some word or words in the string. The string "et" is a substring of the strings "set" and "setq". The string "et" is both a substring and an initial substring of the string "etc". The string "et" is not a substring of the strings "est" and "login".

The following situation shows how you can use this command: You want to know if the database contains any topics about setting values of variables. You guess that any such topics would use the string "set" somewhere in their topic names or keywords. So, you use the command Find Any Candidates to search the database for any topics whose topic names or keywords contain the string "set" as a substring. The search returns a list of almost 400 candidates.

You can provide the command with a string of several words, for instance, the string "resource window". Note that when the given string contains any space or hyphen characters, the command breaks the string into *tokens* using the space and hyphen characters as delimiters. For example, given the string "resource window", the command breaks it into two tokens, "resource" and "window".

The command looks at all the topic names and keywords in the database and lists any in which all the tokens appear as substrings, in effect performing a logical **and** test on the tokens. Given the string "resource window", the command lists several topics, among them the function **zl-user:defwindow-resource** and the section "The Top-level Function". Both tokens are substrings in the topic name **zl-user:defwindow-resource** and in the keywords of "The Top-level Function". The order in which you provide the words does not affect the search for topics.

Again, this search is performed not only on the topic names in the documentation database, but also on the keywords listed for each topic. This means that you often find topic names in which the given string does not appear at all. It does, however, appear among the topic's keywords.

In the Document Examiner, the command lists the topic names it has found in the candidates list. In an editor, the list takes the form of a menu.

In an editor, you can direct the display of a documentation topic to a supported printer (LGP1, LGP2, or DMP1) by issuing Find Any Candidates with a numeric argument. This pops up a menu offering to display the documentation on the screen or route it to a supported printer.

This command is also available as [Find (R)] in the Document Examiner command pane menu.

Find Initial Substring Candidates

Sometimes you want to know if the documentation database contains any topics about a particular subject. You might have an initial substring or substrings in mind dealing with that subject. Using the command Find Initial Substring Candidates, you can search the database for any topics whose topic names or keywords contain the substring or substrings as *initial substring(s).*

An initial substring is a string that appears as the beginning of some string. For example, the string "set" is an initial substring of the string "setq". The string "set" is not an initial substring of the string "reset". The string "set" is a substring of the string "reset".

The following situation shows how you can use this command: You want to know if the database contains any topics about setting values of variables. You guess that any such topics would use the string "set" as an initial substring somewhere in their topic names or keywords. So, you use the command Find Initial Substring Candidates to search the database for any topics whose topic names or keywords contain the string "set" as an initial substring. The search returns a list of over 200 candidates.

You can provide the command with a string of more than one word, for example, the string "set-globally". Note that when the given string contains any space or hyphen characters, the command breaks the string into *tokens* using the space and hyphen characters as delimiters. For example, given the string "set globally", the command breaks it into two tokens, "set" and "globally".

The command looks at all the topic names and keywords in the database and lists any in which all the tokens appear as initial substrings, in effect performing a logical **and** test on the tokens. Given the string "set globally", the command lists two topic names, the functions **zl:set-globally** and **zl:setq-globally**. Both tokens are initial substrings in each topic name. The order in which you provide the words does not affect the search for topics.

Again, this search is performed not only on the topic names in the documentation database, but also on the keywords listed for each topic. This means that you often find topic names in which the given string does not appear at all. It does, however, appear among the topic's keywords.

In the Document Examiner, the command lists the topic names it has found in the candidates list. In an editor, the list takes the form of a menu.

Find Initial Substring Candidates treats leading punctuation as part of the word. Thus, asking for initial substring of "area" does not return "*area" or "%area". If you want "anything containing area" you must use the most general matching command, Find Any Candidates.

In an editor you can direct the display of a documentation topic to a supported printer (LGP1, LGP2, or DMP1) by issuing Find Initial Substring Candidates with a numeric argument. This pops up a menu offering to display the documentation on the screen or route it to a supported printer.

This command is also available as [Find (M)] in the Document Examiner command pane menu.

Find Table of Contents

Displays a menu containing the given topic's table of contents. For example, the table of contents of "The Document Examiner" displays as:

```
The Document Examiner
  Introduction to the Document Examiner
  Looking Up Documentation
    Recovering From a Stuck Document Examiner
    Topics Pruned From the Documentation Database
  Documentation Lookup Commands
     Lookup Commands Available in the Document Examiner, Editor, and Command Processor
       Show Documentation (an Overview)
     Lookup Commands Available in the Document Examiner and Editor
       Find Any Candidates
  .
  .
  .
```

You can ask to see a table of contents for any topic; it is not limited to top-level books. The table of contents of "Documentation Lookup Commands" displays as:

```
Documentation Lookup Commands
  Lookup Commands Available in the Document Examiner, Editor, and Command Processor
    Show Documentation (an Overview)
  Lookup Commands Available in the Document Examiner and Editor
    Find Any Candidates
  .
  .
  .
```

This command is also available as [Show (R)] in the Document Examiner command pane menu.

Find Whole Word Candidates

Sometimes you want to know if the documentation database contains any topics about a particular subject. You might have a word or words in mind dealing with that subject. Using the command Find Whole Word Candidates, you can search the database for any topics whose topic names or keywords contain the word or words as *whole* word(s).

A whole word is a string separated from other strings by space or hyphen characters. The string "set" appears as a whole word in the topic name "Creating a Set of Condition Flavors" and in the topic name "set-globally". It does not appear as a whole word in the topic name "setq". In the topic name "setq", the string "set" appears as an initial substring.

The following situation shows how you can use this command: You want to know if the database contains any topics about setting values of variables. You guess that any such topics would use the string "set" in their topic names or keywords. So, you use the command Find Whole Word Candidates to search the database for any topics whose topic names or keywords contain the string "set" as a whole word. The search returns a list of almost 200 candidates.

You can provide the command with more than one word. For example, you give the command the string "set globally". Note that when the given string contains any space or hyphen characters, the command breaks the string into *tokens* using the space and hyphen characters as delimiters. For example, given the string "set globally", the command breaks it into two tokens, "set" and "globally".

The command looks at all the topic names and keywords in the database and lists any in which all the tokens appear as whole words, in effect performing a logical **and** test on the tokens. Given the string "set globally", the command lists exactly one topic, the function **zl:set-globally**. The order in which you provide the words does not affect the search for topics.

Again, this search is performed not only on the topic names in the documentation database, but also on the keywords listed for each topic. This means that you often find topic names in which the given string does not appear at all. It does, however, appear among the topic's keywords.

In the Document Examiner, the command lists the topic names it has found in the candidates list. In an editor, the list takes the form of a menu.

Find Whole Word Candidates treats leading punctuation as part of the word. Thus, asking for whole word match of "area" does not return "*area" or "%area". If you want "anything containing area" you must use the most general matching command, Find Any Candidates.

In an editor you can direct the display of a documentation topic to a supported printer (LGP1, LGP2, or DMP1) by issuing Find Whole Word Candidates with a numeric argument. This pops up a menu offering to display the documentation on the screen or route it to a supported printer.

This command is also available as [Find] in the Document Examiner command pane menu.

Lookup Commands Available in the Document Examiner

Select Candidate List

Selects from the history of candidates lists, popping up a menu of the documentation find commands and their arguments issued in the current session. You can reinstate a list of candidates by using this command.

This command is very helpful when, for instance, you need to cycle through several lists. Instead of reconstructing a candidate list each time you want to look at it, just use Select Candidate List and click on the list that you want to see.

This command is also available as [Select] in the Document Examiner command pane menu.

Show Overview

Prompts for a topic name. Shows an overview of the given topic. This overview has two parts:

- The top part includes the type (section or function, for instance) and name of the topic, possibly a short summary of the topic, the names of any other topics in the documentation that include this one, the names of any printed books that contain the topic, and the topic's keywords. The names of the topic(s) and book(s) are mouse sensitive.
- The bottom part is a graph of the document hierarchy around the topic you choose. (If the topic has multiple parents, multiple graphs are displayed.) This graph includes:
    - The topic's parent (the topic that includes the original topic)
    - The parent's children (other topics called by the parent – the siblings of the original topic)
    - The topic's children (topics the original topic includes)

Figure 11 shows the display produced by doing Show Overview of the topic "Disk Error Handling".

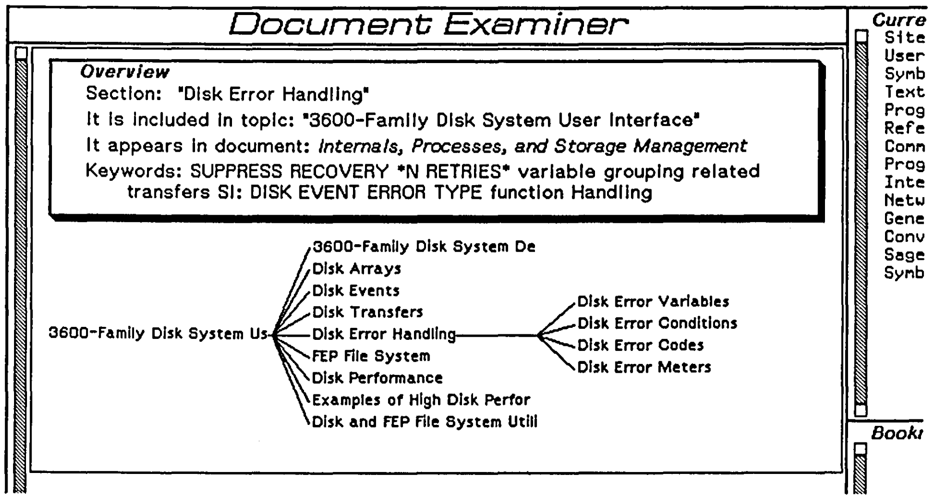
Note that the graph has some limitations:

- Long topic names are truncated in the graph.
- Very large hierarchies cannot be displayed fully; the truncated part of the display is not accessible.

Like the topics in the top part of the overview display, the topics in the graph are mouse sensitive; you can use the same mouse commands on them that you use on topics in the top part of the display. This provides a good way to explore the context in which a topic occurs. Clicking middle on different topics to generate new graphs starting from those locations in the tree is the online equivalent of looking in a printed book at the immediately surrounding pages for a topic.

For example, when you get an overview of "Disk Error Handling", you can see that the topic is included in the topic "3600-Family Disk System User Interface" and that it includes four other topics. If this overview does not give you enough information, try an overview on the parent topic (in this case, "3600-Family Disk System User Interface").

This command is also available as [Show (M)] in the Document Examiner command pane menu.



Document Examiner

Overview
Section: "Disk Error Handling"
It is included in topic: "3600-Family Disk System User Interface"
It appears in document: *Internals, Processes, and Storage Management*
Keywords: SUPPRESS RECOVERY *N RETRIES* variable grouping related
    transfers SI: DISK EVENT ERROR TYPE function Handling

Figure 11. Document Examiner display of Show Overview of topic "Disk Error Handling".

Lookup Commands Available In an Editor

What Document (m-X)

Displays the name of the printed book that contains the given documentation topic. If the topic is included in more than one book, the titles of all the books containing the given topic are listed. In the Document Examiner, this information is available in the topic overview by clicking middle on any mouse-sensitive item in the viewer or any item in the list of current candidates or list of bookmarks.

Lookup Commands Available At a Lisp Listener and In Zmacs

When you are typing at a Lisp Listener or in Lisp Mode in Zmacs, you can use the following input editor commands to look up the documentation for the current Lisp object (the one that precedes point). For example, pressing `m-sh-A` after typing `(zl:login 'whit` displays the documentation for **zl:login**.

`m-sh-A`	Looks up the documentation for the current function.

`m-sh-V`	Looks up the documentation for the current variable.
`m-sh-F`	Looks up the documentation for the current flavor.

When you look up documentation at a Lisp Listener using one of these input editor commands, the documentation appears on the screen, and the input editor then redisplays whatever you were typing.

When these commands do not find any documentation for the current function/variable/flavor in the documentation database, they check the object itself for a documentation string. If they find a documentation string, the string is displayed.

It should be noted that once a documentation string is displayed in this manner, the string has been installed as the documentation in your world. Thereafter, the display does not change if the documentation string is changed. In other words, this facility does not provide support for your putting new documentation into the documentation database.

4.4 Documentation Hardcopy Commands

The Document Examiner provides several commands for hardcopying topics in different combinations:

Hardcopy Documentation	Sends the documentation for a topic to a supported printer (LGP1, LGP2, or DMP1). This option is also available on the menu obtained by clicking right on a mouse-sensitive topic anywhere in the Document Examiner.
Hardcopy Private Document	Prints a private document on a supported printer (LGP1, LGP2, or DMP1). For more information on private documents: See the section "Document Examiner Private Documents", page 61.
Hardcopy Viewer	Sends the topics in a viewer to a supported printer (LGP1, LGP2, or DMP1). Also available as [Viewer (R)] in the command menu.

4.5 Document Examiner Window

When you look at the Document Examiner window you see the following panes:

Pane	*Description*
Viewer	Displays documentation.
Current candidates	When you first enter the Document Examiner, displays the list of books registered in the documentation database. After that, displays the menu of topics that appeared the last time you used one of the documentation find commands.
Bookmarks	Displays a list of bookmarks, which are the names of topics displayed in the viewer or added to the list of bookmarks without being displayed.
Commands	Accepts commands at the prompt to the left and displays a menu of selected Document Examiner commands to the right.

4.5.1 Document Examiner Viewer

The large area on the left of the Document Examiner window is called the viewer. Documentation is displayed in the viewer. You can have multiple viewers, just as you can have multiple editor buffers. The viewer currently visible is called the *current viewer*. You can choose another viewer by using the command menu item [Viewer]. The command prompts you for the name of a viewer. You can see a list of viewers by pressing c-?. This is a completion command. It lists the possible completions of your response so far. Since you have made no response at this point, all viewers are possible completions. Items in the list are mouse sensitive.

To view documentation topics in the Document Examiner viewer, you can do one of several things:

- Click on mouse-sensitive items in the viewer.
- Click on topics in the list of current candidates or the list of bookmarks. See the section "Document Examiner List of Current Candidates", page 57. See the section "Document Examiner List of Bookmarks", page 58.
- Use the Show Documentation command in the command pane. See the section "Show Documentation (an Overview)", page 46.
- Use [Show] in the command pane menu. See the section "Document Examiner Command Pane", page 59.

In the Document Examiner, when you select a topic for viewing, the topic is displayed at the end of the current viewer and the topic's name is added to the list of bookmarks. Topics chosen for display in the viewer are separated by horizontal lines.

When you select a topic for viewing at a Lisp Listener or an editor, the topic is displayed there, added to the end of the current Document Examiner viewer, and the topic name is added to the end of the list of bookmarks. However, when you abort out of viewing a topic at a Lisp Listener or an editor, the Document Examiner just adds the topic name to the end of the list of bookmarks and does not display the topic in the current viewer.

Examples of Lisp code whose lines are wider than the viewer display with those lines wrapped around. When you need to see such examples in their entirety, use the Command Processor command Show Documentation in a wider window (for example, a Lisp Listener).

In the viewer, cross-references and documented Lisp objects are mouse sensitive. The following actions can be performed on mouse-sensitive items:

Mouse click	*Action*
left	Displays the topic in the current viewer.
middle	Shows an *overview* of the topic, in two parts. The top part includes the type (section or function, for instance) and name of the topic, possibly a short summary of the topic, the names of any other topic(s) in the documentation that include this one, the names of the printed books that contain the topic, and the topic's keywords. The bottom part is a graph of the document hierarchy around the topic you choose. (If the topic has multiple parents, multiple graphs are displayed.) This graph includes: • The topic's parent (the topic that includes it) • The children of the parent (other topics called by the parent – the siblings of the original topic) • The children of the original topic (topics the original topic includes) The names of the topic(s) and book(s) in both parts of the overview are mouse sensitive. For an example of an overview: See the section "Show Overview", page 52. Issuing any command, pressing any keyboard key, or clicking a mouse button causes the overview display to go away.
sh-middle	On a mouse-sensitive item in the viewer or list of current candidates, adds the name of the topic to the list of bookmarks. On an item in the list of bookmarks, discards the name of the topic from the list of bookmarks, and, if the topic has been displayed, discards the display from the current viewer.
right	Pops up a menu of several commands with which to act on the display. Commands listed but not mouse sensitive do not apply to the pane on which you clicked.

You can create, remove, and hardcopy viewers whenever you want and select another viewer by using the following commands in the Document Examiner.

Command	*Action*
Select Viewer	Selects or creates a viewer, prompting for a name. Also available as [Viewer] in the command menu.
Remove Viewer	Removes a viewer, prompting for a name, then selects the last viewer displayed. Also available as [Viewer (M)] in the command pane.
Hardcopy Viewer	Sends the topics in a viewer to a supported printer (LGP1, LGP2, or DMP1). Also available as [Viewer (R)] in the command menu.

4.5.2 Document Examiner List of Current Candidates

The upper right-hand pane of the Document Examiner window contains the list of current candidates, which begins as a menu of books registered in the documentation database. It then becomes the menu of topics that appeared the last time you used one of the documentation find commands. A menu remains until it is superseded by the next such command. Note that lines that are wider than the list of current candidates pane are truncated.

You can reinstate a list of candidates by using the command Select Candidate List or the menu command [Select], which pops up a menu of the documentation find commands and their arguments issued in the current session.

The following actions can be performed on topics in the list of current candidates:

Mouse click	*Action*
left	Displays the topic in the current viewer.
middle	Shows an *overview* of the topic, in two parts. The top part includes the type (section or function, for instance) and name of the topic, possibly a short summary of the topic, the names of any other topic(s) in the documentation that include this one, the names of the printed books that contain the topic, and the topic's keywords. The bottom part is a graph of the document hierarchy around the topic you choose. (If the topic has multiple parents, multiple graphs are displayed.) This graph includes: • The topic's parent (the topic that includes it) • The children of the parent (other topics called by the parent – the siblings of the original topic)

	• The children of the original topic (topics the original topic includes) The names of the topic(s) and book(s) in both parts of the overview are mouse sensitive. For an example of an overview: See the section "Show Overview", page 52. Issuing any command, pressing any keyboard key, or clicking a mouse button causes the overview display to go away.
sh-middle	On a mouse-sensitive item in the viewer or list of current candidates, adds the name of the topic to the list of bookmarks. On an item in the list of bookmarks, discards the name of the topic from the list of bookmarks, and, if the topic has been displayed, discards the display from the current viewer.
right	Pops up a menu of several commands with which to act on the display. Commands listed but not mouse sensitive do not apply to the pane on which you clicked.

4.5.3 Document Examiner List of Bookmarks

The lower right-hand pane of the Document Examiner window contains the list of bookmarks. This is a history of bookmarks you place in the documentation. A bookmark is a pointer to a documentation topic. Each time you display a topic, a bookmark is placed in that topic, and the name of the topic is added to the list of bookmarks. You can also simply place a bookmark in a topic without displaying it in the viewer by clicking middle twice on an item in the list of current candidates. When you select another viewer, the list of bookmarks associated with it is also selected.

The list of bookmarks distinguishes between bookmarks whose topics have been displayed and those that have not. Topics that are displayed in the viewer are listed on a white background in the order in which you looked them up. Topics not displayed in the viewer follow and are listed on a gray background in the order in which you created the bookmarks. A marker on the list of bookmarks indicates the topic currently being displayed at the top of the viewer.

Lines that are wider than the list of bookmarks pane are truncated.

The following actions can be performed on topic names:

Mouse click	*Action*
left	Displays the topic in the current viewer.
middle	Shows an *overview* of the topic, in two parts. The top part includes the type (section or function, for instance) and name of the topic, possibly a short summary of the topic, the names of

any other topic(s) in the documentation that include this one, the names of the printed books that contain the topic, and the topic's keywords.

The bottom part is a graph of the document hierarchy around the topic you choose. (If the topic has multiple parents, multiple graphs are displayed.) This graph includes:

- The topic's parent (the topic that includes it)
- The children of the parent (other topics called by the parent – the siblings of the original topic)
- The children of the original topic (topics the original topic includes)

The names of the topic(s) and book(s) in both parts of the overview are mouse sensitive. For an example of an overview: See the section "Show Overview", page 52.

Issuing any command, pressing any keyboard key, or clicking a mouse button causes the overview display to go away.

sh-middle	On a mouse-sensitive item in the viewer or list of current candidates, adds the name of the topic to the list of bookmarks. On an item in the list of bookmarks, discards the name of the topic from the list of bookmarks, and, if the topic has been displayed, discards the display from the current viewer.
right	Pops up a menu of several commands with which to act on the display. Commands listed but not mouse sensitive do not apply to the pane on which you clicked.

4.5.4 Document Examiner Command Pane

The bottom portion of the Document Examiner window contains the command pane. The command pane offers completion on command names as well as topic names. c-? and (after you type at least one character) HELP display a mouse-sensitive list of possible completions. Pressing HELP before starting to type a command name displays the available commands.

The command pane contains a command menu at the lower right. Use the following mouse clicks to perform these actions or commands.

Mouse	*Command*
[Help]	Brief command summary
[Help (M)]	Show the Document Examiner documentation

[Show]	Show Documentation
[Show (M)]	Show Overview
[Show (R)]	Find Table Of Contents
[Viewer]	Select Viewer
[Viewer (M)]	Remove Viewer
[Viewer (R)]	Hardcopy Viewer
[Find]	Find Whole Word Candidates (XXXXX)
[Find (M)]	Find Initial Substring Candidates (XXX....)
[Find (R)]	Find Any Candidates (..XXX..)
[Select]	Select Candidate List
[Private]	Read Private Document
[Private (M)]	Load Private Document
[Private (R)]	Save Private Document

4.6 Repositioning Text in the Document Examiner

The Document Examiner viewer, list of current candidates, and list of bookmarks each have a bar located at its left edge. These bars provide the scrolling capabilities found in dynamic windows throughout Genera. For more information on the scrolling mechanism: See the section "Scrolling", page 10.

Note that, when you display a multipage topic, the positioning mechanism knows about only the part of the topic you have seen. If you look at only one page of a multipage topic, the Document Examiner knows about only that page. Positioning in the Document Examiner works this way so as to limit the amount of space that documentation takes up in memory.

You can perform several types of positioning with the mouse and the bar. You can get a listing of positioning commands by pressing HELP while in the Document Examiner.

- To reposition text forward one screen:
  - Press c-V.
  - Press SCROLL.
  - Click left in the box at the bottom of the bar.

- To reposition text backward one screen:
  - Press m-V.

  - Press m-SCROLL.
  - Click right in the box at the bottom of the bar.

- To reposition quickly to any part of the current viewer:
  - Place the mouse over the scroll bar, note the repositioning options in the mouse documentation line, and click accordingly.

- To reposition a few lines forward:
  - Press c-SCROLL.
  - Click left in the arrow at the top of the bar.

- To reposition a few lines backward:
  - Press c-m-SCROLL.
  - Click right in the arrow at the top of the bar.

- To reposition to the beginning or end of the current viewer or topic:
  - Press m-< to reposition to the beginning of the current viewer.
  - Press m-> to reposition to the end of the current viewer. Note that using this command while you have a partially displayed topic exposed in the viewer refreshes the display.

Two additional commands allow you to reposition text for a single topic:

Beginning of Topic	Repositions the viewer to the beginning of the current topic.
End of Topic	Repositions the viewer to the last screen displayed for the current topic.

4.7 Document Examiner Private Documents

The Document Examiner provides mechanisms for placing bookmarks in the online documentation and for creating private documents out of these bookmarks. The bookmarks are pointers to documentation topics in the database. A private document is a collection of bookmarks you put together and write out to a file. This allows you to create your own customized documents by grouping together selected documentation topics.

To create a private document you first create a list of bookmarks, either by looking up some topics or by clicking appropriately on the list of candidates or on mouse-sensitive items in the viewer. Then you save the list of bookmarks, using the command Save Private Document, answering its prompt with a pathname of a file to contain the bookmarks. You can load (Load Private Document) or read (Read Private Document) the private document back into the Document Examiner at any time, again answering the prompt with the pathname of the file that contains the bookmarks for the private document. For example:

1. Create a list of bookmarks consisting of some topics documenting login procedures and functions. For example:

   ```
   login-forms
   login-setq
   System Initialization Lists
   zl:login
   ```

2. Use Save Private Document. The command prompts you and you answer with the pathname of a file to contain the private document's bookmarks. Following is the prompt for Save Private Document:

   ```
   Enter a pathname for the document to contain these bookmarks
   (default ACME-BLUE: /usr2/whit/private.psb):
   ```

   Pathname merging is supported by this command and the default location for a private document is always your home directory. So, with a home directory of /usr2/whit/ on ACME-BLUE, if you give Save Private Document the filename "login-book", the command writes the list of bookmarks to ACME-BLUE: /usr2/whit/login-book.psb.

The following commands manipulate private documents:

Save Private Document
: Saves the current list of bookmarks as a private document, prompting for a pathname. Save Private Document writes the list of bookmarks to the file whose pathname is given.

Read Private Document
: Reads a private document into your computer and shows it in the viewer. This command prompts for the pathname of a file containing the bookmarks of a private document and the name of a viewer to show it in. The default location for a private document is always your home directory, and pathname merging follows the standard rules.

Load Private Document
: Loads a private document into your computer but does not show it in the viewer. This command prompts for the pathname of a file containing the bookmarks of a private document and the name of a viewer. The default location for a private document is always your home directory, and pathname merging follows the standard rules.

Hardcopy Private Document

Prints a private document on a supported printer (LGP1, LGP2, or DMP1). For more information on private documents: See the section "Document Examiner Private Documents", page 61.

5. Creating and Manipulating Files

5.1 Overview

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

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

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

Since these subsystems share Zwei in the dynamically linked Lisp environment, many of the commands available as Zmacs commands are available in other editing contexts as well.

In this manual, we discuss Zmacs commands in the context of Zmacs only. We also describe Dired, the directory editor, since it is used within Zmacs.
You can find a tutorial for Zmacs in sys:examples; directory. Information on running it can be found in the file teach-zmacs-info.text.

```
Show File sys:examples;teach-zmacs-info.text
```

You can enter, or invoke, the editor in several ways: Press `SELECT E`, use the mouse, or run either the function **ed** or the function **zwei:edit-functions**. You can also use the command Select Activity, specifying either Zmacs or Editor as its argument.

5.2 Entering Zmacs with `SELECT E`

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

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

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

5.3 Entering Zmacs with the Mouse

You can invoke the editor using the mouse.

Summon a System menu by clicking right twice [(R2)]. Then click left on the Edit option [Edit (L)], which puts you into a Zmacs buffer. As for SELECT E, if you are returning to the editor Zmacs puts you back at the same place in the same buffer, and if you are entering Zmacs for the first time it puts you in an empty buffer.

5.4 Entering Zmacs with ed

The Lisp function **ed** enters Zmacs from a Lisp Listener. See the function **ed**, page 180.

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

arg can have these values.

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

5.5 Entering Zmacs with zwei:edit-functions

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

zwei:edit-functions Function

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

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

Zmacs sorts the list into an appropriate order, putting definitions from the same file together, and creates a support buffer called `*Function-Specs-to-Edit-`*n*`*`. It selects the editor buffer containing the first definition in the list.
To insert new text anywhere in the buffer, position the cursor at the place you want the new text to go and type the new text. Zmacs always inserts characters at the cursor. The text to the right of the cursor is pushed along ahead of the text being inserted.

5.6 Keystrokes

A keystroke has a character component and a modifier component, and is performed by pressing a *primary key* (alphanumeric), possibly while holding down a *shift key* or a group of shift keys. The primary key held down with either the `SHIFT` or `SYMBOL` keys determines the *character* part of a keystroke. Whether you hold down the other shift keys, `CONTROL`, `META`, `HYPER`, and `SUPER`, determines the *modifier* part of the keystroke.

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

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

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

5.7 Extended Commands

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

`m-X`	Extended Command

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

Command completion is provided. See the section "Completion for Extended Commands (`m-X` Commands)", page 115.

5.8 Description of Moving the Cursor

To do more than insert characters, you have to know how to move the cursor.

For complete descriptions of the commands summarized here and other cursor-moving commands: See the section "Moving the Cursor in Zmacs" in *Text Editing and Processing.*

5.9 Summary of Moving the Cursor

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

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

`c-F` Forward
Moves forward one character.

`c-B` Backward
Moves backward one character.

`m-F` Forward Word
Moves forward one word.

`m-B` Backward Word
Moves backward one word.

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

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

`c-N`	Down Real Line	Moves down one line.
`c-P`	Up Real Line	Moves up one line.
`m-]`	Forward Paragraph	Moves to the start of the next paragraph.
`m-[`	Backward Paragraph	Moves to the start of the current (or last) paragraph.
`c-X ]`	Next Page	Moves to the next page.
`c-X [`	Previous Page	Moves to the previous page.
`c-V`, `SCROLL`	Next Screen	Moves down to display the next screenful of text.
`m-V`, `m-SCROLL`	Previous Screen	Moves up to display the previous screenful of text.
`m-<`	Goto Beginning	Moves to the beginning of the buffer.
`m->`	Goto End	Moves to the end of the buffer.

5.10 Getting Out of Trouble

5.10.1 Overview of Getting Out of Trouble

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

This section tells you how to recover from these situations.

5.10.2 Getting Out of Prefixes and Prompts

Most of the commands we have described are single keystrokes, but some keystrokes are prefixes that must be completed with a second keystroke to specify a command. `c-X` is the most important of these.

5.10.2.1 Getting Out of Keystroke Prefixes

If you press a c-X and don't mean it, you can get out by pressing either c-G or ABORT. These are general "get me out of here" commands, which you should use whenever you get yourself into a confused state. ABORT and c-G are, for the most part, synonymous in Zmacs.

5.10.2.2 Getting Out of Minibuffer Prompts

Sometimes you accidentally type a command that prompts for some additional information, or you type such a command on purpose and change your mind afterwards. When Zmacs prompts and you just want to get out of the minibuffer and back to where you were, press ABORT. If, instead, you wish to cancel and reenter your response, use c-G, which clears any typein but leaves you still in the minibuffer. When the minibuffer is empty, c-G cancels the minibuffer command. (With some echo area prompts, you have to use ABORT.)

ABORT Abort At Top Level

Cancels the last command typed. It also cancels numeric arguments and region marking.

c-G Beep

Cancels the last command. It also cancels numeric arguments and region marking, except when given an argument. It cancels one thing at a time, so that if you've typed a number of commands or responses, you must use use successive c-Gs to cancel each one and return to top level.

5.10.3 Large Deletions

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

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

5.10.3.1 Getting Text Back

The system has different histories for different contexts. One of these is always the *current history.* The two histories that you need to use for yanking in Zmacs are the *kill history* and the *command history.* The kill history remembers pieces of text that you killed or copied into it. In the context of Zmacs, the command history remembers all the editor commands that use the minibuffer in any way.

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

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

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

For complete descriptions of killing and yanking: See the section "Working with Regions in Zmacs" in *Text Editing and Processing.*

5.11 Overview of Finding Out About Zmacs Commands

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

5.12 Finding Out About Zmacs Commands with HELP

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

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

5.13 Finding Out What a Zmacs Command Does

HELP C

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

5.13.1 Example

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

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

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

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

5.13.2 Finding Out What a Prefix Command Does

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

5.13.3 Finding Out What an Extended Command Does

HELP D

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

5.14 Searching for Appropriate Zmacs Commands

HELP A

m-X Apropos

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

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

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

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

5.14.1 Method for Searching for Appropriate Zmacs Commands

To find the command you want, just press `HELP A` or type `m-X` Apropos. Zmacs prompts you for a substring, you enter your guess, and then Zmacs displays short descriptions of all the commands whose names contain that substring. If the substring that you enter contains a space, then Zmacs displays a short description of all the commands whose names or help documentation includes a similarly positioned space. Each description gives the short documentation for the command and tells what keystrokes invoke it.

5.14.2 Example of a Search String for `HELP A`

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

5.15 Finding Out What You Have Typed

`HELP L`

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

If this happens to you it is often very enlightening to press `HELP L` to list the last 60 keystrokes you typed. By examining your own recent activity, it is often possible to find out where you went wrong and how to save yourself.
Some Zmacs operations require you to provide names – for example, names of extended commands, Lisp objects, buffers, or files. Often you do not have to type all the characters of a name; Zmacs offers *completion* over some names. When completion is available, the word `Completion` appears in parentheses above the right side of the minibuffer.

You can request completion when you have typed enough characters to specify a unique word or name. For extended commands and most other names, completion works on initial substrings of each word. For example, m-X c SPACE b is sufficient to specify the extended command Compile Buffer. SPACE, COMPLETE, RETURN, and END complete names in different ways. Press HELP or click right once, [(R)], on the editor window or minibuffer to list possible completions for the characters you have typed. c-/ displays every command that contains the substring.

SPACE	Completes words up to the current word.
HELP or c-?	Displays possible completions in the typeout area.
[(R)]	Pops up a menu of possible completions.
c-/	Runs Apropos for each of the partially typed words in the name.
COMPLETE	Completes as much as possible. This could be the full name.
RETURN or END	Confirms the name if possible, whether or not you have seen the full name.

5.16 Creating a Buffer

Zmacs creates your initial buffer when you first enter the editor. To create other buffers, use c-X B, Select Buffer, to create an empty buffer or c-X c-F, Find File, to create either an empty buffer or a buffer containing a file.

c-X B prompts for the name of the buffer to which you want to go. Type the buffer name and RETURN. If the buffer exists, Zmacs switches to that buffer and displays it on the screen. If the buffer does not already exist, Zmacs offers to let you create it by terminating the buffer name with c-RETURN. When you create a new (empty) buffer, the display is blank.

The other way to create another buffer is c-X c-F, Find File. (c-X c-F) is described in detail in "Editing Existing Files". c-X c-F prompts for the name of a file, terminated by RETURN.

When you type c-X c-F for the first time in a Zmacs session, Zmacs offers you, as a default file name, an empty file (with the Lisp suffix native to your host computer) in your home directory on your host computer. For example:

System	*Empty Buffer Name*
Lisp Machine	foo.lisp
UNIX	foo.l
VMS	foo.lsp

Base and Syntax Default Settings for Lisp

When you read a file that has a Lisp file type into the buffer, if that file does not begin with an attribute line containing Base and Syntax attributes, Zmacs warns that the file "has neither a Base nor a Syntax attribute" and announces that it will use the defaults, Base 10 and Common-Lisp. See the section "Buffer and File Attributes in Zmacs" in *Text Editing and Processing*.

Buffer Contents with `c-X c-F`

The first time you use `c-X c-F`, you can create an empty buffer using the Zmacs default file name, create an empty buffer using a name that you specify, or create a buffer containing an existing file:

- To create an empty buffer with the initial default file name as the one Zmacs associates with your buffer, press `RETURN`.
- To create a new empty buffer, respond with any name. Zmacs switches to an empty buffer, gives the buffer the new name, and displays (New File) in the echo area.
- To create a new buffer containing some file, respond to the prompt with the name of that file. Zmacs switches to an empty buffer, reads that file in, and names the buffer appropriately.

5.17 Creating a File

The first time you save or write the buffer, Zmacs creates the new file. You can create a new file with `c-X c-S`. Since a new file does not have a name associated with it yet, Zmacs asks for a name for the new file. It offers a *default pathname*, which is the name of the buffer. If you wish to save the file out to the default pathname, simply type a `RETURN` in response to the prompt.

If you wish to save the buffer in another file, provide that name as your response. Completion is offered to simplify your response.

You can also write the buffer out with `c-X c-W`, Write File. Zmacs prompts in the minibuffer for the name of the place you want to write the buffer's contents. `c-X c-W` also offers a default pathname, in this case, the name you supplied with `c-X c-F`.

6. Sending and Receiving Messages and Mail

6.1 Using Zmail

6.1.1 Introduction

Zmail is a display-oriented mail system for Genera. Using Zmail, you can send and receive electronic mail, archive your mail in disk files, and operate on groups of messages selected according to very flexible criteria. This tutorial is intended to give you a brief introduction to the basic features of Zmail. For a complete description of all Zmail's capabilities: See the section "Zmail Reference Guide" in *Communicating with Other Users.*

Since Zmail messages are typed into editor buffers, some familiarity with the editor is also helpful. (See the section "Zmacs Manual" in *Text Editing and Processing.*)

6.1.2 Starting up Zmail

Before running Zmail, be sure that you are logged in. See the section "Logging in", page 17.

To run Zmail, do one of the following:

- Press SELECT M.
- Give the command Select Activity Zmail (or Select Activity Mail).

You get a display similar to Figure 11, called the *top-level display*. Now you can send or read mail.

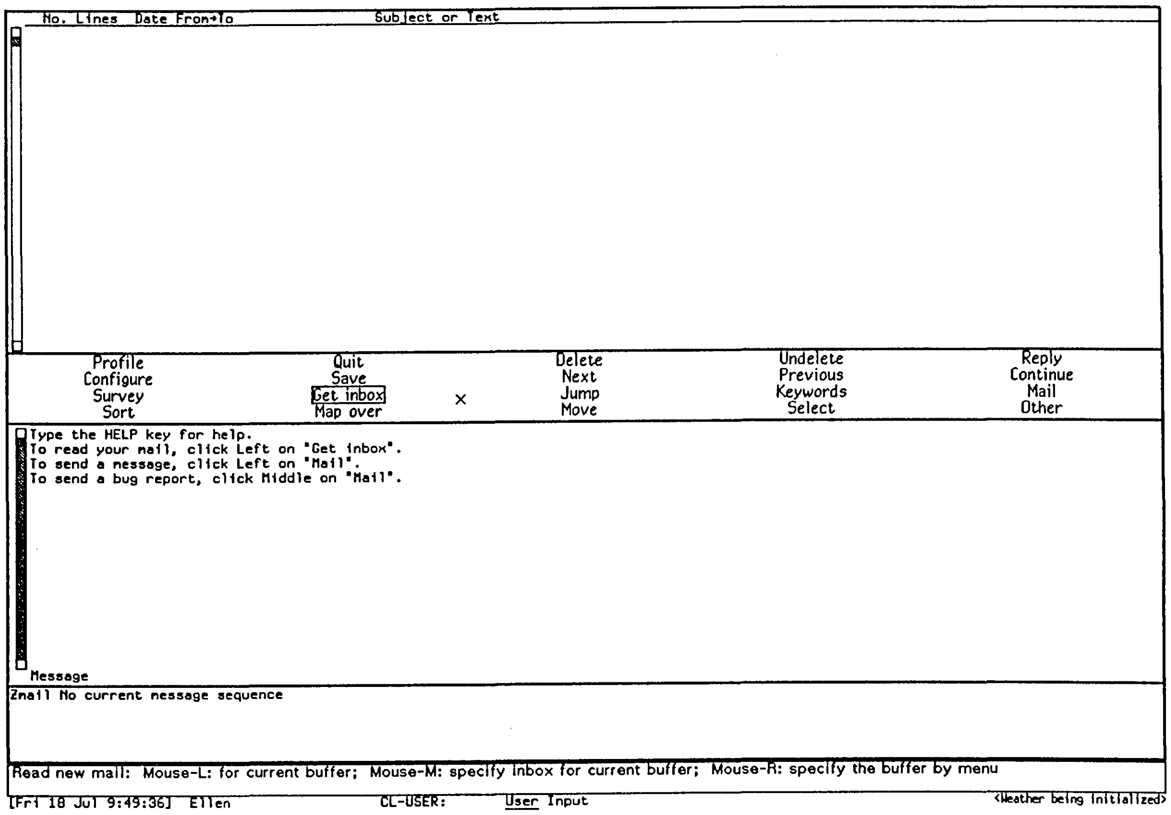
The top-level display consists of four windows: the Summary Window, the Command Menu, the Message Window, and the Minibuffer, which contains the Mode Line. The *Summary Window* displays a line for each message in the current sequence, with an arrow indicating the current message.

The *Command Menu* provides a mouse-sensitive menu of the most useful top-level commands. In Zmail documentation, when we say, for example, "[Get inbox]", we mean the Get inbox command in this menu. Some of these commands (for example, [Delete]) apply only to the current message.

The *Message Window* displays the current message. The message window is an editor buffer.

The *Mode Line* gives status information about Zmail and about the current message, including its properties and keywords.

The top-level display, with a mail file read in, is shown in Figure 12. Command documentation is available online in several forms:

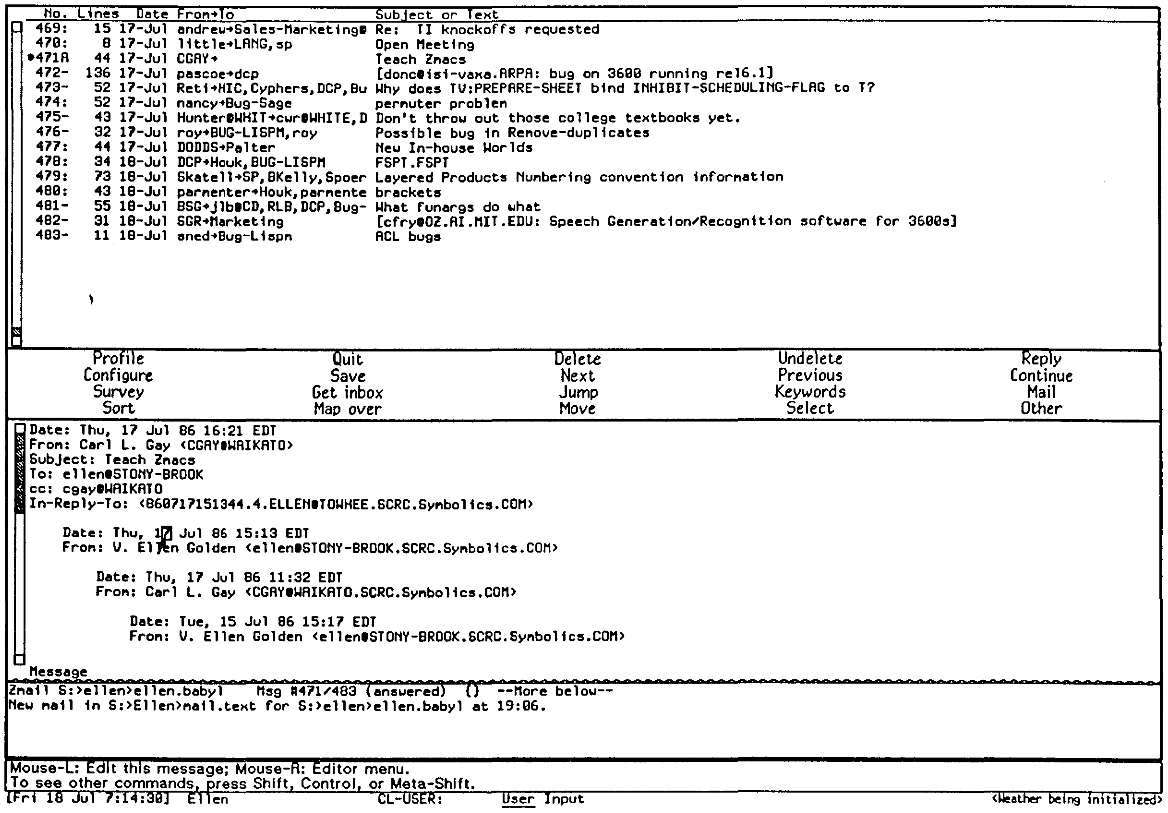


```
 No. Lines  Date From→To                    Subject or Text
```

```
Type the HELP key for help.
To read your mail, click Left on "Get inbox".
To send a message, click Left on "Mail".
To send a bug report, click Middle on "Mail".

Message
Zmail No current message sequence
```

```
Read new mail:  Mouse-L: for current buffer;  Mouse-M: specify inbox for current buffer;  Mouse-R: specify the buffer by menu
[Fri 18 Jul 9:49:36]  Ellen                 CL-USER:        User Input                                   <Weather being initialized>
```

Figure 12. Top-level Display



```
 No. Lines  Date From→To                    Subject or Text
 469:    15 17-Jul andrew→Sales-Marketing@ Re:  TI knockoffs requested
 470:     8 17-Jul little→LANG,sp          Open Meeting
▸471A    44 17-Jul CGAY→                   Teach Zmacs
 472-   136 17-Jul pascoe→dcp              [donc@isi-vaxa.ARPA: bug on 3600 running rel6.1]
 473-    52 17-Jul Reti→HIC,Cyphers,DCP,Bu Why does TV:PREPARE-SHEET bind INHIBIT-SCHEDULING-FLAG to T?
 474:    52 17-Jul nancy→Bug-Sage          permuter problem
 475-    43 17-Jul Hunter@WHIT→cwr@WHITE,D Don't throw out those college textbooks yet.
 476-    32 17-Jul roy→BUG-LISPM,roy       Possible bug in Remove-duplicates
 477:    44 17-Jul DODDS→Palter            New In-house Worlds
 478:    34 18-Jul DCP→Houk,BUG-LISPM      FSPT.FSPT
 479:    73 18-Jul Skatell→SP,BKelly,Spoer Layered Products Numbering convention information
 480:    43 18-Jul parmenter→Houk,parmente brackets
 481-    55 18-Jul BSG→jlb@CD,RLB,DCP,Bug- What funargs do what
 482-    31 18-Jul SGR→Marketing           [cfry@OZ.AI.MIT.EDU: Speech Generation/Recognition software for 3600s]
 483-    11 18-Jul smed→Bug-Lispm          ACL bugs
```

```
Date: Thu, 17 Jul 86 16:21 EDT
From: Carl L. Gay <CGAY@WAIKATO>
Subject: Teach Zmacs
To: ellen@STONY-BROOK
cc: cgay@WAIKATO
In-Reply-To: <860717151344.4.ELLEN@TOWHEE.SCRC.Symbolics.COM>

    Date: Thu, 17 Jul 86 15:13 EDT
    From: V. Ellen Golden <ellen@STONY-BROOK.SCRC.Symbolics.COM>

        Date: Thu, 17 Jul 86 11:32 EDT
        From: Carl L. Gay <CGAY@WAIKATO.SCRC.Symbolics.COM>

            Date: Tue, 15 Jul 86 15:17 EDT
            From: V. Ellen Golden <ellen@STONY-BROOK.SCRC.Symbolics.COM>

Message
Zmail S:>ellen>ellen.babyl    Msg #471/483 (answered)  ()  --More below--
New mail in S:>Ellen>mail.text for S:>ellen>ellen.babyl at 19:06.
```

```
Mouse-L: Edit this message; Mouse-R: Editor menu.
To see other commands, press Shift, Control, or Meta-Shift.
[Fri 18 Jul 7:14:30]  Ellen                 CL-USER:        User Input                                   <Weather being initialized>
```

Figure 13. Top-level Display with Mail File

- Explanations displayed automatically: usually appear below the mode line.
- Mouse documentation line.
- HELP key: provides short command documentation.
- Apropos (m-X): lists commands whose name contains a given string.

See the section "Online Help for Zmail" in *Communicating with Other Users.*

6.1.3 Sending Your Mail

To send a message, click on [Mail], which is displayed in the command menu.

Command	*Meaning*
[Mail] or M (Kbd)	Starts up a window for composing a mail message.
[Mail (M)]	Starts up a window for composing a bug report. You can control the behavior of click middle in your profile. See the variable **zwei:*mail-middle-mode*** in *Communicating with Other Users.*
[Mail (R)]	Calls up a menu of mail sending operations.

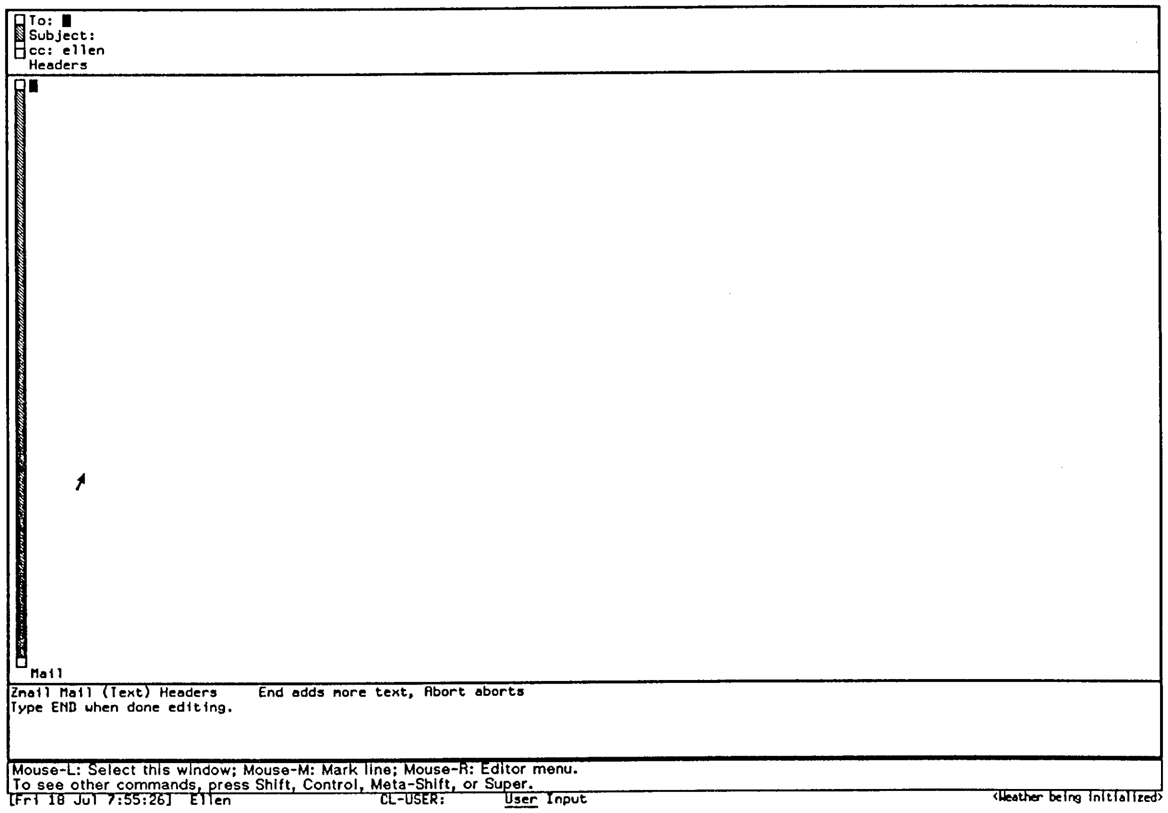
Zmail displays two windows, one for the message headers, and one for the message itself. If you are sending a bug message, information about the software configuration of your machine is automatically added to the message window. (See figure 14)

At this point, the headers window is selected, with the cursor following the word To:. The program is prompting you for the contents of the To: field, which specifies to whom the message is to be sent. Respond by entering a list of one or more user names or mailing lists separated by commas.

If you wish to send someone a carbon copy of the message (which means they also get the message, but are not considered a primary recipient), press RETURN, then type Cc: followed by a list of one or more user names or mailing lists, separated by commas. If you want to save a copy of the message for yourself, include your own name on the Cc: list (or on the To: list).

Use c-N to get down to line containing the word Subject:. Fill in a short subject line for the message. This Subject is used in the summary display of the recipient's mail file. (If you have no Subject: field, the text of the first meaningful line is used.)

To enter the message itself, select the message window by pressing END. The message window is an editor window; you can type in the message using all the commands of the editor. See the section "Zmacs Manual" in *Text Editing and Processing.* The headers window is also an editor window.



```
To: 
Subject:
cc: ellen
Headers

Mail
Zmail Mail (Text) Headers     End adds more text, Abort aborts
Type END when done editing.

Mouse-L: Select this window; Mouse-M: Mark line; Mouse-R: Editor menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Fri 18 Jul 7:55:26]  Ellen            CL-USER:      User Input            <Weather being initialized>
```

Figure 14. Mail Mode Display (One-window Mode)

At any time during editing you can return to the headers window to add or change entries; just click left on the headers window. To get back to the mail window, press END or click left on the mail window.

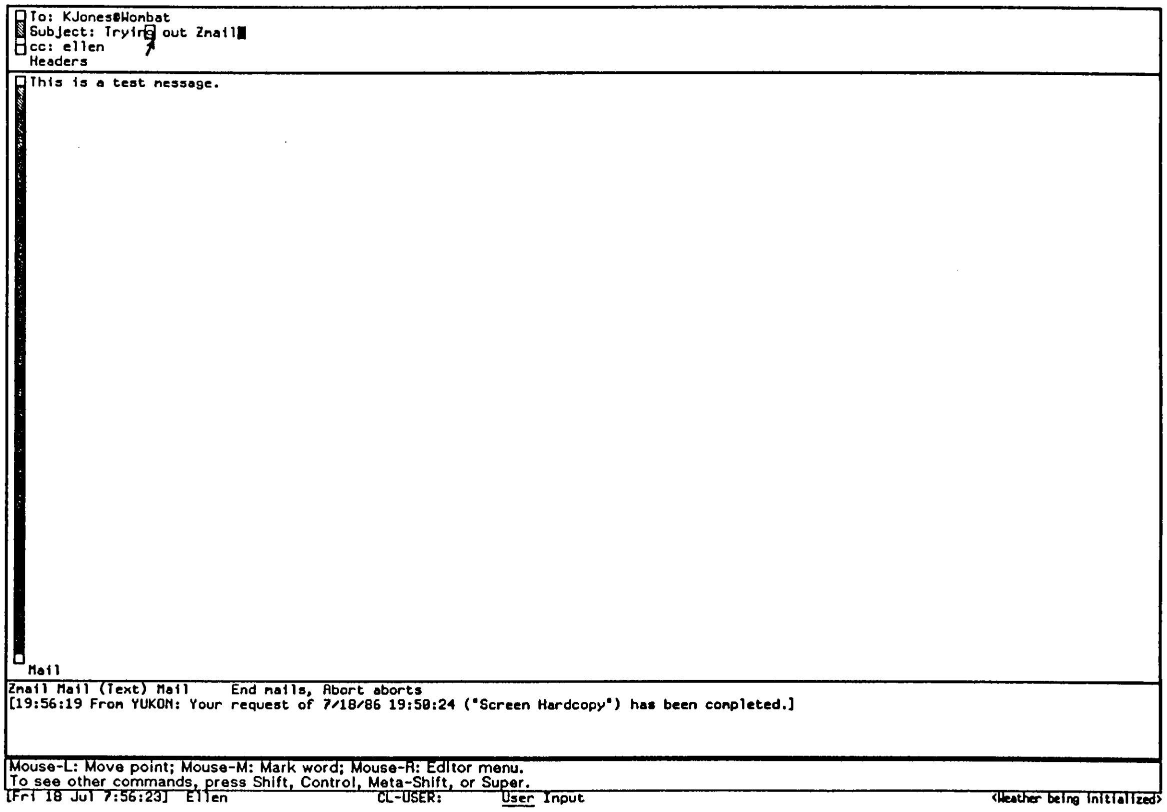
If you change your mind while working on the message and decide that you do not want to send anything, press ABORT, and you return to top level; nothing is sent. If you later decide that you did want to send the message after all, use [Continue]. See the section "Continuing Completed or Aborted Zmail Messages" in *Communicating with Other Users.*

When you are satisfied, press END to send the message. If you are in the headers window, press END twice. See figure 15 for a message about to be sent.

If the message is sent successfully, Zmail displays "Message sent" and returns to top level. If there is a problem, Zmail tells you about it and remains in mail mode. Typical problems are omitting the To: field, trying to send mail to a nonexistent user, or mistyping a user name. Correct the error and resend the message by pressing END twice.

6.1.4 Reading Your Mail

To read your mail, use [Get inbox]; Zmail reads in your primary mail file (containing old mail) and any new mail.



```
To: KJones@Wombat
Subject: Trying out Zmail
cc: ellen
Headers
This is a test message.
Mail
Zmail Mail (Text) Mail      End mails, Abort aborts
[19:56:19 From YUKON: Your request of 7/18/86 19:50:24 ("Screen Hardcopy") has been completed.]
Mouse-L: Move point; Mouse-M: Mark word; Mouse-R: Editor menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Fri 18 Jul 7:56:23]  Ellen                CL-USER:        User Input                    <Weather being initialized>
```

Figure 15. A Message about to be Sent

Command	*Meaning*
[Get inbox] or `G` from the keyboard	Gets the new mail (inbox) for the current buffer. It has no effect when a collection is current.
[Get inbox (M)]	Prompts you for an inbox name for the current buffer.
[Get inbox (R)]	Calls up a menu of possible buffers for which to get the new mail.

For a complete discussion of the [Get Inbox] command: See the section "[Get Inbox] Zmail Menu Item" in *Communicating with Other Users.*

Two files are involved here: your *primary mail file*, which contains messages you have already seen, and your *inbox*, which contains new mail. If you do not have a mail file – as might be the case the first time you run Zmail – the program offers to create one for you. Press `RETURN` to let Zmail create the file, or `ABORT` if for some reason you do not want a mail file. No similar problem with inbox files exists; they are created when needed, and are deleted when Zmail reads your new mail from them.

While an internal data structure used for conversation and reference commands is created, the following message appears in the status line:

`Parsing messages in` *filename*

The parsing required in the creation of reference hash tables is time-consuming for large unparsed files. The appearance of this message notifies you that it is building a reference hash table so that you do not think something is wrong. If you store your mail files in KBIN format, which is already parsed, this wait is eliminated. See the section "Binary Format for Storing Mail Files" in *Communicating with Other Users.*

If you have no new mail, Zmail says so. Otherwise, the summary window starts to scroll as lines appear for new messages, and the first new message is displayed in the message window as the current message.

If the message does not fit entirely in the window, the bottom edge of the window is a jagged line and the words `--more below--` appear in the mode line. When text is off-screen both above and below, both the top and the bottom edge of the window are jagged and the message reads `--more above and below--`; when you reach the final screen of the message, the top edge of the window is jagged and the message reads `--more above--`.

There are several ways to scroll using the keyboard:

To display the next screen of the message

```
SPACE
c-V
SCROLL
```

To go back to the previous screen

```
                    BACKSPACE
                    m-V
                    m-SCROLL
```

To return to the beginning of the current message

```
                    .
                    H
                    c-m-<
```

To use the mouse for scrolling, you can click left on the `--more ...--` messages to scroll forward one screen, or click middle to scroll back one screen. If you click right, you get a menu of four items: [Forward] and [Backward], which move forward and backward by one screen, and [Beginning] and [End], which move to the beginning and the end of the message. For more precise control of scrolling, use the scroll bar in the left margin of the window. See the section "Scrolling", page 10.

6.1.5 What to Do After Reading a Message

Once you have finished reading a particular message, there are several things you might want to do. You might want to read the next new message (if any), you might want to delete the message if it is no longer of value, or you might want to reply to the message.

6.1.5.1 Deleting and Undeleting Messages

After you have finished reading a message, you often want to delete it and move on to the next one. To do this, click on [Delete] or press `D`. This marks the message as deleted – a `D` appears in its summary line – and moves to the next message.

If you change your mind, you can undelete a message; click on [Undelete] or press `U`. This starts at the current message and searches backward for a deleted message, undeletes it, and selects it as the current message. When you delete a message from a mail buffer, the message is not actually removed – it just acquires the property Deleted. You remove the message when you *expunge* the buffer; this happens automatically when you save it, or you can expunge it manually.

6.1.5.2 Moving Among Messages

When you finish reading a message that you do not want to delete, use [Next] to read the next message. To go back to the previous message, use [Previous]. To jump to the first message in the file, use [Previous (M)]; for the last message, use

[Next (M)]. (Note: These commands ignore deleted messages; they actually give you the next undeleted message, previous nondeleted, first nondeleted, and last undeleted.)

To read an arbitrary message, select it from the summary window by clicking left on its summary line. If the summary does not all fit in the window, you might first have to scroll the display using the left-margin scroll bar.

6.1.5.3 Replying to Mail

To reply to the current message, click on [Reply].

Command	*Meaning*
[Reply] or R (Kbd)	Starts up a window to reply to the current message. You can customize the window configuration. See the variable **zwei:*reply-window-mode*** in *Communicating with Other Users.*
[Reply (M)]	Starts up a window to reply to the current message with the message being replied to included. You can control the behavior of click middle in your profile. See the variable **zwei:*middle-reply-mode*** in *Communicating with Other Users.*
[Reply (R)]	Calls up a menu of reply options.

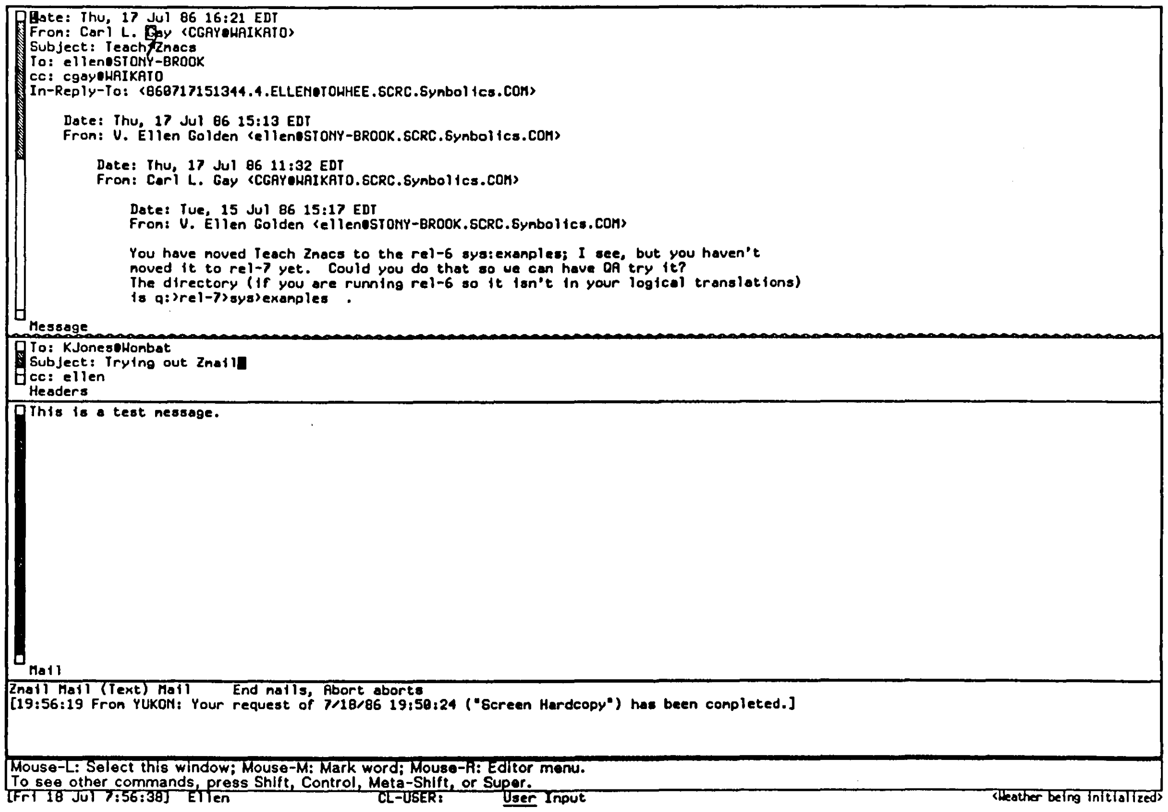
This sets up the screen as three windows: the Message window displays the current message, the Headers window contains the reply headers, and the Mail window is where you write the reply itself. (See Figure 16)

The cursor is in the Mail window, so you can just type in the text of the message, using editor commands to edit what you are typing. To send the message, press END. If you change your mind and do not want to reply, press ABORT. If you want to edit the headers, you can select the Headers window by clicking left on it. These commands are the same as in mail mode. See the section "Sending Your Mail", page 79.

What is special about reply mode is that the reply headers are written automatically. The headers that Zmail writes are the `To:` field, the `CC:` field, the `Subject:` field, and the `In-Reply-To:` field. The `Subject:` field is simply a copy of the original `Subject:`. Defaults for the `To:` and `CC:` fields are provided. Notice the mouse-documentation line. To set up alternate `To:` and `CC:` fields, use [Reply (R)] and choosing from the pop-up menu the combination of `To:` and `CC:` you want. See the section "[Reply] Zmail Menu Item" in *Communicating with Other Users.*

6.1.5.4 Saving the Mail File

When you have finished reading your new mail, you should save your mail file by using [Save]. This expunges deleted messages from the file and then saves it, writing the modified mail file back out to the file system where it is kept until next time.



```
Date: Thu, 17 Jul 86 16:21 EDT
From: Carl L. Gay <CGAY@WAIKATO>
Subject: Teach Zmacs
To: ellen@STONY-BROOK
cc: cgay@WAIKATO
In-Reply-To: <860717151344.4.ELLEN@TOWHEE.SCRC.Symbolics.COM>

    Date: Thu, 17 Jul 86 15:13 EDT
    From: V. Ellen Golden <ellen@STONY-BROOK.SCRC.Symbolics.COM>

        Date: Thu, 17 Jul 86 11:32 EDT
        From: Carl L. Gay <CGAY@WAIKATO.SCRC.Symbolics.COM>

            Date: Tue, 15 Jul 86 15:17 EDT
            From: V. Ellen Golden <ellen@STONY-BROOK.SCRC.Symbolics.COM>

            You have moved Teach Zmacs to the rel-6 sys:examples; I see, but you haven't
            moved it to rel-7 yet.  Could you do that so we can have QA try it?
            The directory (if you are running rel-6 so it isn't in your logical translations)
            is q:>rel-7>sys>examples  .

Message
To: KJones@Wombat
Subject: Trying out Zmail
cc: ellen
Headers
This is a test message.




Mail
Zmail Mail (Text) Mail     End mails, Abort aborts
[19:56:19 From YUKON: Your request of 7/18/86 19:50:24 ("Screen Hardcopy") has been completed.]

Mouse-L: Select this window; Mouse-M: Mark word; Mouse-R: Editor menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Fri 18 Jul 7:56:38] Ellen                CL-USER:      User Input                    <Weather being initialized>
```

Figure 16. Mail Mode Display (Two-window Mode)

If you now wish to leave Zmail, select another program using the `SELECT` key or the System menu.

6.1.6 Getting Fancy with Zmail

Once you have mastered the basics of Zmail, there are many advanced facilities that you can use for composing messages and for organizing your mail files. This section touches on three commonly used facilities. For more detailed information and further suggestions: See the section "Zmail Reference Guide" in *Communicating with Other Users.*

See the section "Using Character Styles in Zmail", page 164.

6.1.6.1 Tagging Messages with Keywords

Zmail allows you to classify and categorize messages by adding keywords to them. Keywords are useful in many ways, among them:

Topic Indicators	Indicate the major topic of the message. If your work involves designing natural language interfaces, for example, you might use keywords such as `dictionary`, `parser`, and `syntax-checker`. The topic indicators you need depend on the sort of messages you get.

Classifiers	Indicate the type of message. For example, you might use keywords such as `bug`, `feature-request`, `documentation-bug`, and `issue` to categorize messages as bug reports, requests for features, reports of documentation bugs, and issues under discussion.
Status Flags	Indicate the status or priority of the message. For example, you might use a keyword such as to-do to flag messages that require you to do something and a keyword such as timing-out to flag messages on which you are awaiting action from other people. You could use `P1`, `P2`, and `P3` to indicate the priority of a message requiring further action.

To add keywords to the current message, click on [Keywords] in the Zmail menu. If you are using keywords for the first time, click right.

[Keywords (R)]	Pops up a highlighted menu of your keywords, in addition to the entry [New] for adding a new keyword. If you have never specified keywords for any messages, the menu contains only three items: [Do It], [Abort], and [New]. Click on [New] and type a keyword. The keyword appears on the menu, highlighted. Click on [Do It] and the keyword appears in braces on the summary line of the message. Keywords are stored in the mail files of the messages they are attached to. You can specify keyword/mail file associations explicitly in your Profile. See the section "Zmail Profile Options" in *Communicating with Other Users.*

Clicking left on [Keywords] adds the last used keyword(s) to the current message.

You can sort your mail files by keywords, to have all the messages on one topic together. See the section "[Sort] Zmail Menu Item" in *Communicating with Other Users.*

Your keywords appear in the menu offered by [Survey] so you can get a list of all the messages with a specific keyword attached to them. See the section "[Survey] Zmail Menu Item" in *Communicating with Other Users.*

For more information about using keywords:

See the section "[Keywords] Zmail Menu Item" in *Communicating with Other Users.*

See the section "Hints for Using Keywords, Mail Collections, and Mail Files" in *Communicating with Other Users.*

6.2 Talking to Other Users

6.2.1 Introduction to Converse

Converse is a facility for communicating interactively with other logged-in users. A message sent with Converse pops up on the screen of the recipient almost instantaneously. The recipient has the choice of replying right in the pop up window, entering Converse to reply, or doing nothing.

The Converse interactive message editor is operated by a window with its own process. Converse keeps track of all of the messages that you have received or sent. The Converse window shows all of the messages that have been sent or received since the machine was cold booted.

Messages sent between you and another user are organized into a *conversation.* Conversations are separated from each other by a thick black line. Within each conversation are all messages, outgoing and incoming, arranged in chronological order, and separated by thin black lines.

You can use Converse to look at conversations, send messages, and receive messages. Converse is built on the Zwei editor, so you can edit your message as you type it, or pick text up and move it around between one message and another, or among messages, files, and pieces of mail.

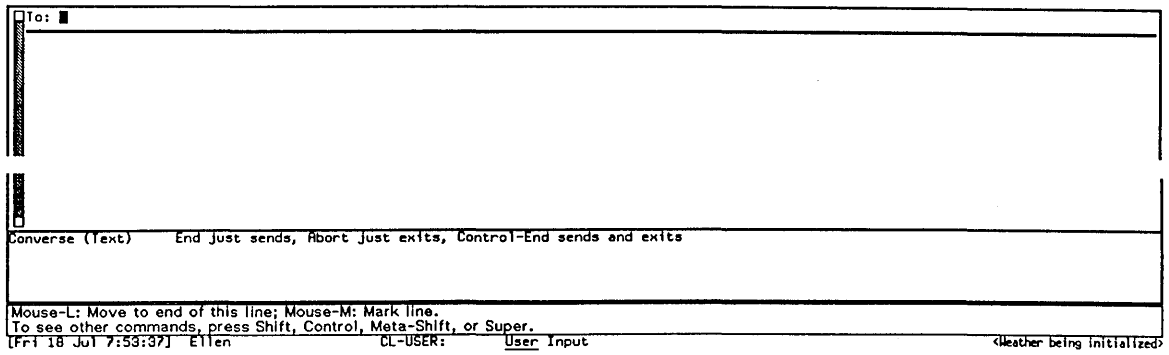
To enter Converse, do one of the following:

- Press `SELECT C`.
- Evaluate **(zl:qsend)**.
- Click on [Select / Converse] in the System menu.
- Answer `C` in the Converse pop-up window when a message arrives.

6.2.2 Using Converse

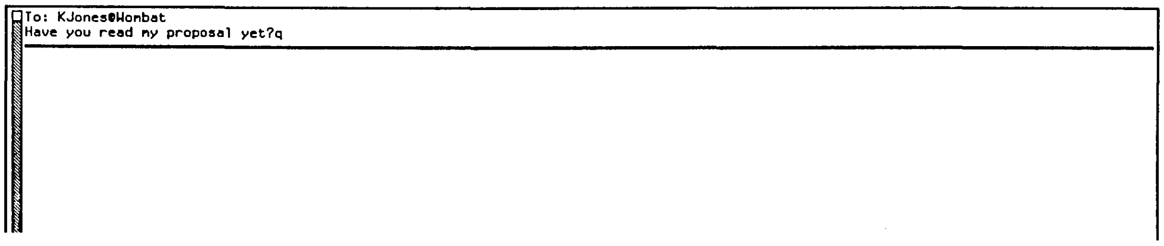
6.2.2.1 Sending and Replying to Messages with Converse

When you enter Converse for the first time, the window is empty except for a blank message at the top of the screen, starting with `To:`. You start a message by filling in a recipient after the `To:`, pressing `RETURN` and then typing the message text. It is not necessary to know what machine the person is using, but if you do know and give the recipient as *name@host* the message is sent considerably faster since it is not necessary to search the namespace to find the machine. To send the finished message, press `END`. When the message has been sent successfully, it appears as part of a conversation. A blank message remains at the top of the screen, and just below that, a heavy black line delimits the message(s) of the



```
To: ▮

Converse (Text)     End just sends, Abort just exits, Control-End sends and exits

Mouse-L: Move to end of this line; Mouse-M: Mark line.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Fri 18 Jul 7:53:37]  Ellen              CL-USER:      User Input                    <Weather being initialized>
```

Figure 17. A Fresh Converse Window



```
To: KJones@Wombat
Have you read my proposal yet?q
```

Figure 18. A Converse Message About to be Sent

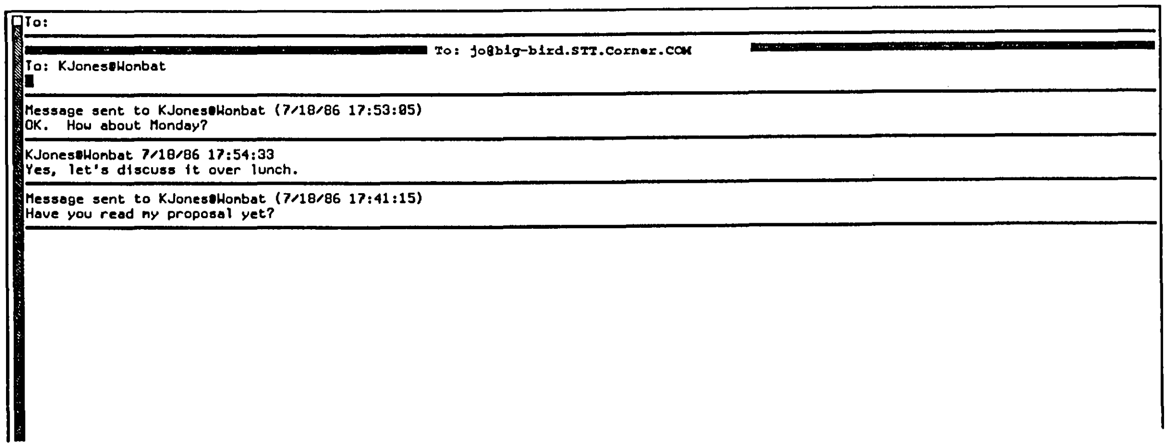
conversation you just started. Just below the heavy black line is another blank message, but this one has the name of the person to whom you sent the message filled in. Below this blank message, separated by a thin black line, the message you just sent appears, with the date and time it was sent.

When the person to whom you sent the message replies, the reply appears in the conversation above the message you sent, and below the blank message. Your cursor is left in the blank message so you can reply easily.

You use regular editor commands to move around in the Converse window. Two commands, specific to Converse, are particularly useful: `c-m-]` (move to next conversation) and `c-m-[` (move to previous conversation).

You exit from Converse by pressing `ABORT` or by selecting another window. You can also press `c-END` when sending a message to send the message and exit from Converse.

To start a conversation, enter Converse, go to the top of the Converse window and fill in the blank message, starting with the `To:` line to specify the new recipient. Finish by pressing `END` to send the message. To send the message and exit Converse, finish by pressing `c-END`.



```
To:
                                   To: jo@big-bird.STT.Corner.COM
To: KJones@Wombat

Message sent to KJones@Wombat (7/18/86 17:53:05)
OK.  How about Monday?

KJones@Wombat 7/18/86 17:54:33
Yes, let's discuss it over lunch.

Message sent to KJones@Wombat (7/18/86 17:41:15)
Have you read my proposal yet?
```

Figure 19. A Converse Conversation

To send a message as part of an existing conversation, find that conversation in Converse and fill in the blank message at the beginning of the conversation, finishing by pressing `END` to send the message, or by pressing `c-END` to send the message and exit Converse.

You do not have to be in the main Converse window to receive messages. Converse will deliver a message to you in any window. Since this might be annoying, you can customize what happens when a message arrives by using the variable **zwei:*converse-mode***. See the section "Customizing Converse", page 106.

When you are in a window other than Converse and a new message arrives, a window pops up at the top of the screen displaying the message. You can respond `R` to type in a reply, `N` (for "no action") to make the message window deexpose, or `C` to enter Converse. Entering Converse has several advantages: you can look over the previous messages in the conversation, and you can use the editor to help you construct a reply.

Converse remembers all messages that you send or receive, even if you did not use the main Converse window to send them or reply to them.

Converse lets you know as soon as a message comes in, by beeping or flashing the screen, and if it is supposed to notify you, it does so without waiting for the main Converse process to wake up. In pop-up mode, if the pop-up message window is already in use, an incoming message causes the message window to beep or flash but not to display the message. This is necessary since only one message at a time should pop up. When the message window is deexposed it is reexposed immediately with the new message in it.

If the main Converse window is exposed, a new message is shown there with its conversation; it is never shown via a notification or a pop-up message window. If the main Converse window is exposed but its process is busy (typically, when it is in the Debugger or in an editor command and waiting for typein), Converse beeps or flashes but does not display the message. You can display the message by clearing the Converse process. You can usually clear the Converse process by pressing ABORT.

6.2.2.2 Converse Commands

Converse has several commands for managing your conversations.

HELP	Displays a summary of Converse commands.
END	Sends the current message. The behavior of this key can be changed by the variable **zwei:*converse-end-exits***.
c-END	Sends the current message and exits from Converse. The behavior of this key can be changed by the variable **zwei:*converse-end-exits***.
ABORT	Exits Converse.
c-M	Mails the current message instead of sending it. This is useful if Converse reports that the person you want to send the message to is not logged in anywhere.
c-m-[	Moves to the previous conversation.
c-m-]	Moves to the next conversation.
Delete Conversation (m-X)	Deletes the current conversation from the Converse window.
Write Buffer (m-X)	Writes the entire Converse buffer (all conversations) to a file. It prompts for a pathname.
Write Conversation (m-X)	Writes only the current conversation to a file. It prompts for a pathname.

Append Buffer (m-X)
: Appends the entire Converse buffer (all conversations) to the end of a file. It prompts for a pathname.

Append Conversation (m-X)
: Appends only the current conversation to the end of a file. It prompts for a pathname.

Regenerate Buffer (m-X)
: Rebuilds the structure of the Converse buffer. This might be necessary if you damage the buffer in some way, for instance by removing one of the black lines separating conversations. Some error messages might ask you to give this command and try again. The message you are currently typing might be lost, but you can prevent this by putting the text on the kill ring by marking it and using m-W before issuing the m-X Regenerate Buffer command.

6.2.2.3 Lisp Listener Commands for Converse

zwei:qsends-off &optional (*gag-message* **t**) *Function*

Sometimes, you might wish not to be interrupted with interactive messages. A function called **zwei:qsends-off** exists for such occasions. If you give it a string argument, *gag-message*, the variable **zwei:*converse-gagged*** is set to this string and the string is returned to anyone who tries to send a message to you. Otherwise, they just get a note saying that you are not accepting messages. **zwei:qsends-on** toggles **zwei:*converse-gagged***.

zwei:qsends-on *Function*

After using **zwei:qsends-off** to notify interactive message senders that you are not accepting messages, **zwei:qsends-on** allows interactive messages to be received again.

chaos:notify-local-lispms &optional *message* &key (*report* **t**) *Function*

Sends *message* to all Lisp Machines at your site based upon information it gets from the namespace database about the Lisp Machines at the local site. *message* should be a string; if it is not provided, the function prompts for a message. Each recipient receives the message as a notification, rather than as an interactive message.

If *report* is **t** (the default), the function reports whether it succeeded or failed to deliver the message to each machine at your site. If *report* is **nil**, it only reports its failures.

zl:qsend &optional *destination message* *Macro*

Sends interactive messages to users on other machines on the network.

destination is normally a string of the form *name@host,* to specify the recipient. If you omit the *@host* part and just give a name, **zl:qsend** looks at all of the Lisp Machines at your site to find any that *name* is logged into; if the user is logged into one Lisp Machine, it is used as the host; if more than one, **zl:qsend** asks you which one you mean. If you leave out *destination* altogether, doing just **(zl:qsend)**, Converse is selected as if you had pressed `SELECT C`.

message should be a string. For example:

```
(qsend kjones@wombat "Want to go to lunch?")
```

If *message* is omitted, **zl:qsend** asks you to type in a message. You should type in the contents of your message and press `END` when you are done.

The input editor is used while you type in a message to **zl:qsend**. So you get some editing power, although not as much as with full Converse (since the latter uses Zwei). See the section "Editing Your Input", page 134. **zl:qsend** predates Converse and is retained for compatibility.

print-sends &optional (*stream* **zl:standard-output**) *Function*

Prints out all messages you have received (but not messages you have sent), in forward chronological order, to *stream*. Converse is more useful for looking at your messages, but this function predates Converse and is retained for compatibility.

zl:qreply &optional *text* *Function*

Sends a reply to the Converse message received most recently. You can supply a string as the text of the message or omit it and let **zl:qreply** prompt for it. It returns a string of the form "*user@host*", indicating the recipient of the message. This function predates Converse and is retained for compatibility.

7. Customizing Genera

7.1 What is Customizing?

When you load a file or set a variable (for example, specifying that your hardcopies are sent to a certain printer, changing the character style of the screen display, or changing the appearance of the command prompt), you alter the default system behavior in your environment for the rest of the time you remain logged in. This type of per-session customization does not remain in effect in your machine after you log out or cold boot. If you load a file or set a variable for an intentionally temporary effect, this is fine.

However, if you decide that you want these changes to be put into effect every time you log in (permanently in your environment), you can save them in an *init file*, thereby instructing the system to automatically execute this sequence of commands every time you log in.

7.2 Init Files

An init file is a Lisp program that gets loaded when you log in; it can be used to set up a personalized environment. An init file contains only Lisp forms. The name depends on the type of file system it is stored on:

3600	lispm-init.lisp
UNIX 4.1	lispm-init.l
UNIX 4.2	lispm-init.lisp
VMS	lispmini.lsp
TOPS-20	lispm-init.lisp
ITS	*name* lispm

A simple init file consists primarily of the **login-forms** and the **setq** special forms. The **login-forms** special form evaluates forms in your init file and arranges for them to be undone when you log out. The **setq** special form sets the value of one or more variables.

Here is an example of a simple init file:

```
; -*- Mode: LISP; Package: USER; Lowercase: T; Patch-file: T -*-

(login-forms
  (setq si:*cp-prompt* 'si:arrow-prompt)
```

```
      zwei:
      (setq text-mode-hook 'auto-fill-if-appropriate)

      (setq si:local-finger-location
            (cond ((y-or-n-p "in your office? ")
                   "340 Domingo x562")
                  (t (format t "~&Where are you? ")
                     (readline query-io)))))

    (si:set-default-hardcopy-device "Echo-Lake")
    (si:set-screen-hardcopy-device "Echo-Lake")
```

In this simple init file, the first **setq** changes the value of the variable that displays the command processor prompt from the default `Command:` to an arrow. The second **setq** specifies that the system automatically fill text that you type in any editor-based activity when appropriate. The third **setq** sets the value of the variable that reports your user ID and on what machine you are logged in to ask you when you log in whether you are in your office, and if not, where you are so that it can send that information to the network namespace database.

The rest of the init file contains two functions that set the default printer for the various commands that hardcopy files and for the `FUNCTION Q` Screen Hardcopy command.

Here is the description of **setq**:

setq *{variable value}...* *Special Form*

Used to set the value of one or more variables. The first *value* is evaluated, and the first *variable* is set to the result. Then the second *value* is evaluated, the second *variable* is set to the result, and so on for all the variable/value pairs. **setq** returns the last value, that is, the result of the evaluation of its last subform. Example:

```
(setq x (+ 3 2 1) y (cons x nil))
```

x is set to **6**, **y** is set to **(6)**, and the **setq** form returns **(6)**. Note that the first variable was set before the second value form was evaluated, allowing that form to use the new value of **x**.

If you do not cold boot your machine after each session, you should arrange for your customizations to be undone when you log out. You do this by using **login-forms**:

login-forms &body *forms* *Special Form*

login-forms is a special form for wrapping around a set of forms in your init file. It evaluates the forms and arranges for them to be undone when you log out.

login-forms always evaluates the forms, even when it does not know how to undo them. For forms that it cannot undo, it prints a warning message.

In the following example, **login-forms** arranges for the base to be reset at logout to 10 (the default) and for **zl-user:bar** either to become undefined or to get its old function definition. It would warn you about **zl-user:quux** being impossible to undo.

```
(login-forms
        (setq-standard-value base 8)
        (setq-standard-value ibase 8)
        (defun bar (x y) (+ x y))
        (quux 3))
```

You can create functions to undo forms that **login-forms** does not recognize. To undo a given form, you put a property on the symbol that is the car of the form to undo. For example, to create a function to undo **zl-user:quux**:

```
(defun (:property quux :undo-function) (form)
  `(undo-quux ,(cadr form)))
```

The value returned by an undo function is a form to be evaluated at logout time.

zl:setq-standard-value is a special form, similar to **setq**, that you should use if you reset any of the variables that control aspects of the Lisp environment (for example, the default base) as opposed to convenience features. See the section "Standard Variables" in *Symbolics Common Lisp.*

Other variables can be set inside **login-forms** using **zl:setq-globally**:

zl:setq-globally &rest *vars-and-vals* *Special Form*

zl:setq-globally should be used with **login-forms** for anything that might be bound while evaluating the **login-forms**.

zl:setq-globally works like **setq** but sets the global values, bypassing any special-variable bindings. **login-forms** knows how to undo this. **zl:setq-globally** is the recommended way to set things in your init file that are not set with **zl:setq-standard-value**.

An example:

```
(login-forms
  (setq-globally zwei:*converse-beep-count* 4))
```

To load individual files from your init file, use the **zl:load** function:

```
(cp:execute-command "show file" "foo.lisp")
(cp:execute-command "show herald" :detailed t)
(cp:execute-command "load system" "mysystem" :compile :always :automatic-answer t)

(load "SYS: LISP; MY-PROJECT")
(load "Tuna:>kjones>examples>decorate")
(load "vixen://usr//kjones//tools//toolkit")
```

See the function **cp:execute-command**, page 102.

The first sample form loads a file using its logical pathname; the second form loads a file from a LMFS using its physical pathname. The third form loads a file from a Unix system in the appropriate syntax (the slashes are doubled).

7.3 How to Create an Init File

The easiest way to create an init file is by copying the sample init file shown here and then building on it, or by copying someone else's init file. Often you acquire customizations that you find out about from people who have been using Genera longer than you.

7.4 Useful Customizations to Put in Your Init File

The number and kinds of customizations you can put in your init file is limited only by your imagination. This section offers some suggestions that many users have found useful, but it is by no means an exhaustive list.

7.4.1 Adjusting Console Parameters

si:*kbd-auto-repeat-enabled-p* *Variable*

Controls whether or not keys repeat if held down (auto-repeat). The default is **nil**, meaning that holding keys down does not cause repetition. It can be set using **setf**:

```
(setf si:*kbd-auto-repeat-key-enabled-p* t)
```

Setting **si:*kbd-auto-repeat-key-enabled-p*** to **t** turns on auto-repeat. You can set the length of time a key must be held down before it starts to repeat with **si:*kbd-auto-repeat-initial-delay***.

Controls how long you must hold down a key before auto-repetition starts, in sixtieths of a second. The default is 42, which is between half and three-quarters of a second. You can adjust it using **setf**.

si:*kbd-auto-repeat-initial-delay* *Variable*

Controls how long you must hold down a key before auto-repetition starts, in sixtieths of a second. The default is 42, which is between half and three-quarters of a second. You can adjust it using **setf**.

si:set-auto-repeat-p *key* &optional (*state* **t**) *Function*

Allows you to specify keys that should not auto-repeat even if auto-repeat is enabled. By default all keys can auto-repeat except for FUNCTION, SELECT, NETWORK, ABORT, SUSPEND, and RESUME. For example,

```
(si:set-auto-repeat-p #\Square nil)
```

turns off auto-repetition for the SQUARE key. You can make SQUARE auto-repeat again by setting it back to **t**.

tv:screen-brightness *main-screen-mixin* *Function*

Returns the brightness of the screen as a floating point number between 0 and 1. **(tv:screen-brightness tv:main-screen)** may be set in your init file using **zl:setf** to adjust the screen brightness. Console hardware varies slightly so you must experiment to find the value that suits you best. One technique for doing this is to adjust the brightness using LOCAL-B and LOCAL-D until it is to your liking. Then use **(tv:screen-brightness tv:main-screen)** to find that value. For example:

```
(tv:screen-brightness tv:main-screen) returns 0.43307087
```

Then in your init file you place the form

```
(setf (tv:screen-brightness tv:main-screen) 0.43307087)
```

and each time you log in with your init file the screen brightness is automatically set to that value.

tv:*dim-screen-after-n-minutes-idle* *Variable*

Controls the length of time a console must be idle before its screen dims. You can set this in your init file to adjust the length of time it takes the screen dimmer to activate. The default is 20 minutes. Setting it to **nil** disables the screen dimmer entirely.

tv:*screen-dimness-percent* *Variable*

Controls the brightness value of the screen when it is dimmed. You can set this in your init file to adjust the dimness of the screen. The default is 0, meaning black. 100 is bright. If you want a number that will leave the screen very dim but visible, the value will vary with your particular hardware. Experiment to find a good setting, starting with 50.

sys:console-volume &optional (*console* **sys:*slb-main-console***) *Function*

Returns the current volume setting for the console, which is a number between 1.0 (loudest) and 0 (softest). The console volume can be changed with **setf**, as in the example:

```
(cl:setf (sys:console-volume) 0.5)
```

Starting up Zmail in the Background

You can start up Zmail from your init file by using the function **zwei:preload-zmail**. See the function **zwei:preload-zmail**, page 98.

zwei:preload-zmail &rest *files* *Function*

Starts up Zmail, loading in *files*.

```
(zwei:preload-zmail "wombat:>kjones>mail.text")
```

This gets the mail loading operation underway while you are doing something else.

There are two keyword options to **zwei:preload-zmail**:

:find-file	Find the file and load it in for processing.
:examine-file	Finds the file and reads it into Zmail but in read only mode.

As an example, the following form can be included in your LISPM-INIT to preload several mail files into Zmail with some of them being read only:

```
(zwei:preload-zmail '(:find-file "y:>palter>mailboxes>palter.xmail")
                    '(:find-file "y:>palter>mailboxes>reminders.xmail")
                    '(:examine-file "y:>palter>mailboxes>junk.xmail")
                    '(:examine-file "y:>palter>mailboxes>videotech-digest.xmail"
                    '(:hang-when-deexposed t)
                    '(:hang-when-deexposed nil))
```

The last two operations in the above form cause the Zmail background to stop after reading the mail files in question without parsing the contained messages. The background parsing will commence as soon as Zmail is selected. (If the last two operations had been placed first, Zmail would not preload anything until it was first selected.)

7.4.2 Customizing the Command Processor

You can change the command processor's mode, prompt, and special characters, and you can customize the display of the prompt and help messages. Usually you customize the command processor by setting special variables. You might want to do this in your init file, inside a **login-forms** special form.

Whenever you change the command processor's mode, prompt, or other characteristics, you set its state for all Lisp Listeners and **zl:break** loops. You cannot put the command processor into one mode in one Lisp listener and another mode in another.

If you change the command processor's mode or prompt, or if you turn the command processor on or off, the change takes place immediately in that Lisp Listener or **zl:break** loop but not in others that are waiting for input. For example, suppose you use the Set Command Processor command in a **zl:break** loop to change the prompt and dispatch mode. These changes do not take effect in a Lisp Listener that is waiting for input until you execute a command or form or you press ABORT there.

7.4.2.1 Setting the Command Processor Mode

The command processor *mode* determines how input is treated. Following are the four modes and their meanings:

:form-only	All input is treated as a Lisp form.
:command-only	All input is treated as a command invocation.
:form-preferred	Input is treated as a Lisp form unless you precede it by a command dispatch character. In this case it is treated as a command invocation. By default, the command dispatch character is a colon.
:command-preferred	Input is treated as a command invocation if it begins with an alphabetic character. Input is treated as a Lisp form if it is does not begin with an alphabetic character or if you precede it by a form dispatch character. By default, the form dispatch character is a comma.

You can set the command processor mode for Lisp Listeners and **zl:break** loops in two ways:

1. Use the Set Command Processor command. The first argument to this command is the dispatch mode. See the section "Set Command Processor Command", page 254.

2. Set the value of the special variable **cp:*dispatch-mode***.

cp:*dispatch-mode* *Variable*

The current command processor dispatch mode in Lisp Listeners and **zl:break** loops; a keyword. Possible values are **:form-only**, **:form-preferred**, **:command-only**, and **:command-preferred**. For the meanings of these values: See the section "Setting the Command Processor Mode", page 99. The default is **:command-preferred.**

The default dispatch mode for **cp:read-command-or-form** is the value of **cp::*default-dispatch-mode*.**

7.4.2.2 Setting the Command Processor Prompt

You can set the command processor prompt for Lisp Listeners and **zl:break** loops in two ways:

1. Use the Set Command Processor command. The second argument to this command is a string to be displayed as the prompt. See the section "Set Command Processor Command", page 254.

2. Set the value of the special variable **cp:*prompt*.**

cp:*prompt* *Variable*

A prompt option for displaying the current command processor prompt in Lisp Listeners and **zl:break** loops. The value of this variable is passed to the input editor as the value of the **:prompt** option. The value can be **nil**, a string, a function, or a symbol other than **nil** (but not a list): See the section "Displaying Prompts in the Input Editor" in *Reference Guide to Streams, Files, and I/O.*

The default is "Command: ". If the value is **nil** or the empty string, no prompt is displayed. If the value is **si:arrow-prompt**, an arrow is displayed as the prompt.

The default prompt for **cp:read-command** and **cp:read-command-or-form** is the value of **cp::*default-prompt*.**

7.4.2.3 Setting Command Processor Special Characters

You can change the command and form dispatch characters by setting the special variables ***cp:*command-dispatchers*** and **cp::*form-dispatchers*.**

cp::*command-dispatchers* *Variable*

A list of characters that precede commands, distinguishing them from input to the Lisp interpreter, when the command processor is in **:form-preferred** mode. The default is **(#/:).**

cp::*form-dispatchers* *Variable*

A list of characters that precede Lisp forms, distinguishing them from commands, when the command processor is in **:command-preferred** mode. (These characters are needed only when the Lisp form begins with an alphabetic character.) The default is **(#/,).**

7.4.2.4 Customizing Command Processor Display

By setting special variables, you can control the action the command processor takes when you type a blank line and how it displays the screen when you ask for help.

cp:*blank-line-mode* *Variable*

A keyword that determines what action the command processor takes when you type a blank line in Lisp Listeners and **zl:break** loops:

:reprompt	Redisplay the prompt, if any. This is the default.
:beep	Beep.
:ignore	Do nothing.

The default blank line mode for **cp:read-command** and **cp:read-command-or-form** is the value of **cp::*default-blank-line-mode***.

cp::*typeout-default* *Variable*

A keyword that determines how the command processor prints help messages. Possible values are those acceptable as the first argument to the **:start-typeout** message to interactive streams:

:insert	The help message, like a notification, is inserted before the current input.
:overwrite	The help message is inserted before the current input, but the next time an **:insert** or **:overwrite** operation is done, this message is overwritten. This is the default.
:append	The help message appears after the current input, which is reprinted after the help message.
:temporary	The help message appears after the current input and disappears when you type the next character.
:clear-window	The window is cleared and the help message appears at the top.

For more information: See the method **(:method si:interactive-stream :start-typeout)** in *Reference Guide to Streams, Files, and I/O.*

7.4.3 Calling Command Processor Commands From Your Init File

If you want to put command processor commands in your init file, you can do so using the function **cp:execute-command**:

cp:execute-command *command-name* &rest *command-arguments* *Function*

Invokes a command processor command from within a program.

command-name
: Symbol or string naming the command to invoke; if a string, it must be in the command table to which **cp:*command-table*** is currently bound.

command-arguments
: Positional and keyword arguments to the named command.

Examples:

```
(cp:execute-command "show file" "test-data.text")

(cp:execute-command 'si:com-load-system "unifier"
                    :condition :always :automatic-answer t)
```

For an overview **cp:execute-command** and related facilities: See the section "Overview of Basic Command Facilities" in *Programming the User Interface.*

7.4.4 Zmacs Customization in Init Files

You can set Zmacs parameters in your init file also. This section gives you some guidelines for how to set different types of parameters. For information about the available features: See the section "Zmacs Manual" in *Text Editing and Processing*.

7.4.4.1 Setting Editor Variables

The forms described show how to set Zmacs variables (the kind that Set Variable (m-X) sets).

To set these variables, which are symbol macros, you must use the **zl:setf** macro. For a description of symbol macros: See the section "Symbol Macros" in *Symbolics Common Lisp.* For a description of the **zl:setf** macro: See the macro **zl:setf** in *Symbolics Common Lisp.*

Ordering Buffer Lists

```
(SETF ZWEI:*SORT-ZMACS-BUFFER-LIST* NIL)
```

This displays the list of buffers in the order the buffers were created rather than in the order they were most recently visited.

Putting Buffers Into Current Package

```
(SETF ZWEI:*DEFAULT-PACKAGE* NIL)
```

This puts buffers created with c-X B (Select Buffer) into whatever package is current; the default is to put them in the **user** package.

Setting Default Major Mode

```
(SETF ZWEI:*DEFAULT-MAJOR-MODE* ':TEXT)
```

This sets the default major mode to Text Mode for buffers with no Mode attribute and no major mode deducible from the file type; the default is Fundamental Mode.

Setting Find File Not to Create New Files

```
(SETF ZWEI:*FIND-FILE-NOT-FOUND-IS-AN-ERROR* T)
```

This beeps and prints an error message when you give c-X c-F (Find File) the name of a nonexistent file. The default prints (New File) and creates an empty buffer, which when saved by c-X c-S (Save File) creates the file that was nonexistent.

Setting Goal Column for Real Line Commands

```
(SETF ZWEI:*PERMANENT-REAL-LINE-GOAL-XPOS* 0)
```

This moves subsequent c-N and c-P (Down Real Line and Up Real Line) commands to the left margin, like doing c-0 c-X c-N (Set Goal Column to zero).

Fixing White Space for Kill/Yank Commands

```
(SETF ZWEI:*KILL-INTERVAL-SMARTS* T)
```

This tells the killing and yanking commands optimize white space surrounding the killed or yanked text.

7.4.4.2 Key Bindings

To bind keys, you first define the comtab in which to put the binding. For example, ***standard-comtab*** and ***standard-control-x-comtab*** define features of all Zwei-based editors; ***zmacs-comtab*** and ***zmacs-control-x-comtab*** define features that are Zmacs-specific.

White Space in Lisp Code

```
ZWEI:(SET-COMTAB *STANDARD-CONTROL-X-COMTAB*
                 '(#\SP COM-CANONICALIZE-WHITESPACE))
```

This defines c-X SPACE as a command that makes the horizontal and vertical white space around point (or around mark if given a numeric argument or immediately after a yank command) conform to standard style for Lisp code.

c-m-L on the SQUARE Key

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
                 '(#\SQUARE COM-SELECT-PREVIOUS-BUFFER))
```

This defines the SQUARE key to do the same thing as c-m-L. This key binding is placed in ***zmacs-comtab*** rather than ***standard-comtab*** since buffers are a feature of Zmacs, not of all Zwei-based editors.

Edit Buffers on c-X c-B

```
ZWEI:(SET-COMTAB *ZMACS-CONTROL-X-COMTAB*
                 '(#\c-B COM-EDIT-BUFFERS))
```

This makes c-X c-B invoke Edit Buffers rather than List Buffers. This key binding is placed in ***zmacs-control-x-comtab*** rather than ***standard-control-x-comtab*** since buffers are a feature of Zmacs, not of all Zwei-based editors.

Edit Buffers on m-X

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
                 ()
                 (MAKE-COMMAND-ALIST '(COM-EDIT-BUFFERS)))
```

This makes Edit Buffers available on m-X in Zmacs (by default it is only available on c-m-X).

m-. on m-(L)

```
ZWEI:(SET-COMTAB *ZMACS-COMTAB*
                 '(#\m-MOUSE-L COM-EDIT-DEFINITION))
```

This makes clicking the left mouse button while holding down the META key do what m-. does. Invoking this command from the mouse is convenient when you specify the name of the definition to be edited by pointing at it rather than typing it.

7.4.4.3 Setting Mode Hooks

Each major mode has a *mode hook*, a variable which, if bound, is a function that is called with no arguments when that major mode is turned on.

Electric Shift Lock in Lisp Mode

```
(SETF ZWEI:LISP-MODE-HOOK 'ZWEI:ELECTRIC-SHIFT-LOCK-IF-APPROPRIATE)
```

This tells Lisp major mode to turn on Electric Shift Lock minor mode unless the buffer has a Lowercase attribute. The effect is that by default Lisp code is written in upper case.

Auto Fill in Text Mode

```
(SETF ZWEI:TEXT-MODE-HOOK 'ZWEI:AUTO-FILL-IF-APPROPRIATE)
```

This tells Text major mode to turn on Auto Fill minor mode unless the buffer has a Nofill attribute. The effect is that by default lines of text are automatically broken by carriage returns when they get too wide.

7.4.5 Customizing the Input Editor

To change the behavior of the yank system, use **login-forms** and **zl:setq-globally** to set the following Lisp internal variables in your init file.

Alternatively, you can set them with Set Variable (m-X); when Set Variable prompts you for a variable name, supply the name given in each of the following descriptions.

zwei:*history-menu-length* *Variable*

The maximum number of history elements displayed. Default is 20.

History Menu Length is the name to use with Set Variable (m-X).

zwei:*history-yank-wraparound* *Variable*

Determines what happens after m-Y runs off the end of a history or m- - m-Y runs off the beginning of a history. Default is **t**.

Value	*Meaning*
t	m-y wraps around to the other end of the history. For example, after m-Y yanks the oldest element in the history, it returns to the top of the history and yanks the newest element.
nil	m-Y does not wrap around to the other end of the history. Instead, the 3600 flashes (the LM-2 beeps).

History Yank Wraparound is the name to use with Set Variable (m-X).

zwei:*history-rotate-if-numeric-arg* *Variable*

Determines what happens when c-Y or c-m-Y is given after m-Y. Default is **nil**.

Value	*Meaning*
t	A numeric argument to c-Y or c-m-Y is measured from the origin, not the newest element in the history. The origin is always element #1. All other elements are numbered relative to the origin. Elements that are newer than the origin are assigned negative numbers, in ascending order with their distance from the origin.
nil	A numeric argument to c-Y or c-m-Y is measured from the the newest history element, not the origin. However, c-Y or c-m-Y given *without* an argument yanks the element at the origin; thus, the origin has meaning only when you use a top-level command without an argument. When you display a history, its elements are numbered from 1 on and the origin is indicated with a pointer.

History Rotate If Numeric Arg is the name to use with Set Variable (m-X).

See the section "The Input Editor Program Interface" in *Reference Guide to Streams, Files, and I/O.*

7.4.6 Customizing Converse

The following variables allow you to customize Converse's behavior. You can set them in your init file.

zwei:*converse-mode* *Variable*

Controls what happens when an interactive message arrives. It should have one of the following values:

:pop-up	(This is the default.) A message window pops up at the top of the screen, displaying the message. You are asked to type R (for Reply), N (for Nothing), or C (for Converse). If you type R, you can type a reply to the message inside the message window. When you type END, this reply is sent back to whomever sent the original message to you, and the pop-up message window window disappears. If you type N, the message window disappears immediately. If you type C, the Converse window is selected. The input editor is used while you reply to a message in the

pop-up message window, so you get some editing power, although not as much as with full Converse (since the latter uses Zwei).

:auto The Converse window is selected. This is the window that shows you all of your conversations, letting you see everything that has happened, and letting you edit your replies with the full power of the Zwei editor. With this window selected, you can reply to the message that was sent, send new messages, participate in other conversations, or edit and write out messages or conversations. You can exit with c-END or ABORT (c-END sends a message and exits; ABORT just exits), or you can select a new window by any of the usual means (such as the FUNCTION or SELECT keys).

:notify A notification is printed, telling you that a message arrived and from whom. If you want to see the message, enter Converse by pressing SELECT C. There you can read the message and reply if you want to.

:notify-with-message
A notification is printed, which includes the entire contents of the message and the name of the sender. If you want to reply, you can enter Converse.

zwei:*converse-append-p* *Variable*

If the value is **nil** (the default), a new message is prepended to its conversation. If the value is not **nil**, a new message is appended to its conversation. **print-sends** is not affected by this variable; it always displays messages in forward chronological order.

zwei:*converse-beep-count* *Variable*

The value is the number of times to beep or flash the screen when a message arrives. The default value is two. Beeping or flashing occurs only if the Converse window is exposed or if the value of **zwei:*converse-mode*** is **:pop-up** or **:auto**. (Otherwise, notification tells you about the message and includes the usual beeping or flashing.)

zwei:*converse-end-exits* *Variable*

Controls the behavior of END and c-END. If **zl-user:*converse-end-exits*** is set to **nil**, the default, END sends the message and you remain in Converse. c-END sends the message and exits Converse. Setting **zl-user:*converse-end-exits*** to **t** reverses this, so that c-END sends the message and remains in Converse and END sends and exits.

7.4.7 Customizing Hardcopy Facilities

You can specify the printer you want to use for hardcopying files and screen images in your init file.

There are two variables that determine which printer is used for a hardcopy request:

hardcopy:*default-text-printer* *Variable*

hardcopy:*default-text-printer* is a variable whose value is the printer to be used for printing text files, that is a printer object. Its initial value is determined from the printer slot in the namespace object for your machine, or if your machine does not specify a printer, from the namespace object for your site.

hardcopy:*default-bitmap-printer* *Variable*

hardcopy:*default-bitmap-printer* is a variable whose value is the printer to be used for printing screen hardcopy, that is a printer object. Its initial value is determined from the bitmap printer slot in the namespace object for your machine, or if your machine does not specify a bitmap printer, from the namespace object for your site.

These variables can be set with the following two functions:

hardcopy:set-default-text-printer *name* *Function*

hardcopy:set-default-text-printer specifies the printer to be used for all of the hardcopy commands except the screen copy command. *name* is a string specifying the device name. This is the real name of the printer, its **name** attribute not its **pretty-name**. For example:

```
(login-forms
  (hardcopy:set-default-text-printer "caspian-sea"))
```

`caspian-sea` is the real name of the printer whose pretty name is `Caspian Sea`. (The valid set of device names are the **printer** objects in your namespace database.)

hardcopy:set-default-bitmap-printer *name* *Function*

hardcopy:set-default-bitmap-printer specifies the printer to be used for screen copies (by the `FUNCTION Q` command). *name* is a string specifying the device name. This is the real name of the printer, its **name** attribute not its **pretty-name**. For example:

```
(login-forms
  (hardcopy:set-default-bitmap-printer "caspian-sea"))
```

`caspian-sea` is the real name of the printer whose pretty name is `Caspian Sea`. (The valid set of device names are the **printer** objects in your namespace database.)

You can specify your preferred character styles for each printer in your init file by setting **hardcopy:*hardcopy-default-character-styles***.

hardcopy:*hardcopy-default-character-styles* *Variable*

hardcopy:*hardcopy-default-character-styles* is a variable whose value is an association list where each element specifies a device and a set of keyword/value pairs designating the character style. The keywords are **:body-character-style** and **:heading-character-style**.

For example:

```
(login-forms
  (setq hardcopy:*hardcopy-default-character-styles*
        '(("Itasca" :body-character-style (:fix :roman :small))
          ("Caspian Sea" :body-character-style (:fix :roman :normal)))))
```

in your init file will specify fixed width small-sized roman as the default character style for the printer Itasca and fixed width normal-sized roman as the default character style for the printer Caspian Sea. The value of **hardcopy:*hardcopy-default-character-styles*** is merged with the default style for the printer, so if the printer is using a fixed width normal sized roman and you want it larger, you only need to specify **(nil nil :larger)**. See the section "Character Styles" in *Converting to Genera 7.0.*

7.4.8 Censoring Fields for lispm-finger and name Services

You might prefer to keep certain fields of information private, and prevent those fields from being returned by the **lispm-finger** and **name** protocol servers.

You can censor the information returned by those servers by pushing recognized keywords onto one or both of the following lists: **neti:*finger-fields-to-suppress*** and **neti:*finger-fields-to-suppress-for-untrusted-hosts***.

The recognized keywords include:

:software-info
:hardware-info
:whois
:project
:supervisor
:work-address
:work-phone
:home-address
:home-phone

neti:*finger-fields-to-suppress* *Variable*

This variable is a list of keywords that should be censored for the **lispm-finger** and **name** servers. Use **push** to add others to the list. The default value is **nil.**

For a list of recognized keywords: See the section "Censoring Fields for **lispm-finger** and **name** Services", page 109.

neti:*finger-fields-to-suppress-for-untrusted-hosts* *Variable*

This variable is a list of keywords that should be censored for the **lispm-finger** and **name** servers, for untrusted hosts only. Use **push** to add others to the list. The default value is **nil.**

For a list of recognized keywords: See the section "Censoring Fields for **lispm-finger** and **name** Services", page 109.

7.5 Logging in Without Processing Your Init File

You might want to log in and work in the standard default system environment, that is, without having your init file set up your usual customizations. Perhaps you want to test a program of yours in the standard environment or try a new system feature in an unpolluted environment. Log in this way:

```
Login username :init file none
```

to tell the Login command that you do not want your init file automatically loaded.

7.6 Customizing Zmail

The Profile command allows you to customize Zmail by setting various display and command options to your personal taste. You can set an option temporarily or permanently, the latter by saving the option in your *Zmail Profile.*

Classes of options you can set include the following:

- Format used for hardcopies of messages
- Mail-file attributes
- Lists of mail files and other objects that Zmail knows about at startup
- Associations between certain objects
- (M) actions for many top-level commands

- Screen configurations
- Default actions taken when reading, sending, replying to, or forwarding mail
- Command Tables

Customizing is done in *profile mode*, entered by clicking on [Profile] in the command menu at top level. The profile mode display (Figure 20) shows the text of your profile and the current settings of various options.

Setting and Saving Zmail Options

Option settings are stored in eight distinct places:

1. *Your mind:* your conception of how the options should be set.
2. *The Zmail environment:* the way the options are actually set at the moment.
3. *The defaults:* the way the options are actually set before you alter them.
4. *The editor buffer:* the in-memory buffer of your profile.
5. *The source version of your profile:* on disk.
6. *The compiled version of your profile:* also on disk.
7. *Mail buffers:* options associated and stored with the individual mail buffers.
8. *Mail files:* options associated with a mail buffer saved as a file.

Enter Profile Mode by clicking on [Profile] in the Zmail menu. The simplest way to use profile mode is:

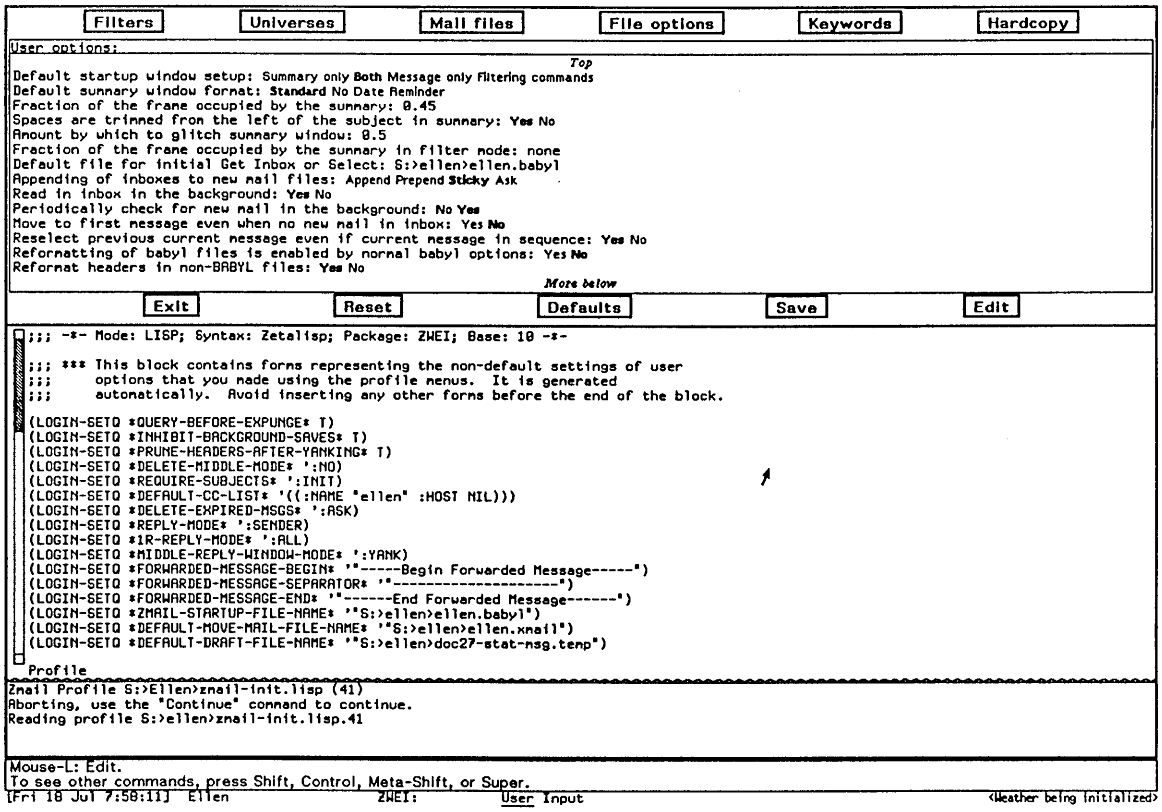
1. Make the changes you want using the menu items or user options window, two regions of the display indicated in Figure 20. For a list of the various options and what they mean: See the section "Zmail Profile Options" in *Communicating with Other Users.*

2. Click on [Exit] to leave profile mode. Check to see that you like your changes.

3. To save your changes, reenter profile mode and click on [Save]. Before you do this for the first time, use [Save (M)] and press RETURN to the question Zmail asks. This specifies that you want your file compiled, which makes it load and run faster. Answer *yes* to any questions about inserting changes or recompiling your file. At this point Lisp code corresponding to your option

settings will be stored in your profile. Options changed using [File options] or [Keywords] are stored in the individual mail buffers and must be saved using [Save] on the top-level command menu.

What [Save] actually does is move option settings from the environment (where you altered them in the first step) to the editor buffer, then from the editor buffer to the source copy of your init file, and finally from the source file to the compiled file (by recompiling). You can also move option settings one step at time, by using [Reset] and [Default], and the menu options available by using [Save].



```
User options:
                                                            Top
Default startup window setup: Summary only [✓ Both] Message only Filtering commands
Default summary window format: [✓ Standard] No Date Reminder
Fraction of the frame occupied by the summary: 0.45
Spaces are trimmed from the left of the subject in summary: [✓ Yes] No
Amount by which to glitch summary window: 0.5
Fraction of the frame occupied by the summary in filter mode: none
Default file for initial Get Inbox or Select: S:>ellen>ellen.babyl
Appending of inboxes to new mail files: Append Prepend [✓ Sticky] Ask
Read in inbox in the background: [✓ Yes] No
Periodically check for new mail in the background: No [✓ Yes]
Move to first message even when no new mail in inbox: Yes [✓ No]
Reselect previous current message even if current message in sequence: [✓ Yes] No
Reformatting of babyl files is enabled by normal babyl options: Yes [✓ No]
Reformat headers in non-BABYL files: [✓ Yes] No
                                                            More below
```

```
;;; -*- Mode: LISP; Syntax: Zetalisp; Package: ZWEI; Base: 10 -*-

;;; *** This block contains forms representing the non-default settings of user
;;;     options that you made using the profile menus.  It is generated
;;;     automatically.  Avoid inserting any other forms before the end of the block.

(LOGIN-SETQ *QUERY-BEFORE-EXPUNGE* T)
(LOGIN-SETQ *INHIBIT-BACKGROUND-SAVES* T)
(LOGIN-SETQ *PRUNE-HEADERS-AFTER-YANKING* T)
(LOGIN-SETQ *DELETE-MIDDLE-MODE* ':NO)
(LOGIN-SETQ *REQUIRE-SUBJECTS* ':INIT)
(LOGIN-SETQ *DEFAULT-CC-LIST* '((:NAME "ellen" :HOST NIL)))
(LOGIN-SETQ *DELETE-EXPIRED-MSGS* ':ASK)
(LOGIN-SETQ *REPLY-MODE* ':SENDER)
(LOGIN-SETQ *1R-REPLY-MODE* ':ALL)
(LOGIN-SETQ *MIDDLE-REPLY-WINDOW-MODE* ':YANK)
(LOGIN-SETQ *FORWARDED-MESSAGE-BEGIN* '"-----Begin Forwarded Message-----")
(LOGIN-SETQ *FORWARDED-MESSAGE-SEPARATOR* '"--------------------")
(LOGIN-SETQ *FORWARDED-MESSAGE-END* '"------End Forwarded Message------")
(LOGIN-SETQ *ZMAIL-STARTUP-FILE-NAME* '"S:>ellen>ellen.babyl")
(LOGIN-SETQ *DEFAULT-MOVE-MAIL-FILE-NAME* '"S:>ellen>ellen.xmail")
(LOGIN-SETQ *DEFAULT-DRAFT-FILE-NAME* '"S:>ellen>doc27-stat-msg.temp")

Profile
Zmail Profile S:>Ellen>zmail-init.lisp (41)
Aborting, use the "Continue" command to continue.
Reading profile S:>ellen>zmail-init.lisp.41

Mouse-L: Edit.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Fri 18 Jul 7:58:11]  Ellen               ZWEI:        User Input                <Weather being initialized>
```

Figure 20. Profile mode display

8. Getting Help

The Genera environment contains many help facilities. This chapter summarizes the facilities for finding out information about the program you are writing and about the general state of Genera.

This chapter is a collection of the support tools and facilities available for finding the kind of information you need while programming. It is not exhaustive but suggestive. It does not recommend strategies for applying these facilities but rather lays out what is available for creating a personal style of using Genera effectively.

8.1 HELP Key

The key labelled HELP looks up context-dependent documentation.

HELP	Shows documentation available for the current activity. In some programs, c-HELP, m-HELP, and so on, provide additional documentation.
c-HELP	Shows a list of input editor commands (when typed at a Lisp Listener).
sy-HELP	Shows a list of the special function keys and the special character keys.
SELECT HELP	Shows programs and utilities that you can select using the SELECT key.
FUNCTION HELP	Shows a list of useful functions that you can invoke using the FUNCTION key.

See the section "HELP Key in Any Zmacs Editing Window", page 114.

8.2 Interaction with Completion and Typeout Windows

The Genera software has some general interaction conventions. For example, many editor commands offer name completion. You can apply these facilities to exploring the command space of the machine. This section describes some general facilities and strategies for making more effective use of the machine.

8.2.1 HELP Key in Any Zmacs Editing Window

The HELP key enables you to locate help material that is relevant to the current context. Individual programs are responsible for providing the routines that support the HELP key. The most complex general help facility is that provided by Zmacs editing windows. The HELP key provides access to a number of distinct kinds of help, depending on the key you press after the HELP key.

Command	*Description*
HELP ? or HELP HELP	Displays a brief summary of the Zmacs help options (similar to the rest of this chart).
HELP A	For looking up all Zmacs commands whose names contain a specified substring. You type the substring. Zmacs displays the one-line documentation for the command and tells you which key, if any, invokes it in the current context. See HELP V for looking up variable names. The "A" stands for "apropos". When people say, "Use Apropos," they are referring to this command.
HELP C	For looking up which command is bound to a particular key. You type the key; Zmacs displays the name of the command and its summary paragraph. HELP C uses Self Document.
HELP D	For looking up the summary paragraph for a Zmacs command. You enter the command name. Completion is available. It does not tell you how to invoke a command. Use HELP W for that. HELP D uses Describe Command.
HELP L	For finding out what you did that caused unexpected behavior. Zmacs displays a representation of the last 60 keys that you pressed. HELP L uses What Lossage.
HELP U	For undoing the last major operation. Zmacs preserves a copy of the buffer before doing certain operations, in particular, sorting and filling. You can revert to the state prior to one of those kinds of operations by using HELP U. Zmacs queries you whether to go ahead with undoing; the only information you have about what is being undone is the name of the class of operation, for example, "fill" or "sort". HELP U uses Undo.
HELP V	For looking up all Zmacs user variables whose print names contain a specified substring. You type the substring. Zmacs displays the variable names and their current values.

	See HELP A for looking up command names. HELP V uses Variable Apropos.
HELP W	For finding the key assignment for a particular command. You type the command name; Zmacs displays the current key assignment. Completion is available. HELP W uses Where Is.
HELP SPACE	Repeats the last HELP command you used.

In this chapter, all Zmacs commands appear by name rather than by key binding. Command tables indicate whether the command has a standard key binding or whether it must be used as an extended command. For example, Edit Zmacs Command is an extended command and requires that you invoke it with m-X. Forward Word is bound to m-F; you invoke it by holding down the META key and pressing F.

Command	*Type of command*
Edit Zmacs Command (m-X)	An extended command
Forward Word (m-F)	A command with a standard key binding
Find File (c-X c-F)	A command with a standard key binding

Functions and their arguments appear as in the following example:

(apropos *string* package inferiors superiors)

Words in italics are the arguments to the function. The words reflect the meaning of the argument. Bold words are optional arguments; you can leave them out. The reference description for the function explains the meanings of the arguments and the default values for optional arguments.

8.2.2 Zmacs Completion

Zmacs minibuffer commands offer *completion*, a facility for reducing the number of keys you need to type to specify a name. As soon as you have typed enough characters for a name to be recognized as unique, you can ask for completion. Up until then, you can ask to see which names are possible completions of what you have typed. You can tell when completion is available; the notation "(Completion)" appears at the right end of the minibuffer label line.

8.2.2.1 Completion for Extended Commands (m-X Commands)

The following table summarizes the keys that control completion for entering extended commands.

Key	*Action in m-X commands*
SPACE	Completes the words up to the current word, as far as they are unique.
HELP or c-?	Shows the possible completions in the typeout area.
Mouse-R	Pops up a menu of the possible completions.
c-/	Runs Apropos for each of the partially typed words in the name.
COMPLETE	Displays the full command name, if possible.
RETURN, END	Confirms the command when possible, whether or not you have seen its full name.

Request completion by pressing either COMPLETE or RETURN. Using COMPLETE shows the completed name, requiring a further RETURN to confirm it; using RETURN gets you completion and confirmation in one step.

Any time you are typing a Zmacs extended command name, completion is available. Zmacs command name completion works on initial substrings of each word in the command. For example, "m-X e z" is enough to specify the extended command "Edit Zmacs Command".

Until Zmacs can recognize the name as unique, your request for completion just completes as far as possible and moves the input cursor to the first ambiguous place in the command name.

Whenever you are entering a name in a minibuffer that offers completion, you can find out all possible completions of what you have typed so far. Two styles are possible. Pressing HELP or c-? shows the list of completions in the typeout area; the names are mouse sensitive. Clicking Mouse-R shows the list in a pop-up menu. One good strategy for browsing is to look at the list of completions for initial substrings that are common command verbs, like "show" or "set".

8.2.2.2 Completion for m-.

The m-. (Edit Definition) command offers completion over the set of names that is in the files that have already been loaded into editor buffers. In this case, you request completion with COMPLETE and then confirm it with RETURN.

m-. offers initial substring name completion, with hyphens rather than spaces delimiting the words. For example, "e-d-i" would be sufficient for specifying **zwei:edit-definition-internal** (assuming that Zmacs had previously parsed the source file containing it into a buffer).

8.2.3 Completion in Other Contexts

Completion is available in several other contexts, for example, buffer names and package names. Be on the lookout for the presence of "(Completion)" in the minibuffer label line. The conventions for extended commands usually apply.

8.2.4 Typeout Windows in Zmacs

Most of the Zmacs commands for looking up information display the information in a *typeout window*. A typeout window overlays the current buffer display with its contents and disappears as soon as you type any character. Most typeout windows contain mouse-sensitive items. In particular, Zmacs commands and Lisp function specs are mouse sensitive and small menus of operations on the names are available (Arglist, Edit Definition, and so on). See the mouse documentation line.

8.2.5 FEP Command Completion

While the keyboard is connected to the FEP, the following forms of completion are available:

- Pressing the HELP key at the FEP prompt (Fep>) or after typing part of the first word of a command shows the commands understood by the FEP command processor.

- Pressing the HELP key after typing the first word of a command shows a list of commands that begin with that word. Example: set SPACE HELP gives a list of commands that begin with the word set.

8.2.5.1 See...

- For more information about help facilities in editing: See the section "Getting Out of Trouble", page 69.

- For more information about help facilities in the mail program: See the section "Using Zmail", page 77.

- For more information about the FEP and a listing of available commands: See the section "The Front-End Processor" in *Site Operations*.

8.3 Summary of Help Functions in Different Contexts

Both Zmacs and Lisp offer facilities for finding information either about themselves or about the current environment. In addition, Zmacs offers ways to find information about Lisp functions and variables.

This section lists the names of the functions and commands that are available, grouped according to the context in which they are available. The purpose of this section is to summarize the capabilities and to help you determine both the overall contexts for which you can find help and a particular function that might be what you are looking for. Explanations for each of these functions appear in an alphabetical listing. See the section "Reference Description of Help Functions", page 120.

8.3.1 Zmacs Commands for Finding Out About the State of Buffers

Edit Buffers (m-X)
Edit Changed Definitions (m-X)
Edit Changed Definitions Of Buffer (m-X)
List Buffers (c-X c-B)
List Changed Definitions (m-X)
List Changed Definitions Of Buffer (m-X)
List Definitions (m-X)
List Matching Lines (m-X)
Print Modifications (m-X)
Select System as Tag Table (m-X)
Tags Search (m-X)

8.3.2 Zmacs Commands for Finding Out About the State of Zmacs

Apropos (m-X), HELP A
Describe Variable (m-X)
Edit Zmacs Command (m-X)
List Commands (m-X)
List Registers (m-X)
List Some Word Abbrevs (m-X)
List Tag Tables (m-X)
List Variables (m-X)
List Word Abbrevs (m-X)

8.3.3 Zmacs Commands for Finding Out About Lisp

Describe Variable At Point (c-sh-V)
Edit Callers (m-X)
Edit CP Command m-X
Edit Definition (m-.)
Edit File Warnings (m-X)
Function Apropos (m-X)
List Callers (m-X)
List Matching Symbols (m-X)
Long Documentation (c-sh-D)
Multiple Edit Callers (m-X)
Multiple List Callers (m-X)

Quick Arglist (c-sh-A)
Show Documentation (m-sh-D)
Show Documentation Function (m-sh-A)
Show Documentation Variable (m-sh-V)
Where Is Symbol (m-X)

8.3.4 Zmacs Commands for Finding Out About Flavors

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

8.3.5 Zmacs Commands for Interacting with Lisp

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

8.3.6 Lisp Facilities for Finding Out About Lisp

(apropos *string* **package inferiors superiors)**
(arglist *function* **flag)**
(describe *object***)**
(describe-area *area-name***)**
(describe-defstruct *instance* **structure-name)**
(describe-flavor *flavor-name***)**
(describe-package *package-name***)**
(describe-system *system-name***)**
(disassemble *function***)**
(documentation *function***)**
(si:flavor-allowed-init-keywords *flavor-name***)**
(inspect *object***)**
(compiler:load-compiler-warnings *file* **flush-flag)**
(mexp)
(trace specs)
(untrace specs)
(variable-boundp *variable***)**
(what-files-call *string* **package)**
(where-is *symbol* **package)**
(who-calls *symbol* **package inferiors superiors)**

8.4 Reference Description of Help Functions

This section contains a summary paragraph of documentation for each of the information-finding commands and functions. See the section "Summary of Help Functions in Different Contexts", page 117.

This reference list is in alphabetical order by name of the command or function. Zmacs editor commands appear according to the names of the commands that implement them, rather than according to the names of the keys that invoke them. For example, Compile Buffer (`m-X`) appears under "C" rather than under "M"; `c-sh-A` appears under "Q" (because its name is Quick Arglist) rather than under "C". For commands that are usually invoked by a single key rather than by `m-X`, the key name appears with the command. (Remember that you can always use `HELP W` to find the key that invokes a particular command.)

Some Zmacs commands come in pairs, such as List Callers and Edit Callers. The commands are very similar. The List version allows you to just look at the list or to decide to start editing the items in the list. The list items are always mouse sensitive. For the Edit version of the command, `c-.` is always the command for moving to the next item.

Apropos (m-X), HELP A
Displays all the Zmacs commands whose names contain a specified substring. You type the substring. Zmacs displays one line of documentation for the command and which key invokes it in the current context, if any.

(apropos *string* package inferiors superiors)
Displays all the symbols whose print names contain the string. By default, it looks in the **zl-user:global** package and its descendants, but you can specify a package name. For symbols that have function bindings, it displays the argument list. For symbols that are bound, it displays the notation "Bound". **zl:apropos** returns the symbols that it found as a list.

```
(apropos "forward" 'zwei)
```

(arglist *function* flag) (see also Quick Arglist)
Returns a representation of the arguments that the function expects. When the original function definition contained an **arglist** declaration, **arglist** returns that list when *flag* is not specified or **nil**. When *flag* is not **nil**, then **arglist** returns the real argument list from the function. When the original function used a **values** declaration, **arglist** returns the names for the values returned by the function.

```
(arglist 'make-array)
```

You cannot use **arglist** to find the arguments for combined methods.

Break (SUSPEND)	Enters a Lisp Listener from the current window. It uses the screen area of the frame that was selected when you used SUSPEND. When you use it from the editor, any Lisp forms you type are evaluated in the current package (the one showing in the status line). Use RESUME to return to the original context.
c-m-sh-E	See Evaluate Region Verbose.
c-sh-A	See Quick Arglist.
c-sh-C	See Compile Region.
c-sh-D	See Long Documentation.
c-sh-E	See Evaluate Region.
c-sh-V	See Describe Variable At Point.

Compile And Exit (m-Z)
Compiles the buffer and returns from top level. It selects the

window from which the last **(ed)** function or the last debugger `c-E` command was executed.

Compile Buffer (`m-X`)
: Compiles the entire buffer. With a numeric argument, it compiles from point to the end of the buffer. (This is useful for resuming compilation after a prior Compile Buffer has failed.)

Compile Changed Definitions (`m-X`)
: Compiles any definitions that have changed in any Lisp mode buffers. With a numeric argument, it queries individually about whether to compile each changed definition.

Compile Changed Definitions Of Buffer (`m-sh-C`, `m-X`)
: Compiles any definitions in the current buffer that have been changed. With a numeric argument, it prompts individually about whether to compile each changed definition.

Compile File (`m-X`)
: Compiles a file, offering to save it first. It prompts for a file name in the minibuffer, using the file associated with the current buffer as the default. It offers to save the file if the buffer has been modified.

Compile Region (`c-sh-C`, `m-X`)
: Compiles the region, or if no region is defined, the current definition.

Compiler Warnings (`m-X`) (see also Edit Compiler Warnings)
: Puts all pending compiler warnings in a buffer and selects that buffer. It loads the compiler warnings database into a buffer called *Compiler-Warnings-1*, creating that buffer if it does not exist.

(describe ***object*****)** (see also **inspect**)
: Displays available information about an object, in a format that depends on the type of the object. For example, describing a symbol displays its value, definition, and properties. **describe** returns the object.

```
(describe 'time:get-time)
```

(describe-area ***area-name*****)**
: Displays attributes of the specified area.

```
(describe-area (%area-number 'foo))
(describe-area 'working-storage-area)
```

(describe-defstruct *instance* **structure-name)**

Displays a description of the instance, showing the contents of each of its slots. *structure-name* is not necessary for named structures but must be provided for unnamed structures. When you supply **zl-user:structure-name**, you force the function to use that structure name instead of letting the system figure it out; in addition, it overrides the **:describe** option for structures that know how to describe themselves.

Describe Flavor (m-X) (see also **zl-user:describe-flavor)**

Displays a description of a flavor. It reads a flavor name via the mouse or from the minibuffer using completion. It displays a description of the flavor in a typeout window. The description includes names of flavors that the specified one directly depends on and names of flavors that depend on it. It also displays the documentation and the names of its instance variables.

(describe-flavor *flavor-name***)** (see also Describe Flavor)

Displays descriptive information about a flavor.

```
(describe-flavor 'tv:basic-menu)
```

(describe-package *package-name***)**

Displays information about a package.

```
(describe-package 'zwei)
```

That example is the same as this one:

```
(describe (pkg-find-package 'zwei))
```

(describe-system *system-name***)**

Displays information about a system, including the name of the file containing the system declaration and when the files in the current version of the system were compiled.

Describe Variable (m-X)

Displays the documentation and current value for a Zmacs variable. It reads the variable name from the minibuffer, using completion.

Describe Variable At Point (c-sh-V)

Displays information, in the echo area, about the current Lisp variable. The information includes whether the variable is declared special, whether it has a value, what file defines it, and whether it has documentation put on by **defvar** or **zl:defconst**. When nothing is available, it checks for lookalike symbols in other packages.

(disassemble *function***)** (see also **mexp**, Macro Expand Expression)

Displays the macro-instructions for the function. It does not work for functions that are not compiled or that are implemented in microcode, like **cons** or **car**.

```
(disassemble 'plus)
```

Use this function for things like finding clues about whether a macro is being expanded correctly.

Edit Buffers (m-X) (see also List Buffers)

Displays a list of all buffers, allowing you to save or delete buffers and to select a new buffer. A set of single character subcommands lets you specify various operations for the buffers. For example, you can mark buffers to be deleted, saved, or not modified. Use HELP to see further explanation. The buffer is read-only; you can move around in it by searching and with commands like c-N or c-P.

Edit Callers (m-X) (see also List Callers, Multiple Edit Callers)

Prepares for editing all functions that call the specified one. It reads a function name via the mouse or from the minibuffer with completion. By default, it searches the current package. You can control the package being searched by giving the function an argument. With c-U, it searches all packages; with c-U c-U, it prompts for a package name. It selects the first caller; use c-. (Next Possibility) to move to a subsequent definition. It takes about 5 minutes to search all packages.

Edit Changed Definitions (m-X) (see also List Changed Definitions)

Determines which definitions in any Lisp mode buffer have changed and selects the first one. It makes an internal list of all the definitions that have changed in the current session and selects the first one on the list. Use c-. (Next Possibility) to move to a subsequent definition. Use a numeric argument to control the starting point for determining what has changed:

1 For each buffer, since the file was last read (the default).
2 For each buffer, since it was last saved.
3 For each definition in each buffer, since the definition was last compiled.

Edit Changed Definitions Of Buffer (m-X) (see also List Changed Definitions Of Buffer)

Determines which definitions in the buffer have changed and selects the first one. It makes an internal list of all the definitions that have changed since the buffer was read in and selects the first one on the list. Use c-. (Next Possibility) to move to subsequent definitions. Use a numeric argument to control the starting point for determining what has changed:

1 Since the file was last read (the default).
2 Since the buffer was last saved.
3 Since the definition was last compiled, for each definition in the buffer.

Edit Combined Methods (m-X) (see also List Combined Methods)

Prepares to edit the methods for a specified message to a specified flavor. It prompts first for a message name, then for a flavor name. It selects the first combined method component. Use c-. (Next Possibility) to move to a subsequent definition. The definitions appear in the order that they would be called when the message was sent. Error messages appear when the flavor does not handle the message and when the flavor requested is not a composed, instantiated flavor.

Edit Compiler Warnings (m-X) (see also Compiler Warnings)

Prepares to edit all functions whose compilation caused a warning message. It queries, for each of the files mentioned in the database, whether you want to edit the warnings for the functions in that file. It splits the screen, putting the warning message in the top window. The bottom window displays the source code whose compilation caused the message. Use c-. (Next Possibility) to move to a subsequent warning and source function. After the last warning, it returns the screen to its previous configuration.

Edit Definition (m-.)

Prepares to edit the definition of a function, variable, flavor, or anything else defined with a "defsomething" special form. It prompts for a definition name from the minibuffer. Name completion is available for definitions in files that have already been loaded into buffers. You can select a name by clicking the mouse over a definition name in the current buffer. It selects the buffer containing the definition for that name, first reading in the file if necessary. With a numeric argument, it selects the next definition that satisfies the most recent name given. It tells you in the echo area when it finds more than one definition for a name.

Edit File Warnings (m-X)

Prepares to edit any functions in a specified file for which warnings exist. It prompts for a file name, which can be either a source file or a compiled file. It splits the screen, putting a warning message from the warnings database in the top window. The bottom window displays the source code whose compilation caused the message. If the database does not contain any warnings for this file, it prompts for the name of a file containing the warnings. Use c-. (Next Possibility) to move to a subsequent warning and source function. After the last warning, it returns the screen to its previous configuration.

Edit Methods (m-X) (see also List Methods)
: Prepares to edit all the methods on any flavor for a particular message. It prompts for a message name. It finds all the flavors with handlers for the message, makes an internal list of the method names, and selects the definition for the first one. Use c-. (Next Possibility) to move to subsequent definitions.

Edit Zmacs Command (m-X)
: Finds the source for the function installed on a key. You can press any key combination or enter an extended command name. Use a numeric argument to edit the function that implements a prefix command (like m-X or c-X).

Evaluate And Exit (c-m-Z)
: Evaluates the buffer and returns from top level. It selects the window from which the last **ed** function or the last debugger c-E command was executed.

Evaluate And Replace Into Buffer (m-X)
: Evaluates the Lisp object following point in the buffer and replaces it with its result.

Evaluate Buffer (m-X)
: Evaluates the entire buffer. With a numeric argument, it evaluates from point to the end of the buffer.

Evaluate Changed Definitions (m-X)
: Evaluates any definitions that have changed in any buffers. With a numeric argument, it prompts individually about whether to evaluate particular changed definitions.

Evaluate Changed Definitions Of Buffer (m-sh-E, m-X)
: Evaluates any definitions in the current buffer that have been changed. With a numeric argument, it prompts individually about whether to evaluate particular changed definitions.

Evaluate Into Buffer (m-X)
: Evaluates a form read from the minibuffer and inserts the result into the buffer. You enter a Lisp form in the minibuffer,

which is evaluated when you press END. The result of evaluating the form appears in the buffer before point. With a numeric argument, it also inserts any typeout that occurs during the evaluation into the buffer.

Evaluate Minibuffer (m-ESCAPE)

Evaluates forms from the minibuffer. You enter Lisp forms in the minibuffer, which are evaluated when you press END. The value of the form itself appears in the echo area. If the form displays any output, that appears as a typeout window.

Evaluate Region (c-sh-E, m-X)

Evaluates the region. When no region has been defined, it evaluates the current definition. It shows the results in the echo area.

Evaluate Region Hack (m-X)

Evaluates the region, ensuring that any variables appearing in a **defvar** have their values set. When no region has been defined, it evaluates the current definition. It shows the results in the echo area.

Evaluate Region Verbose (c-m-sh-E)

Evaluates the region. When no region has been defined, it evaluates the current definition. It shows the results in a typeout window.

(flavor-allowed-init-keywords *flavor-name***)** (In **si:**)

Returns a list containing the init keywords and initable instance variables allowed for a particular flavor.

```
(si:flavor-allowed-init-keywords 'tv:basic-menu)
```

Function Apropos (m-X)

Displays all the Lisp functions whose print names contain a particular substring. It reads the substring from the minibuffer. By default, it searches the current package. You can control the package being searched by giving the function an argument. With c-U, it searches all packages; with c-U c-U, it prompts for a package name.

(inspect *object***)** (see also **describe**)

Creates or selects an Inspector window and displays available information about an object. **inspect** and **describe** provide similar information, except that **inspect** is an interactive facility for further exploring a data structure.

```
(inspect tv:selected-window)
(inspect (tv:window-under-mouse))
```

List Buffers (c-X c-B) (see also Edit Buffers)

Prints a list of all the buffers and their associated files. The lines in the list are mouse sensitive, offering a menu of operations on the buffers. Clicking left on a line selects the buffer. For buffers with associated files, the version number of the file appears. For buffers without associated files, the size of the buffer in lines appears. For Dired buffers, the pathname used for creating the buffer appears as the version. The list of buffers appears sorted in order of last access, with the currently selected one at the top. You can inhibit sorting by setting **zwei:*sort-zmacs-buffer-list*** to **nil**.

List Callers (m-X) (see also Edit Callers, Multiple List Callers)

Lists all functions that call the specified function. It reads a function name via the mouse or from the minibuffer with completion. By default, it searches the current package. You can control the package being searched by giving the function an argument. With c-U, it searches all packages; with c-U c-U, it prompts for a package name. The names are mouse sensitive. Use c-. (Next Possibility) to start editing the definitions in the list. It takes about 5 minutes to search all packages.

List Changed Definitions (m-X) (see also Edit Changed Definitions)

Displays a list of any definitions that have been edited in any buffer. Use c-. (Next Possibility) to start editing the definitions in the list. Use a numeric argument to control the starting point for determining what has changed:

1 For each buffer, since the file was last read (the default).
2 For each buffer, since it was last saved.
3 For each definition in each buffer, since the definition was last compiled.

List Changed Definitions Of Buffer (m-X) (see also Edit Changed Definitions Of Buffer)

Displays the names of definitions in the buffer that have changed. It makes an internal list of the definitions changed since the buffer was read in and offers to let you edit them. Use c-. (Next Possibility) to move to subsequent definitions. Use a numeric argument to control the starting point for determining what has changed:

1 Since the file was last read (the default).
2 Since the buffer was last saved.

3 Since the definition was last compiled, for each definition in the buffer.

List Combined Methods (m-X) (see also Edit Combined Methods)
: Lists the methods for a specified message to a specified flavor. It prompts first for a message name, then for a flavor name. It lists the names in a typeout window. Error messages appear when the flavor does not handle the message and when the flavor requested is not a composed, instantiated flavor. Use `c-.` (Next Possibility) to start editing the definitions in the list.

List Commands (m-X)
: Lists names and one-line summaries for all extended commands available in the current context. It displays the names in a typeout window. The list is not sorted.

List Definitions (m-X)
: Displays the definitions from a specified buffer. It reads the buffer name from the minibuffer, using the current buffer as the default. It displays the list as a typeout window. The individual definition names are mouse sensitive.

List Matching Lines (m-X)
: Displays all the lines following point in the current buffer that contain a given string. It prompts for the string in the minibuffer. With a numeric argument, it shows only the first *n* occurrences of the string following point. The lines are mouse sensitive.

List Matching Symbols (m-X)
: Lists the symbols that satisfy a predicate. It prompts for a predicate lambda expression of one argument. The predicate gets compiled, for speed. The predicate must return something other than **nil** for the symbol to be included in the list. For example

you pressed:	`m-X L M S`
minibuffer contains:	`'(LAMBDA (SYMBOL))`
revised minibuffer:	`'(LAMBDA (SYMBOL) (string-search "foo" symbol))`

By default, it searches the current package. You can control the package being searched by giving the function an argument. With `c-U`, it searches all packages; with `c-U c-U`, it prompts for a package name. It selects the first one; use `c-.` (Next Possibility) to move to a subsequent definition.

List Methods (m-X) (see also Edit Methods)
Lists all the function specs for all methods on any flavor that handle a particular message. It prompts for the message name. It finds all the flavors with methods for the message and displays the information in a typeout window. The function specs are mouse sensitive.

List Registers (m-X)
Displays names and contents of all defined registers. Use Apropos to see commands for manipulating registers.

List Some Word Abbrevs (m-X)
Lists the abbreviations or expansions that contain the given string. Use Apropos to see the other abbreviation commands.

List Tag Tables (m-X)
Lists the names of all the tag tables currently available. Use Apropos to see other commands using tags.

List Variables (m-X)
Lists all Zmacs variable names and their values. With a numeric argument, it also displays the documentation line for the variable. Zmacs variables are those that have been provided for customizing the editor. Their names differ from their internal Lisp names:

Zmacs variable name: Fill Column

Internal Lisp name: **zwei:*fill-column***

List Word Abbrevs (m-X)
Lists all current abbreviations and their expansions.

(load-compiler-warnings *file* **flush-flag)** (In **compiler:**) (see also Load Compiler Warnings)
Loads a file containing compiler warning messages into the warnings database. It expects to load a file containing the printed representation of compiler warnings (as saved by **print-compiler-warnings**). It uses *flush-flag* to determine whether to replace any of the warnings already in the database. When the flag is not **nil**, it deletes any warnings associated with a source file before loading any new warnings for that file. Otherwise, it merges warnings from the file with those already in the warnings database. The default is **t**.

Load Compiler Warnings (m-X) (see also **compiler:load-compiler-warnings**)
Loads a file containing compiler warning messages into the warnings database. It prompts for the name of a file that

contains the printed representation of compiler warnings. It always replaces any warnings already in the database.

Long Documentation (c-sh-D) (see also Show Documentation)
: Displays the summary documentation for the specified Lisp function. It prompts for a function name, which you can either type in or select with the mouse. The default is the current function.

m-.	See Edit Definition.
m-ESCAPE	See Evaluate MiniBuffer.
m-sh-A	See Show Documentation Function.
m-sh-C	See Compile Changed Definitions Of Buffer.
m-sh-D	See Show Documentation.
m-sh-E	See Evaluate Changed Definitions Of Buffer.
m-sh-F	See Show Documentation Flavor.
m-sh-V	See Show Documentation Variable.

Macro Expand Expression (c-sh-M, m-X)
: Displays the macro expansion of the next Lisp expression in the buffer. It reads the Lisp expression following point and expands all macros within it at all levels, displaying the result on the typeout window. With a numeric argument, it pretty-prints the result back into the buffer, immediately following the expression.

(mexp) (see also **disassemble**)
: Displays the expansion of a macro. It prompts for a macro invocation to expand and then displays its expansion without evaluating it. It continues prompting until you enter something that is not a cons (for example, () stops it.)

Multiple Edit Callers (m-X) (see also Edit Callers)
: Prepares for editing all functions that call the specified ones. It reads a function name from the minibuffer, with completion. It then keeps asking for another function name until you end it with just RETURN. By default, it searches the current package. You can control the package being searched by giving the function an argument. With c-U, it searches all packages; with c-U c-U, it prompts for a package name. It selects the first caller; use c-. (Next Possibility) to move to a subsequent definition.

Multiple List Callers (m-X) (see also List Callers)
Lists all the functions that call the specified functions. It reads a function name from the minibuffer, with completion. It continues prompting for a function name until you end it with just RETURN. By default, it searches the current package. You can control the package being searched by giving the function an argument. With c-U, it searches all packages; with c-U c-U, it prompts for a package name. Use c-. (Next Possibility) to start editing the definitions in the list.

Print Modifications (m-X)
Displays the lines in the current buffer that have changed since the file was first read into a buffer. With a numeric argument, it displays the lines that have changed since the last save. To provide context, it shows the first line of each section that contains a change, whether or not that line has changed. The lines are mouse sensitive, allowing you to move to the location of a change.

Quick Arglist (c-sh-A) (see also **arglist**)
Displays the argument list for the current function. With a numeric argument, it reads the function name via the mouse or from the minibuffer. When the original function uses a **values** declaration, Quick Arglist returns the names for the values returned by the function.

Quit (c-Z)
Returns from top level. It selects the window from which the last **(ed)** function or the last debugger c-E command was executed.

Select Some Buffers as Tag Table (m-X)
Creates a tag table by selecting some buffers currently read in, querying about each one. With a numeric argument, it asks only about buffers whose name contains a given string.

Select System as Tag Table (m-X)
Creates a tag table for all the files in a system. It uses the file names as they appear in the **defsystem** function for that system.

Show Documentation (m-X, m-sh-D)
Looks up a topic from the documentation database and displays it on a typeout window. It offers the current definition as a default, but prompts for a definition, which can be supplied by mouse or minibuffer. It accepts only those topics for which documentation has been installed.

Show Documentation Flavor (m-sh-F)
Displays the documentation for the current flavor. With a numeric argument, it prompts for a device. The devices currently supported are the screen and an LGP printer.

Show Documentation Function (m-sh-A)
Displays the documentation for the current function. With a numeric argument, it prompts for a device. The devices currently supported are the screen and an LGP printer.

Show Documentation Variable (m-sh-V)
Displays the documentation for the current variable. With a numeric argument, it prompts for a device. The devices currently supported are the screen and an LGP printer.

Tags Search (m-X) Searches all files in a tags table for a specified string. It reads the string from the minibuffer and then prompts for a tags table name.

Trace (m-X) (see also **untrace**)
Toggles tracing for a function. With a numeric argument, it simply enables tracing for some function, without prompting you for trace options. It uses the same interface for specifying options as the Trace program in the System menu. See the section "Tracing Function Execution" in *Program Development Utilities.*

(trace specs) (see also **untrace**)
Turns on tracing for a function. With no arguments, it returns a list of all things currently being traced. With no additional options, tracing displays the name and arguments for a function each time it is called and its name and value(s) each time it returns. Complex options are available for entering breakpoints or executing code conditionally during tracing. See the section "Tracing Function Execution" in *Program Development Utilities.* See the section "Trace" in *Text Editing and Processing.*

```
(trace foo bar)
(trace #'(:method command-found :push))
```

Tracing very common functions (like **zl:format**) or functions used by **trace** itself or by the scheduler (like **time:get-time**) can crash the machine.

(untrace specs) Turns off tracing for a function that is being traced. With no argument, it turns off tracing for all functions currently being traced.

(variable-boundp *variable***)**
Returns **nil** or **t** indicating whether or not the variable is bound.

```
(variable-boundp tv:current-window)
```

(what-files-call *symbol* **package)**
Displays the names of files that contain uses of *symbol* as a function, variable, or constant. It searches all the function cells of all the symbols in *package*. By default, it searches the global package and its descendants. It returns a list of the pathnames of the files containing the callers.

Where Is Symbol (m-X)
Displays the names of packages that contain symbols with the specified name.

(where-is *string* **package)**
Displays the names of all packages that contain a symbol whose print name is *string*. It ignores the case of string. By default, it looks in the global package and its descendants. **where-is** returns a list of the symbols that it finds.

```
(where-is "foobar")
```

(who-calls *symbol* **package inferiors superiors)**
Displays a line of information about uses of the symbol as a function, variable, or constant. It searches all the function cells of all the symbols in *package*. By default, it searches the global package and its descendants. It returns a list of the names of the callers.

```
(who-calls 'time:get-time 'hacks)
```

8.5 Editing Your Input

When you make a mistake in typing or change your mind when typing a command or expression to the system, you have two choices:

- Press ABORT and begin again.
- Edit your input.

You do not need to invoke the input editor explicitly. The input editor is a feature of all interactive streams.

8.5.1 Input Editor Commands

Input Editor Commands: c-number, c-Minus and c-U provide numeric arguments.

REFRESH	Refresh Window	HELP	Display Documentation
PAGE	Erase Typeout	c-HELP	Display Commands
m-<	Beginning Of Buffer	m-HELP	Display Internal State
m->	End Of Buffer	ESCAPE	Display Input History
CLEAR INPUT	Clear Input	c-ESCAPE	Display Kill History
c-F	Forward Character	c-Y	Yank
c-B	Backward Character	m-Y	Yank Pop
c-D	Delete Character	c-m-Y	Yank Input
RUBOUT	Rubout Character	c-W	Kill Region
c-T	Exchange Characters	m-W	Save Region
c-A	Beginning Of Line	c-SPACE	Set Mark
c-E	End Of Line	c-<	Mark Beginning
c-P	Previous Line	c->	Mark End
c-N	Next Line	c-sh-Y	Yank Matching
c-K	Kill Line	m-sh-Y	Yank Pop Matching
m-F	Forward Word	c-m-sh-Y	Yank Input Matching
m-B	Backward Word	SCROLL	Scroll Vertical Forward
m-D	Delete Word	c-V	Scroll Vertical Forward
m-RUBOUT	Rubout Word	m-SCROLL	Scroll Vertical Backward
m-T	Exchange Words	m-V	Scroll Vertical Backward
m-U	Upcase Word	s-SCROLL	Scroll Horizontal Forward
m-L	Downcase Word	s-m-SCROLL	Scroll Horizontal Backward
m-C	Capitalize Word	c-m-S	Save Scroll Position
c-m-F	Forward Parentheses	c-m-R	Restore Scroll Position
c-m-B	Backward Parentheses	s-W	Kill Ring Push Region Strings
c-m-K	Delete Parentheses		
c-m-RUBOUT	Rubout Parentheses	s-S	Scroll Search Forward
LINE	New Line	s-R	Scroll Search Backward
BACK SPACE	Backward Character	c-sh-A	Describe Arguments
c-L	Refresh Window	c-sh-V	Describe Variable
c-O	Open Line	c-sh-D	Document Symbol
c-Q	Quote Character	c-m-J	Set Typein Style
m-sh-A	Lookup Function Documentation		
m-sh-V	Lookup Variable Documentation		
m-sh-F	Lookup Flavor Documentation		

8.5.2 Histories and Yanking

A *history* remembers commands and pieces of text, placing them in a history list. Additions to the history are placed at the top of the list, so that history elements are stored in reverse chronological order – the newer elements at the top of the history, the older elements toward the bottom.

Yanking commands pull in the elements of a history. *Top-level commands* start a yanking sequence. Other commands perform all subsequent yanks in the same sequence. A yanking sequence ends when you type new text, execute a form or command, or start another yanking sequence.

The system has different histories for different contexts. One of these is always the *current history*.

8.5.3 Types of Histories

Genera uses the following histories:

Type	*Description*
Input	History containing text typed at the input editor; a separate history exists for each window.
Kill	History of text deleted or saved in any window; a global history.
Replace	History of arguments to Query Replace (m-X) and related commands.
Buffer	History of editor buffers visited in this window.
Pathname	History of file names that have been typed.
Command	History of editor commands that use the minibuffer, and their arguments. Commands that do not use the minibuffer, such as m-RUBOUT, are not recorded in the history.
Definition	History of names of definitions that have been typed.

Except for the input histories, which are per-window, only a single instance of each of these histories exists, shared among all editors, including Zmacs, Zmail, and Dired.

8.5.4 Input Editor

In the input editor c-m-Y yanks from the history of previous inputs.

Because the input editor's kill history is the same as the Zwei kill history, c-SPACE, c-W, m-W, c-<, c->, c-Y, and related commands can be used in the input editor to move text back and forth between Zmacs, Zmail and the Input Editor . (Press c-HELP for a summary of commands.) Unlike Zwei, however, the input editor does not underline a marked region.

You can use most Zwei editing commands on yanked forms. Reactivating a yanked form is simple: just press END anywhere within or at the end of the form. ESCAPE displays the history of previous inputs. A numeric argument controls the length of the input history to be displayed. An argument of 0 displays the entire history.

c-ESCAPE displays the default kill history. A numeric argument controls the length of the kill history to be displayed. An argument of 0 displays the entire history.

8.5.5 The Displayed Default

When a command that reads an argument in the minibuffer displays a default, it puts the default onto the history temporarily. After reading and defaulting your input, it puts the argument onto the history instead. Thus c-m-Y always yanks the displayed default and c-m-2 c-m-Y yanks the last thing typed in that context. If no default is displayed, c-m-Y yanks the last thing typed in that context.

The displayed default is usually not the same as the most recent item in the history; often it is computed according to some heuristic based on past history and the exact command being given. It is pushed onto the top of the history in order to allow you to easily yank and edit it. This is useful when the heuristic comes close but does not provide exactly what you want.

8.5.6 Using Numeric Arguments for Yanking

A numeric argument of 0 to any yank command displays a list of the history and the numeric argument required to get each element of the history.

Example: The input history invoked in a Dynamic Lisp Listener by c-m-0 c-m-Y:

```
Lisp Listener 1 Input history:
  1: (+ 210 32)
  2: (* 17 6)
  3: Load Patches
  4: Show System Modifications
  5: Show Herald
  6: Login KJones
```

The history is displayed in reverse chronological order – the newest element first, for example, (+ 210 32); the oldest last, for example, Login KJones.

By default, a positive argument to c-Y and c-m-Y specifies how far from the newest element into *past* history is the element to be yanked. The numbers in the history display can be used as numeric arguments. (Optionally, you can set the variable **zwei:*history-rotate-if-numeric-arg*** so that arguments to the yanking commands are measured relative to the origin. See the section "Customizing the Input Editor", page 105.)

Example: c-m-1 c-m-Y yanks element #1, (+ 210 32), from the history.

Example: `c-m-2 c-m-Y` yanks element #2, `(* 17 6)`, from the history.

A positive or negative argument to `m-Y` is measured relative to the last element yanked, not the newest element.

Example: Pressing `c-m-2 c-m-Y` yanks `(* 17 6)`; then pressing `m-4 m-Y` yanks `Login KJones`, not element #4. Displaying the history at this point looks this:

```
Lisp Listener 1 Input history:
   1: (+ 210 32)
   2: (* 17 6)
   3: Load Patches
   4: Show System Modifications
   5: Show Herald
-> 6: Login KJones
```

Element #6, marked by a pointer, is the origin. (Note: The origin is not the most recent element because `m-Y` has changed the origin.)

A top-level command given without an argument retrieves the element at the origin, which is the last element yanked in the previous yanking sequence, not necessarily the newest element of the history.

Example: `c-m-Y` yanks `Login KJones)` from the history.

A numeric argument of `c-U` not followed by any digits is the same as no numeric argument with one exception: Point is placed before the text yanked and mark is placed after – the reverse of the ordinary placement.

To find out how to customize the input editor: See the section "Customizing the Input Editor", page 105.

8.6 System Conventions and Helpful Hints

8.6.1 Miscellaneous Conventions

All uses of the phrase "Lisp reader", unless further qualified, refer to the part of Lisp that reads characters from I/O streams (the **zl:read** function), and not the person reading this documentation.

By default, Symbolics-Lisp displays numbers in base 10. If you wish to change it: See the section "What the Reader Recognizes" in *Reference Guide to Streams, Files, and I/O.*

Several terms that are used widely in other references on Lisp are not used much in Symbolics documentation, as they have become largely obsolete and misleading. They are: "S-expression", which means the printed representation of a lisp object; "Dotted pair", which means a cons; and "Atom", which means, roughly, symbols and numbers and sometimes other things, but not conses. For definitions of the terms "list" and "tree": See the section "Manipulating List Structure" in *Symbolics Common Lisp.*

8.6.2 Answering Questions the System Asks

The system occasionally asks you to confirm some command. There are two forms this can take:

- Simple commands such as Load File or Save File Buffers might ask you to confirm with a question requiring a Y (for yes) or an N (for no).

  ```
  Save Buffer program.lisp >kjones>new-project> tuna: ? (Y or N)
  ```

  You press `Y` or `SPACE` for yes, `N` for no.

- Destructive commands, such as Initialize Mail, require that you type the entire word yes to confirm them.

  ```
  Do you really want to do this? (Yes or No)
  ```

  You must type the entire word yes to confirm the the command. Thus you are less likely to issue such a command accidentally.

Lisp provides several functions for this kind of querying: See the section "Querying the User".

8.6.3 Questions Users Commonly Ask

What is a Logical Pathname?

A logical pathname is a kind of pathname that doesn't correspond to any particular physical file server or host. Logical pathnames are used to make it easy to keep software on more than one file system. An important example is the software that constitutes the Lisp Machine system. Every site has a copy of the basic sources of the programs that are loaded into the initial Lisp environment. Some sites might store the sources on a UNIX file system, while others might store them on a TOPS-20. However, the software needs to find these files no matter where they are stored. This is accomplished by using a logical host called SYS. All pathnames for system software files are actually logical pathnames with host SYS. At each site, SYS is defined as a logical host, and there is a translation table that maps the SYS host to the actual physical machine for that site.

Here is how translation is done. For each logical host, there is a mapping that takes the name of a directory on the logical host, and produces a device and a directory for the corresponding physical host. For example, the logical host SYS has a directory SITE;. At a site that keeps its sources on a TOPS-20 this might map to SS:<SITE> . Then the file SYS:SITE;NAMESPACE.LISP translates to SS:<SITE>NAMESPACE.LISP. On a UNIX system this same file might translate to /usr/system/namespace.l. The important thing is that everyone can refer to the file by its logical pathname, SYS:SITE;NAMESPACE.LISP, where the name before the ":" is the logical host name, and logical directories are separated by ";"s. You can define the translation of a logical pathname to be any physical pathname of any operating system type, but to access a file with a logical pathname you need only to use logical pathname syntax.

The function **fs:set-logical-pathname-host** is used to define a logical host and its logical directories. Here are some sample uses:

```
(fs:set-logical-pathname-host "SYS" :physical-host "my-vms"
                    :translations '(("games;" "[games]")
                                    ("*;" "[symbolics.*]")))
```

This says that sys:games; translates to my-vms:[games], and that any other logical directory on the logical host SYS translates to a subdirectory under [symbolics] of the same name. See the function **fs:set-logical-pathname-host** in *Reference Guide to Streams, Files, and I/O.*

What is a World Load?

A world load can be thought of as a snapshot of an operating Lisp environment. All of the functions, variables, and other Lisp objects that were present in the Lisp environment when the snapshot was made are contained in the world load file on the disk. Typically, snapshots of worlds are made only when such a snapshot would save significant time later. For example, after you have initially configured your new machine at your site, it is useful to make a snapshot of the configured environment because it saves you time in the future (you don't have to configure the machine each time you boot it). If you usually load MACSYMA or FORTRAN each time you boot, it is advantageous to make a snapshot of a world with that software loaded, to save you the time of loading it. Remember, everything in the environment is contained in the snapshot, so you don't want to create a world load file after you've been using the editor or most system facilities (you don't want to find old text in your editor buffer when you cold boot.). The way to create a snapshot and save it to disk is by using the command Save World or the function **(zl:disk-save)**.

Why Do You Name Machines and Printers?

Naming inanimate objects such as hosts, printers, sites, and networks may seem foolish if you have only one of each, but if you have large numbers of machines, names are a convenient way to easily refer to a particular machine with a particular address without having to remember its network address, machine type, and physical location. One customer named its machines after the characters in Winnie the Pooh, while another named its machines after the wives of Henry VIII.

8.6.4 Questions About the FEP and LMFS

Why Can't I Write Out Files When I Have Free Disk Space?

The 3600 disk is physically divided into partitions known as FEP files. This division of the disk is called the FEP file system. However, when one speaks of the file system of a Lisp Machine, one is generally referring to the LMFS (Lisp Machine File System) of that machine. This is the file system you edit when you click left on [Tree Edit Root] in the FSEdit window, and is the file system used when you specify file names of the form *Lisp Machine Name:>directory>filename.type.version*. The entire Lisp Machine local file system normally resides inside one big file of the FEP file system (typically FEP0:>LMFS.FILE.1). Thus, LMFS is full when the amount of space allocated to it (in other words, FEP0:>LMFS.FILE.1) is full. Thus, LMFS could be full but there could still be 100,000 unused blocks on the disk (not even allocated as FEP files). See the section "Adding a Spare World Load as LMFS File Space" in *Internals, Processes, and Storage Management.*

How Do I Create a FEP File?

There aren't too many reasons for creating FEP files. If you want to create a file to allocate more LMFS file space, simply enter the File System Editing Operations window, by using SELECT F, by clicking on [File System] in the System Menu, or by using the Select Activity File System Operations command. Then click on [Local LMFS Operations]. The second level menu pops up. Click on [LMFS Maintenance Operations]. Click right on [Initialize]. A menu pops up. Click on [Auxiliary Partition] and click on the name above this so that you can specify a name for the auxiliary partition. Typically, a good name is FEP0:>LMFS-AUX.FILE. (Of course, if you have more than one drive, or a FEP file named LMFS-AUX.FILE already exists, you should choose another name.) Then click on [Do It]. It will ask you how much space to allocate to this file; specify a number of blocks.

When working with FEP files, the File System Editor is good only for creating FEP files to be allocated to LMFS. If you need a FEP file for another purpose (extra paging, for example) and create it with FSEdit, the LMFS data structure contained on your disk might become very confused, and can potentially destroy the file system of your machine. The Create FEP File command creates a FEP file for purposes other than a LMFS partition. See the section "Create FEP File Command", page 232. The following Lisp function also creates such a FEP file:

```
(WITH-OPEN-FILE (FILE FEPn:>Filename.type.version
                      :DIRECTION :BLOCK
                      :IF-EXISTS :ERROR)
   (SEND FILE :GROW 30000))
```

The italicized string above represents the name of the FEP file to be created, and the italicized 30000 represents the size you want to make the file.

For more information about LMFS and the FEP file system: See the section "FEP File System" in *Internals, Processes, and Storage Management.*

9. Recovering From Errors and Stuck States

9.1 Introduction

Sometimes it is hard to know whether or not your machine is in trouble, because some operations, particularly those involving other network machines, can take a long time. Periodically check the process state and the run bars on the status line. The run bars flicker when the machine is working. As long as the run bars are flickering and the process state is changing occasionally, the machine is probably working properly. Some process states mean trouble if they persist, say, for a minute or more.

Look at the clock in the status line. If the clock is ticking, processes are being scheduled. If the clock is not ticking, the 3600 is halted. As long as the FEP is working, it prints a message near the top of the screen when the 3600 has halted and then gives its `Fep>` prompt. When the 3600 resumes its previous state, it updates the clock with the correct time.

9.2 Recovery Procedures

If the status line displays one of the following process states, recover by using the appropriate procedure:

State	*Recovery procedure*
`Wait Forever`	Select a different window, then reselect the one you were in.
`Output Hold`	Press `FUNCTION ESCAPE` (the `ESCAPE` key is in the top row, second from the left); if that puts you in the Debugger, use `ABORT`.
`Arrest`	Press `FUNCTION - A` (that is, a three-key sequence).
`Lock`	Try `FUNCTION 0 S` to see if any windows want to type out. If that does not help, press `c-ABORT`.
`Selected`	Press `FUNCTION 0 S`.
`(no window)`	Use the mouse or `SELECT` key to select the window you want.

You can press `SUSPEND` to get to a Lisp read-eval-print loop. You can press `c-m-SUSPEND` to force the current process into the Debugger.

9.3 The Debugger: Recovering From Errors and Stuck States

Errors that are not caught and handled by the program that triggered them invoke the Debugger. See the section "Entering the Debugger" in *Program Development Utilities.*

9.4 Resetting the FEP

Resetting the FEP restarts the FEP system, thereby discarding knowledge of the FEP's free storage area. Resetting might be necessary if you unplug the console video cable from either end or turn the console off and on. You also need to reset the FEP if you receive the error message: `No More Memory`. [You can reset the FEP from either the keyboard or the processor front panel. Note that when the FEP is being reset the fault light (located on the front panel of the processor box) is turned on by the hardware. Then, when the FEP finishes initializing itself the FEP turns the fault light off.]

- To reset the FEP from the keyboard:

  1. Type the Halt Machine command at a Lisp command prompt to stop Lisp and give control of the keyboard to the FEP.

     If no Lisp Listener is responsive, press `h-c-FUNCTION` to stop Lisp.

  2. Type the command Reset FEP to the FEP prompt.

  3. Press `Y` to answer the confirmation prompt.

  4. Type the command Hello to the FEP prompt to initialize the overlay files.

- To reset the FEP from the processor front panel:

  1. Push the red RESET button on the processor front panel.

  2. Press the spring-loaded YES switch to answer the "Reset FEP?" question (This question is asked only if you have a 3600 machine model).

After you reset the FEP, the keyboard is connected to the FEP, not to Lisp. Type the Hello command to the FEP prompt, and then give the Start command and press `RETURN` to warm boot the machine and Lisp, and return control of the keyboard to Lisp.

9.5 Warm Booting

If an error occurs in the keyboard process, window system, or scheduler, making the machine unresponsive to the keyboard, you may have to warm boot the machine. Warm booting causes either a **:flush** or **:reset** message to be sent to all processes in the system, depending on the type of process.

To warm boot the computer, use the following procedure:

1. Type one of the following to a Lisp Listener:

   - Halt Machine

   - `(si:halt)`

   You are now connected to the FEP.

   If you cannot obtain a Lisp Listener window or if no Lisp Listener is responding to keyboard input, you should use h-~~c-*upper-left*, (*upper-left* is the key in the upper-left corner of the keyboard. It corresponds to LOCAL on old keyboards and~~ FUNCTION ~~on new keyboards.)~~

2. Type `start` at the FEP prompt (`Fep>`) and press RETURN.

Sometimes, the machine prints `lisp stopped itself` and returns control to the FEP. When this happens, at the FEP prompt (`Fep>`) you should type `show status`, check the information it provides, and then type `start`.

9.6 Halting

Halting the 3600 leaves all Lisp states intact. To halt the 3600 in order to connect to the FEP, type **sys:halt** to a Lisp Listener or use h-c-*upper-left*. You are now connected to the FEP. To return to Lisp, type `continue` at the FEP prompt (`Fep>`) and press RETURN.

```
on the screen:   Fep>
you type:        continue
you press:       RETURN
```

The 3600 can halt itself under exceptional conditions. In this case, try typing `continue`. If `continue` does not work, use `start`.

10. How to Get Output to a Printer

10.1 Introduction to the Hardcopy Facilities

The hardcopy System provides a uniform interface for sending output to a printer. It allows the user or the program to specify formatting information in a device independent way for output on a supported printer.

The first section of this document deals with the commands provided for the user to request and control hardcopy. The second section deals with the functions a program needs to request hardcopy.

In order for menu items, commands, and functions that refer to printing and hardcopy to work, your site must have a properly connected printing device.

See the *Printer Installation Guide* for your printer type.
See the section "Namespace System Printer Objects" in *Networks.*

10.2 Printing and Hardcopy Commands

10.2.1 Commands for Producing Hardcopy

You can produce hardcopy from the Command Processor, by using the System Menu, from the editor, from Zmail, from Dired in the editor, and from the file system editor. You can also get a hardcopy of your screen at any time.

10.2.1.1 Hardcopying From the Command Processor

The simplest way to produce hardcopy is with the Command Processor:

Hardcopy File *file-spec printer keywords*

Sends a file to a hardcopy device.

file-spec	The pathname of the file to be printed. The default is the usual file default.
printer	The printer to use to output the file. The default is determined from your init file or from the default-printer attribute for the host in the namespace database.
keywords	:Body Character Style :Copies :Delete :Ending Page :File Types :Heading Character Style :Orientation :Running Head :Starting Page

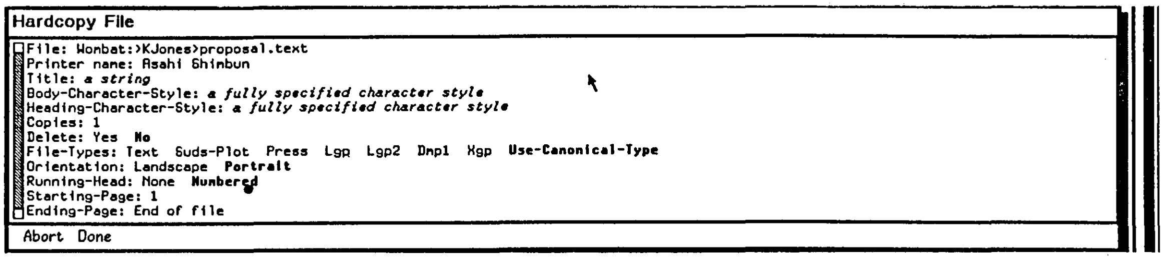
:Body Character Style	The character style to use for printing the text of the file and against which to merge any character styles in the file. The default is the *null style*, **(nil nil nil)**, meaning use the default for the printer.
:Copies	{*number*} The number of copies to print. The default is 1.
:Delete	{yes, no} Whether to delete the file after it is printed. The default is no, not to delete. Adding the :delete keyword to your hardcopy command string is the same as :delete yes.
:Ending Page	{*number*} The last physical page to print. The default is the last page of the file. A page is defined by the presence of a PAGE character or form feed in the file. Thus plain text files with no page markers in them are treated as a single page, although they take up several sheets of paper. Press format files, on the other hand, have form feeds or PAGE characters in them. It is important to remember that these are physical pages and do not necessarily correspond to the page numbering appearing in the heading. For example, the first physical page of a press file is probably a title page and the second physical page might be numbered *i* so the page numbered 1 might be the third physical page.
:File Types	{Text, Suds-Plot, Press, Lgp, Lgp2, Dmpl, Xgp or use-canonical-type} The internal format of the contents of the file, to interpret for printing. The default is use-canonical-type, meaning that the type is determined from the extension to the file name.
:Heading Character Style	The character style to use for the running head supplied by the :running head keyword.
:Orientation	{landscape, portrait} Orientation on the paper for the output. Portrait is left to right across the short dimension of the paper. Landscape is left to right across the long dimension. The default is portrait.
:Running Head	{none, numbered} Type of running head to print on the top of each page. The default is numbered.
:Starting Page	{*number*} The first physical page to print. The default is the first page of the file. A page is defined by the presence of a PAGE character or form feed in the file. Thus plain text files with no page markers in them are treated as a single page,

although they take up several sheets of paper. Press format files, on the other hand, have form feeds or PAGE characters in them. It is important to remember that these are physical pages and do not necessarily correspond to the page numbering appearing in the heading. For example, the first physical page of a press file is probably a cover page and the second physical page might be numbered *i* so the page numbered 1 might be the third physical page.

:Title	{*string*} Title to appear on the cover page to identify the output. The default is your user name.

10.2.1.2 Hardcopying From the System Menu

To produce hardcopy using the System Menu, click on [Hardcopy]. This pops up a Choose Variable Values menu that allows you to specify the pathname of the file to be hardcopied and to select the printer, character style and other parameters.



```
Hardcopy File
File: Wonbat:>KJones>proposal.text
Printer name: Asahi Shimbun
Title: a string
Body-Character-Style: a fully specified character style
Heading-Character-Style: a fully specified character style
Copies: 1
Delete: Yes  No
File-Types: Text  Suds-Plot  Press  Lgp  Lgp2  Dmp1  Xgp  Use-Canonical-Type
Orientation: Landscape  Portrait
Running-Head: None  Numbered
Starting-Page: 1
Ending-Page: End of file
Abort  Done
```

Figure 21. The Hardcopy Menu

10.2.1.3 Hardcopying From Zmacs

You can hardcopy a region, a buffer, or a file from Zmacs.

Hardcopy Region (m-X)

Sends a region's contents to the local hardcopy device for printing.

For full information on Genera hardcopying: See the section "How to Get Output to a Printer", page 147.

Hardcopy Buffer (m-X)

Prompts for the name of a buffer and then prints the specified buffer on the local hardcopy device.

For full information on Genera hardcopying: See the section "How to Get Output to a Printer", page 147.

Hardcopy File (m-X)

Prompts for the name of a file and then prints the specified file on the local hardcopy device.

For full information on Genera hardcopying: See the section "How to Get Output to a Printer", page 147.

Kill Or Save Buffers (m-X)

Puts up a multiple-choice menu listing all existing buffers. Choices are: Save, Kill, Unmodify, and Hardcopy. Specify these options next to the buffer names in the menu. This command appears on the editor menu.

10.2.1.4 Hardcopying From Zmail

You can hardcopy a single message or a collection of messages from Zmail.

Hardcopying

Hardcopy Message (m-X)	Hardcopies the current message.
Hardcopy All (m-X)	Hardcopies all the messages in the current sequence.

You can click right on [Other] in the Zmail menu and select Hardcopy to hardcopy the current message.

You can also click right on [Move] and select Hardcopy.

For any individual message you can click right on its summary line then click right on [Move] and select Hardcopy.

To copy all messages in current sequence click right on [Map Over] then right on [Move] and select Hardcopy.

In any of these commands you can click right on Hardcopy to get a menu that permits you to specify the number of copies and which printer to use. The `Other` option in the list of printers allows you to specify an arbitrary printer, using either its pretty name or its namespace name. This printer becomes the selected printer, and remains in the menu for subsequent hardcopy commands.

You can check the status of a printer from Zmail.

Show Printer Status (`m-X`)
: Prompts for the name of a printer and displays its print queue.

You can also hardcopy files from Zmail.

Hardcopy File (`m-X`)
: Prompts for a pathname and sends the specified file to the printing device specified in Hardcopy Options in your Zmail profile. The default is the first pathname specified in the `File-References:` header field.

Format File (`m-X`)
: Prompts for a pathname and displays the specified file formatted using the editor's formatting capability. `c-U m-X` Format File formats the file and sends it to a printer. The default pathname is the first pathname specified in the `File-References:` header field.

10.2.1.5 Hardcopying From Dired

You can mark files to be hardcopied in Dired. When you exit from Dired, the files marked to be hardcopied are sent to the printer.

`P` Dired Hardcopy File

Marks the current file for printing. Dired puts a P in the first column to show that the file has been so marked.

With a numeric argument *n*, marks the next *n* files for printing.

10.2.1.6 Hardcopying the Screen

You can get a hardcopy of what is displayed on your screen by pressing `FUNCTION Q`:

`Q`	Hardcopies the entire screen.
`c-Q`	Hardcopies the selected window.
`m-Q`	Hardcopies the entire screen, minus the status and mouse documentation lines.

10.2.1.7 Hardcopying From the File System Editor

You can use the system hardcopy menu from FSEdit. You click on `Hardcopy` in the menu of file operations invoked by clicking right on a file name.

10.2.2 Other Hardcopy Commands

10.2.2.1 Changing the Default Printer

When a site has more than one printer, one of the printers is specified as the site default printer. Show Printer Defaults tells you what the current printer is:
Show Printer Defaults

Displays the current default printer(s). If you send all your hardcopy output to one printer, this is displayed as

```
Default Printer: printer-name
```

If you use a different printer for text and screen hardcopy, this is displayed as

```
Default Text Printer: printer-name1
Default Bitmap Printer: printer-name2
```

You can change the default printer with the Set Printer command:

Set Printer *printer-name keywords*

Sets the default printer for hardcopy.

printer-name	The name of a supported printer that can be reached by your machine.
keywords	:Output Type
:Output Type	{text bitmap both} The type of output to send to that printer. Text means files and mail messages, bitmap means graphics and screen hardcopy. The default is both, meaning use the same printer for both types of output.

You can change the default in your init file to specify the printer that is most convenient for you. See the function **hardcopy:set-default-text-printer**, page 108.

The Hardcopy File command accepts a keyword argument of :printer, allowing you to specify a printer when you give the command. For example:

```
Hardcopy File q:>kjones>report.pr :printer beacon
```

In the System Menu, using [Hardcopy] allows you to specify a different printer name; the printer name is mouse-sensitive.

10.2.2.2 Managing the Print Spooler Queue

You can find out the status of a printer and its spooler queue with the Show Printer Status command:

Show Printer Status Command

Show Printer Status *printer*

Displays the print queue for the specified printer or printers.

printer	The name of a printer or printers (separated by commas) whose print queue to show, or All to show all the queues for all printers at your site. The default is your current printer default. If your text printer and your bitmap printer are different, your text printer is used as the default for Show Printer Status.

The display of requests is mouse sensitive and can be clicked on to select arguments for the Delete Printer Request and Restart Printer Request commands. This is only true for print spoolers running Release 7.0.

This command is also available in Zmail as `m-X` Show Printer Status.

Print requests can be canceled by using the Delete Printer Request command.

Delete Printer Request Command

Delete Printer Request *printer-request*

Deletes the specified print request from the print queue.

printer-request	A string specifying the printer and the request. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.

Print requests can be restarted by using the Restart Printer Request command.

Restart Printer Request Command

Restart Printer Request *printer-request*

Restarts a print request that has not yet finished. This is useful if something goes wrong with the printing, for example the paper is coming out crumpled.

printer-request	A string specifying the printer and the request. The print

request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.

The Printer can be halted, started and reset remotely.

Halt Printer Command

Halt Printer *printer printer-request keywords*

Halts the specified printer.

printer	The name of the printer to halt.
printer-request	(Optional) If the printer is printing a request when the Halt Printer command is given, it displays the request and asks you to confirm the halt command. If you supply a *printer-request* argument and it matches the request that is printing, the printer is halted immediately without requiring confirmation. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.
keywords	:Disposition, :Extent, :Reason, :Starting From, :Urgency
:Disposition	{Delete, Hold, Restart} What to do with the request that is printing. Delete deletes the request from the queue; you must request it again to have it printed. Hold retains the request in the queue but does not print it when the printer restarts. Restart restarts printing the interrupted request from the beginning when the printer restarts. The default is Delete.
:Reason	{*string*} The reason for the shutdown. This appears in the display from Show Printer Status to explain what is happening to users. The default is "Printer suspended by operator."

The following three keywords are related and interact to control precisely when the printer halts.

:Extent	{Entire, Copy} The extent of the request to be cancelled. Entire refers to the whole request. Copy refers to a single copy. In a request for one copy of a document, Entire and Copy are synonymous. The default is Entire.
:Starting From	{*number*} The copy number. If :Extent is Entire, this has no meaning. If :Extent is Copy, this is the number of the copy after which to halt the printer. The default is 0.

:Urgency	{Asap, After-Extent} When to halt. Asap means instantly, ignoring any settings of :Extent and :Starting From. After-Extent means halt based on the settings of the :Extent and :Starting From keywords. The default is Asap.

Start Printer Command

Start Printer *printer*

Starts the specified printer processing its print queue again after it has been halted with the Halt Printer command.

printer	The name of the printer to start.

Reset Printer Command

Reset Printer *printer printer-request keywords*

Resets a printer.

printer	The printer to reset.
printer-request	(Optional) If the printer is printing a request when the Reset Printer command is given, it displays the request and asks you to confirm the reset command. If you supply a *printer-request* argument and it matches the request that is printing, the printer is reset immediately without requiring confirmation. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.
keywords	:Disposition
:Disposition	{Delete, Hold, Restart} What to do with the request that is printing. Delete deletes the request from the queue; you must request it again to have it printed. Hold retains the request in the queue but does not print it when the printer restarts. Restart restarts printing the interrupted request from the beginning when the printer restarts. The default is Delete.

Resetting is like turning the printer off and then on again except that it is done remotely, you do not have to go over to the printer.

11. When and How to Use the Garbage Collector

11.1 Principles of Garbage Collection

It is fundamental to the nature of Lisp that programs and systems allocate memory dynamically and in large amounts. (The allocation of memory for a basic list element, or *cons*, or for any other purpose, is called *consing* for the purpose of this discussion and in most other writings on Lisp.) Even with the large amount of virtual memory on a Symbolics computer, it is possible for a program to use it all up. At this point the machine halts and must be rebooted. This event can always be delayed, almost indefinitely, if the underlying system can reclaim memory that is unused.

Objects that are no longer in use, with no references from other objects, are termed *garbage.* Garbage is distinguished from *good objects* or *good data* by the fact that it no longer serves any purpose in the current Lisp world. For example, if the car of a cons is changed from object A to object B, and there are no other references to A, then A is garbage. Objects in the Genera environment can be said to have a *lifetime,* which means how long the object remains "good" or valid. Three lifespans are distinguishable:

Static	Object will probably never become garbage. Example: standard system functions.
Dynamic	Object will probably become garbage eventually. Example: lines in editor buffers.
Ephemeral	Object will probably become garbage very quickly. Example: intermediate structure generated by the compiler.

You can control the garbage collection status of your own areas with the **make-area** function.

Garbage collection (GC) involves these three steps:

- *Scavenging* virtual memory, that is, periodically sifting through areas of memory, separating good objects from the garbage

- *Transporting* good objects to a safe place

- Reclaiming the memory occupied by garbage

Several strategies for garbage collection exist. Some allow you to continue doing other work and some do a more complete job but require additional machine resources for some period of time.

Garbage collection need not be used at all. It should be used either when you are running a program that allocates large amounts of virtual memory (where the total allocated might exceed the amount of free memory in a cold-booted system) or when the total allocations of many programs might, over a relatively long period of time, exceed the capacity. In either case, garbage collection is a strategy aimed primarily at preserving the state of an operating Lisp world as long as possible and avoiding a cold boot.

Incremental Versus Immediate GC

There are two basic modes of garbage collection, each with some variations possible:

- *Incremental garbage collection* works in parallel with other processes in the system, allowing you to continue working while it is in progress. This mode is based on *incremental copying*, so called because objects are copied one at a time and there is relatively little effect on the user's interaction with the system. *Dynamic-object garbage collection* incrementally collects garbage in all *nonstatic* areas of memory. *Ephemeral-object garbage collection* incrementally collects garbage, concentrating on specific parts of memory that are known to contain short-lived objects. Both kinds of incremental operation ignore *static* areas of memory that change slowly and so are unlikely to contain garbage. For an explanation of static memory: See the section "Theory of Operation of the GC Facilities" in *Internals, Processes, and Storage Management.*

- *Nonincremental, or immediate, garbage collection* takes less free memory and less total processor time to work successfully than does the incremental mode. Nonincremental garbage collection is normally done with the Start GC :Immediately command or with the **gc-immediately** function, although those directives still ignore static areas. These directives allow no other work to be done by the process running it, although other processes are still scheduled. In most cases, though, immediate garbage collection places a heavy enough burden on the machine that other processes are not useful while it is operating. The immediate garbage collection invoked by the function **si:full-gc** deals with static areas.

Note: Areas of memory can be specified as being static with the function **make-area.**

The command Show GC Status allows you to check on how much free space you have and determine whether or not you should turn on the garbage collector.

```
Show GC Status
Status of the ephemeral garbage collector:                  On
First level of METERING:METERING-CONS-AREA: capacity 196K, 0K allocated, 0K
used.
Second level of METERING:METERING-CONS-AREA: capacity 98K, 0K allocated, 0K
used.

First level of DW::*EQL-DISPATCH-AREA*: capacity 98K, 256K allocated, 56K used.
Second level of DW::*EQL-DISPATCH-AREA*: capacity 49K, 0K allocated, 0K used.

First level of WORKING-STORAGE-AREA: capacity 196K, 448K allocated, 29K used.
Second level of WORKING-STORAGE-AREA: capacity 98K, 2048K allocated, 47K used.

Status of the dynamic garbage collector:                    On
Dynamic (new+copy) space 6,490,761.  Old space 0.  Static space 12,479,751.
Free space 26,574,848.  Committed guess 22,488,118, leaving 3,824,586 to use
before flipping.
There are 9,779,900 words available before Start GC :Immediately might run out
of space.
Doing Start GC :Immediately now would take roughly 33 minutes.
There are 26,574,848 words available if you elect not to garbage collect.

Garbage collector process state: Await ephemeral or dynamic full
Scavenging during cons: On, Scavenging when machine idle: On
The GC generation count is 328 (1 full GC, 2 dynamic GC's, and 325 ephemeral
GC's).
Since cold boot 53,043,930 words have been consed, 45,867,153 words of garbage
have
been reclaimed, and 11,658,295 words of non-garbage have been transported.
The total "scavenger work" required to accomplish this was 121,864,225 units.
Use Set GC Options to examine or modify the GC parameters.
```

The command Start GC turns on the garbage collector.

Start GC *keywords*

Turns on the garbage collector.

keywords	:Dynamic, :Ephemeral, :Immediately
:Dynamic	{yes, no} Dynamic Level of incremental GC.
:Ephemeral	{yes, no} Ephemeral Level of incremental GC.
:Immediately	{yes, no} Perform a complete garbage collection right now.

Start GC :ephemeral is recommended for general purposes. This cleans up after you as you work, keeping virtual memory requirements for garbage collecting to a minimum. When, in spite of scavenging, enough garbage has accumulated, you receive a notification. At that point you can use Start GC :immediately to do a complete garbage collection. See the section "Ephemeral-Object Garbage Collection" in *Internals, Processes, and Storage Management.*

12. Understanding Character Styles

See the section "Using Character Styles in Zmacs" in *Text Editing and Processing.*

What Is a Character Style?

A *character style* is a combination of three characteristics that describe how a character appears. These characteristics are the *family*, *face*, and *size*.

Family	Characters of the same family have a typographic integrity, so that all characters of the same family resemble one another. Examples: SWISS, DUTCH, and FIX.
Face	A modification of the family, such as BOLD or ITALIC.
Size	The size of the character, such as NORMAL or VERY-SMALL.

The character style is the grouping of the family, face, and size fields. A character style is often represented by the convention:

family.face.size

An example of a fully specified character style is:

```
SWISS.ITALIC.LARGE
```

Each element of the character style can be specified or left unspecified. A family, face, or size of NIL means to use the default value. Most characters have the following character style:

```
NIL.NIL.NIL
```

Characters of style NIL.NIL.NIL are displayed in the default character style established for the current output device.

12.1 Default Character Styles

The appearance of a character depends on two things: the character style of the character, and the default character style. Windows, buffers, files, and printers have each have default character styles for output. The default character style specifies the appearance of a character whose character style is NIL.NIL.NIL. The character's style is merged against the default character style to produce the final appearance of the character.

We recommend that you use character styles by making good use of the default character styles. You preserve the most flexibility by keeping the character style of the characters themselves as unspecified as possible. If you want to change the appearance of all characters in a Zmacs buffer, a Zmail message or a window, you can change the default character style instead of changing the character style of each character.

The default character style affects the appearance of a character on output. There is also a typein character style, which is normally NIL.NIL.NIL. The typein character style affects the character style in which characters are entered as input. If the typein character style is NIL.BOLD.NIL, any characters you enter at the keyboard have the character style NIL.BOLD.NIL. It is important to be sure that the application program can handle characters whose character style is something other than NIL.NIL.NIL, if you are going to use a typein character style other than NIL.NIL.NIL.

If you only want to change the way that characters echo, but not the way they are entered as input, you can change the echo character style. See the section "Using Character Styles in the Input Editor", page 162.

12.2 Merging Character Styles

This section gives some examples of how the character style of a character is merged against the default character style to produce a final result.

In general, we advise that you specify as little as possible when changing a character style. That is, if you want the character's face to be italic, specify only the face component and let the family and size come from the default character style.

Character Style of a Character	*Default Character Style*	*Result of Merging*
NIL.NIL.NIL	FIX.ROMAN.NORMAL	FIX.ROMAN.NORMAL
NIL.ITALIC.NIL	FIX.ROMAN.NORMAL	FIX.ITALIC.NORMAL
NIL.BOLD-ITALIC.LARGE	FIX.ROMAN.NORMAL	FIX.BOLD-ITALIC.LARGE
SWISS.BOLD.LARGER	FIX.ROMAN.NORMAL	SWISS.BOLD.LARGE

The family and face components are either NIL or the name of a family or face. The size component can be NIL, an *absolute size* (such as LARGE or VERY-SMALL) or a *relative size* (such as LARGER or SMALLER). A relative size is merged against the default size such that when you merge LARGER against NORMAL, the result is the next size larger than NORMAL.

12.3 Using Character Styles in the Input Editor

The default character style for the input editor is FIX.ROMAN.NORMAL.

You can use the Set Window Options CP command to change the default character style, the typein character style, or the echo character style. The default character style and typein character style are described elsewhere: See the section "Default Character Styles", page 161.

It is important to be sure that the application program can handle characters whose character style is something other than NIL.NIL.NIL, if you are going to use a typein character style other than NIL.NIL.NIL. If you only want to change the way that characters echo, but not the way they are entered as input, you can change the echo character style.

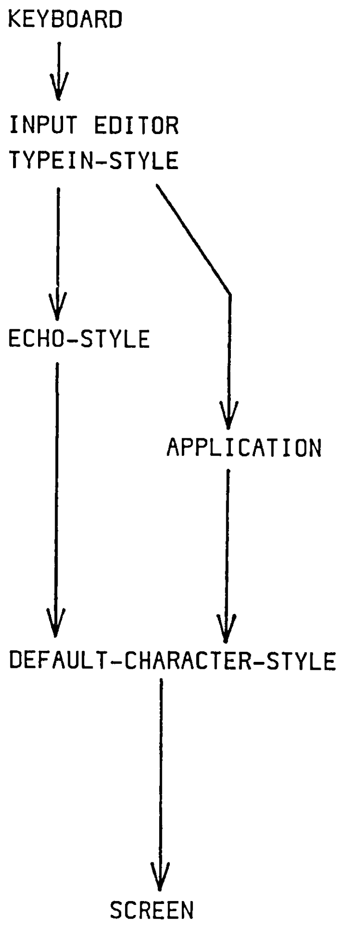
The echo character style affects the way that characters you enter at the keyboard are echoed. The appearance of characters that you type depends on: the character style of the character (which is usually the same as the typein style), which is merged against the echo character style, which is merged against the default character style.

In addition to the Set Window Options command, you can change the typein character style in the input editor by using c-m-J. You are then prompted for a character style. Enter something in the *family.face.size* convention, such as DUTCH.BOLD-ITALIC.LARGER.

12.4 Character Styles and the Lisp Listener

This section and diagram describes the role of the typein character style, the echo character style, and the default character style in the Lisp Listener.

When you type a character at the keyboard, it follows one path which eventually causes it to be echoed on the screen. The same character follows a path to the application program. The application program might produce some output, which is also displayed on the screen.



User enters a character at keyboard. This is called the input character.

The Input Editor reads the input character and sets its style to TYPEIN-STYLE. The input character proceeds to the application and and toward the screen to be echoed.

The input character is merged against the echo style.

Meanwhile, the application program receives the input character. The program performs its function and produces output, which can be in any character style.

Both the output characters from the application and the input character are merged against the default character style, which could have been modified if the application program used **with-character-style**. The characters are displayed on the screen.

For example:

Typein style is: SWISS.NIL.NIL
Echo style is: NIL.BOLD.NIL
Default character style is: FIX.ROMAN.NORMAL

The input editor reads in the input character according to the typein style, so the input character has the character style of SWISS.NIL.NIL. The input character is merged against the echo style, so it is then SWISS.BOLD.NIL. The input character is then merged against the default character style, so it is finally displayed on the screen in the character style SWISS.BOLD.NORMAL.

Meanwhile the original input character of style SWISS.NIL.NIL is sent to the application. As it runs, the application program produces output characters of style NIL.BOLD-ITALIC.NIL. The output characters are merged against the default character style; they are displayed in the character style FIX.BOLD-ITALIC.NORMAL.

Note that the application program can use **with-character-style**, **with-character-face** and so on when producing output. If this is done, the specified style information is merged against the default character style. Thus it affects both the way that the input characters are echoed and the way any output characters are displayed.

If you want to specify how your input characters appear, you can change the echo style. If you want your input characters to have the character style set to something other than NIL.NIL.NIL, you can change the typein style.

12.5 Using Character Styles in Zmail

Every message has its own default character style used for displaying the message. The default is recorded in a new **Default-Character-Style** header. If this header is not present, the message is displayed using FIX.ROMAN.NORMAL as the default character style.

A new command, (m-X) Set Message Default Character Style, may be used to change the default character style of the current message. This command is also available in the menu offered by Mouse-Right.

When composing draft messages, the new (m-X) Set Default Character Style command may be used to change the default character style for the message under composition. Again, this command is also available in the menu offered by Mouse-Right. If you do not set the default style of a draft, it will be set to FIX.ROMAN.NORMAL on transmission.

When you are in Zmail, you can use the Zmacs commands for changing character styles. For a list of available commands and a description of how to enter a new character style: See the section "Using Character Styles in Zmacs" in *Text Editing and Processing.*

12.6 Using Character Styles in Hardcopy

When you hardcopy a file, the outcome depends on the character style of the characters in the file and the printer's default values for the body character style and header character style.

The printer's defaults are stored in the printer object in the namespace database, in the attributes:

body-character-style
: The default character style to be used by this printer

headers-character-style
: The default character style to be used for the headers by this printer.

You can override the defaults stored in the printer objects by using **setq** on the variable **hardcopy:*hardcopy-default-character-styles***. Note that the value of **hardcopy:*hardcopy-default-character-styles*** is merged with the default style for the printer. See the variable **hardcopy:*hardcopy-default-character-styles***, page 109.

13. Understanding Networks and the Namespace System

13.1 Introduction to the Namespace System

The namespace database consists of a collection of *objects*. Each object has:

- A *class*: See the section "Namespace System Classes", page 167.
- *Attributes*: See the section "Namespace System Attributes", page 168.
- A *name*: See the section "Names and Namespaces", page 169.

Each type of object contains a few required attributes and many optional attributes. Note that when you are using the namespace editor, the required attributes appear with an asterisk (*) after them.

All objects except namespaces themselves are added to the namespace database by using the namespace editor, which is invoked with the CP command Edit Namespace Object, or by choosing Namespace from the System menu. See the section "Updating the Namespace Database", page 171.

13.1.1 Namespace System Classes

Every object has a *class*, which indicates its type. Each class is identified by a global-name. For a discussion of global-names: See the section "Data Types of Namespace System Attributes", page 168.

The following classes are especially important to the Symbolics system:

host	A host object represents any computer, usually connected to a network.
user	A user object represents a person who uses any of the hosts, or a daemon user, for example, a Symbolics computer.
network	A network object represents a computer network, to which some hosts are attached.
printer	A printer object represents a device for producing hardcopy.
site	A site object represents a collection of hosts, printers, and networks, grouped together in one physical location.
namespace	A namespace object represents a mapping from object names to objects.

13.1.2 Namespace System Attributes

Attributes represent characteristics of an object. Each attribute has an *indicator* (the name of the attribute) and a *value;* they work like property lists in Lisp. For example, every host has a system-type (saying which operating system it runs), every printer has a type (saying what type of printer it is), and every user has a personal-name.

Each object class has one or more required attributes. However, most attributes are optional; for example, hosts can optionally have a pretty-name, printers can have a default-font, and a user can have a home-address. Some attributes can occur more than once for a given object; for example, a host object can have multiple addresses if it is attached to multiple networks.

When editing a namespace object, you can easily determine whether an attribute is required or optional. Required attributes contain an asterisk by them, whereas optional attributes do not.

Each object class has a fixed set of required and optional attributes. You cannot create additional attributes.

13.1.3 Data Types of Namespace System Attributes

Each class has attributes defined to have specific data types. Since the actual representation of the various types of data represented in the database varies from system to system, the namespace system uses the following system-independent types:

Data type	**Value**
object-class	An object in the database, for example, a site object. See the section "Namespace System Classes", page 167.
name	A name in some namespace; name is not shared by all namespaces.
global-name	A name which is not specific to a particular namespace but is shared by all namespaces.
token	An arbitrary character string.
set	An ordered set of elements of the same data type. For example, a value can be a set of names or a set of triples.
pair	A list of two elements of specific data types; each element can be of a different data type.
triple	A list of three elements; each element can be of a different data type.

Name, *global-name* and *token* require simple values, whereas *set*, *pair* and *triple* require compound values.

Note: Namespace data types specific to the Symbolics computer are described elsewhere: See the section "Namespace System Lisp Data Types" in *Networks*.

13.1.4 Names and Namespaces

Every object has a *name*, which is a character string. Two objects of different classes can have the same name. For example, there can be a printer named george and a user named george; the two are unrelated. An object is identified by its class and its name. If you want to look up an object in the database and you know its name, you have to say "Find the printer named george" or "Find the user named george", not just "Find george".

When long-distance networks are used to link together different sites, however, the possibility of name conflicts arises; that is, two sites might use the same name in the same class for conflicting purposes. For example, suppose you had a host named orange, and you wanted to connect your site over a long-distance network to some other site that happens to have picked the name orange for one of its own hosts. Neither site is forced to change its host names just because it wants to connect to the other site.

To avoid these naming conflicts, the database can include more than one *namespace*. A namespace is a mapping from names to objects, and names in one namespace are unrelated to names in another namespace. More strictly, a namespace is a mapping from [class, name] pairs to objects, since an object is identified by its class and its name. Normally each site has one namespace, and the names of all the objects at that site are in that namespace. An object in some namespace other than your own can be referred to by a *qualified name*, which consists of the name of the namespace, a vertical bar, and the name of the object in that namespace.

For example, suppose both Harvard and Yale have computer centers. Harvard has three hosts named yellow, orange, and blue, and Yale has three hosts named apple, orange, and banana. Each computer center would have its own namespace, one named harvard and one named yale. At Harvard, the Harvard computers would be referred to by their unqualified names (yellow, orange, and blue), whereas the Yale computers would be referred to (by users at Harvard) by qualified names (yale|apple, yale|orange, and yale|banana). At Yale it would all work the other way around.

Each namespace also has a list of namespaces called *search rules*. When a name is looked up, each of the namespaces in the search rules list is consulted in turn, until an object of that name is found in one of the namespaces. If you have some other namespace in your search list, it is easier to refer to objects in that namespace, because you do not have to use qualified names unless a name conflict exists.

For example, in the scenario above, the search list for the harvard namespace could have the harvard namespace first and the yale namespace second. Then users at Harvard could refer to Yale's computers as apple, yale|orange, and banana. The qualified name is only necessary if a name conflict exists.

Actually, only some classes of objects have names that are in namespaces; other classes of objects are *globally* named, which means that the names are universal, and conflicts are not permitted. In particular, classes, namespaces, and sites are globally named; networks, hosts, printers, and users are named within namespaces. There is never a need for multiply-qualified names; the names of namespaces are global and never need to be qualified themselves.

Some namespaces do not correspond to any local site. Most large nationwide or worldwide networks have their own host-naming convention. For example, the U.S. Department of Defense Arpanet has its own set of host names, and this is considered a namespace. If a local site includes some hosts that are on the Arpanet, it might want to put the Arpanet namespace into its search list, and install gateways on its Arpanet machine so that other machines on the local network can access the Arpanet.

Some objects can also have *nicknames*. In particular, networks and hosts can have nicknames; objects of other classes cannot. A nickname serves as an alternative name for the object. Sometimes you give an object a nickname because its full name is too long to type conveniently, such as a host whose name you type frequently. However, each object has one primary name, which is always used when the object is printed.

It is possible for an object to be in several namespaces at once. For example, a host which is on both the Arpanet and a local network at some site might be in both the Arpanet namespace and the local namespace. In this case, each namespace maintains its own separate information on the object. The information from each namespace is merged before being presented to the user.

Note: Search lists are not followed recursively. If a user at Harvard looks up a name and Yale's namespace is in Harvard's search list, Yale's search list is *not* relevant.

13.2 Connecting to a Remote Host Over the Network

If your Symbolics computer is on a network and configured properly, you can access other hosts on the network with the Terminal program.

To use the Terminal program, press `SELECT T`. The prompt is:

```
Connect to host:
```

Type the name of the host to which you want to connect. The network system makes a connection, and you will see the prompt of the remote host displayed on the screen. You are now communicating directly with the remote machine.

When you are connected to a remote host, the NETWORK key provides several useful commands. For example:

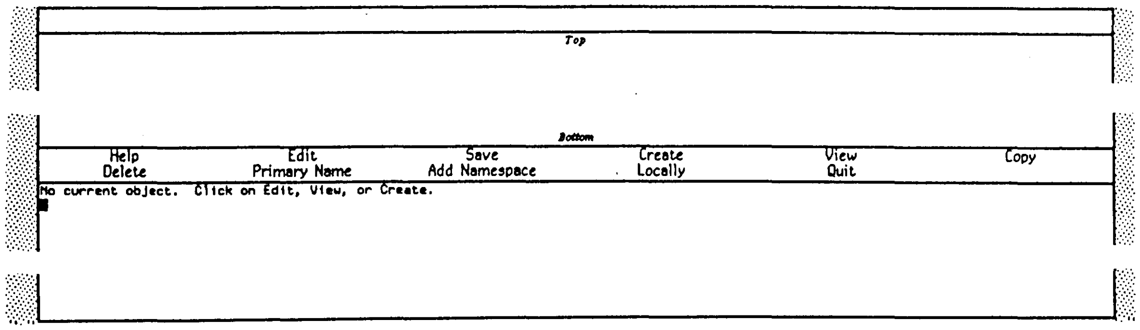
NETWORK HELP	Displays the list of options for the NETWORK key.
NETWORK L	Logs out of remote host, and breaks the connection.
NETWORK D	Disconnects without logging out first.

See the section "NETWORK Key", page 221.

If you want to use the Terminal program to log in to a remote Symbolics computer when someone is logged in to that machine, you must first enable remote login by evaluating the form **(net:remote-login-on)** on that machine. See the function **net:remote-login-on** in *Networks.*

13.3 Updating the Namespace Database

To begin editing the namespace database, use the CP command Edit Namespace Object or choose [Namespace] from the System menu. Once in the namespace window, you can use the [Edit] command to modify information stored in the database, or use the [View] command to examine information without changing it.



```
No current object.  Click on Edit, View, or Create.
```

The namespace editor window has three parts. The top pane shows the current information about the object being edited. The middle pane is the command pane; the commands that appear here are mouse-sensitive. The namespace editor uses the bottom pane to prompt you for new information.

The namespace editor commands include:

Help	Displays a brief explanation.
View	Displays information about an object for inspection but not editing.

Edit	Displays information about an object for editing.
Locally	Toggles whether to edit the local or global copy of the information for an object. The initial state is global.
Save	Saves the current information about an object.
Delete	Removes an object from the database.
Create	Adds a new object to the database.
Quit	Exits from the namespace editor, without saving the current information. If you want to save information, use [Save] before using [Quit].
Copy	Creates a new object by copying the current one.
Add Namespace	Adds an existing object to a new namespace.
Primary Name	Changes the primary name of the current object.

13.3.1 Editing a Namespace Object

First select the Namespace editor by using the Edit Namespace Object command. To edit an existing namespace object, click on [Edit]. A menu of object classes pops up. Click on the class of object you want to edit. You are prompted for the name of an object to edit. The current information for the object is retrieved from the namespace database and displayed in the top window.

The attribute fields are mouse-sensitive. Clicking on an attribute prompts you for information in the bottom window. Mouse clicks have the following meaning:

Left	Replace the information in the attribute.
Middle	Delete information in the attribute.
Right	Edit the information in the attribute.

The window can be scrolled. See the section "Scrolling with the Mouse", page 212.

Once you have finished editing the information, you have three possible ways to proceed. You can [Quit] without saving the changed information. If you are just practicing using the namespace editor, that would be appropriate.

The other two choices are to save the information locally or globally. If you save it globally, the new information is stored in the site's namespace database. If you save it locally, the new information is stored only in your machine's local copy of the namespace; these changes would affect only your machine.

The initial state of the namespace editor is the global mode. When you are in global mode the top line of the screen looks like:

Editing: Host SCRC|JUNCO

If you have clicked on [Locally], you are in local mode. The top line of the screen looks like:

Editing: Host SCRC|JUNCO (locally)

You can click on [Locally] to toggle the mode between global and local. When you are ready, click on [Save] to save the information. Then click on [Quit] to exit the namespace editor.

13.3.2 Creating a New Namespace Object

First select the Namespace editor by using the Edit Namespace Object command. To create a new namespace object, click on [Create]. A menu of object classes pops up. Click on the class of object you want to create. You are prompted for the name of the new object. A template for the information is displayed in the top window. The attributes are mouse-sensitive. Clicking on an attribute prompts you in the bottom window for the information to put in the attribute.

Note that the required attributes appear with an asterisk (*) after them. All object classes have a small number of required attributes, and several optional attributes.

You can also create a new object by copying an existing object by clicking on [Copy] and then editing the object as appropriate.

The window can be scrolled. See the section "Scrolling with the Mouse", page 212.

When you are satisfied with the information, you can enter it in the database by clicking on [Save]. Then click on [Quit] to exit the namespace editor.

For a discussion of saving (locally or globally) new information in the namespace database: See the section "Editing a Namespace Object", page 172.

14. A Brief Introduction to the Lisp World

14.1 Lisp Objects

14.1.1 Functions

A typical description of a Lisp function looks like this:

function-name *arg1 arg2* &optional *arg3 (arg4* **(foo3)***)* *function*

Adds together *arg1* and *arg2*, and then multiplies the result by *arg3*. If *arg3* is not provided, the multiplication is not done. **function-name** returns a list whose first element is this result and whose second element is *arg4*.
Examples:

```
(function-name 3 4) => (7 4)
(function-name 1 2 2 'bar) => (6 bar)
```

The word "&optional" in the list of arguments tells you that all of the arguments past this point are optional. The default value of an argument can be specified explicitly, as with *arg4*, whose default value is the result of evaluating the form **(foo 3)**. If no default value is specified, it is the symbol **nil**. This syntax is used in lambda-lists in the language. (For more information on lambda-lists: See the section "Evaluating a Function Form" in *Symbolics Common Lisp.*) Argument lists can also contain "&rest", which is part of the same syntax.

Note that the documentation uses several *fonts*, or typefaces. In a function description, for example, the name of the function is in boldface in the first line, and the arguments are in italics. Within the text, printed representations of Lisp objects are in the same boldface font, such as **(+ foo 56)**, and argument references are italicized, such as *arg1* and *arg2*.

Other fonts are used as follows:

"Typein" or "example" font (`function-name`)
Indicates something you are expected to type. This font is also used for Lisp examples that are set off from the text and in some cases for information, such as a prompt, that appears on the screen.

"Key" font (RETURN, c-L)
For keystrokes mentioned in running text.

14.1.2 Macros and Special Forms

The descriptions of special forms and macros look like the descriptions of these imaginary ones:

do-three-times *form* *Special Form*

Evaluates *form* three times and returns the result of the third evaluation.

with-foo-bound-to-nil *form...* *Macro*

Evaluates the *forms* with the symbol **foo** bound to **nil**. It expands as follows:

```
(with-foo-bound-to-nil
form1
form2 ...) ==>
(let ((foo nil))
form1
form2 ...)
```

Since special forms and macros are the mechanism by which the syntax of Lisp is extended, their descriptions must describe both their syntax and their semantics; unlike functions, which follow a simple consistent set of rules, each special form is idiosyncratic. The syntax is displayed on the first line of the description using the following conventions.

- Italicized words are names of parts of the form that are referred to in the descriptive text. They are not arguments, even though they resemble the italicized words in the first line of a function description.

- Parentheses ("( )") stand for themselves.

- Brackets ("**[ ]**") indicate that what they enclose is optional.

- Ellipses ("**...**") indicate that the subform (italicized word or parenthesized list) that precedes them can be repeated any number of times (possibly no times at all).

- Braces followed by ellipses ("{ }**...**") indicate that what they enclose can be repeated any number of times. Thus, the first line of the description of a special form is a "template" for what an instance of that special form would look like, with the surrounding parentheses removed.

The syntax of some special forms is too complicated to fit comfortably into this style; the first line of the description of such a special form contains only the name, and the syntax is given by example in the body of the description.

The semantics of a special form includes not only its contract, but also which subforms are evaluated and what the returned value is. Usually this is clarified with one or more examples.

A convention used by many special forms is that all of their subforms after the first few are described as "*body...*". This means that the remaining subforms constitute the "body" of this special form; they are Lisp forms that are evaluated one after another in some environment established by the special form.

This imaginary special form exhibits all of the syntactic features:

twiddle-frob *[(frob option...)] {parameter value}...* *Special Form*

Twiddles the parameters of *frob*, which defaults to **default-frob** if not specified. Each *parameter* is the name of one of the adjustable parameters of a frob; each *value* is what value to set that parameter to. Any number of *parameter/value* pairs can be specified. If any *options* are specified, they are keywords that select which safety checks to override while twiddling the parameters. If neither *frob* nor any *options* are specified, the list of them can be omitted and the form can begin directly with the first *parameter* name.

frob and the *values* are evaluated; the *parameters* and *options* are syntactic keywords and are not evaluated. The returned value is the frob whose parameters were adjusted. An error is signalled if any safety check is violated.

14.1.3 Flavors, Flavor Operations, and Init Options

Flavors themselves are documented by the name of the flavor.

Flavor operations are described in three ways: as methods, as generic functions, and as messages. When it is important to show the exact flavor for which the method is defined, methods are described by their function specs. Init options are documented by the function spec of the method.

When a method is implemented for a set of flavors (such as all streams), it is documented by the name of message or generic function it implements.

The following examples are taken from the documentation.

sys:network-error *Flavor*

This set includes errors signalled by networks. These are generic network errors that are used uniformly for any supported networks. This flavor is built on **error**.

:clear-window of **tv:sheet** *Method*

Erase the whole window and move the cursor position to the upper left corner of the window.

:tyo *char* *Message*

The stream will output the character *char*. For example, if **s** is bound to a stream, then the following form will output a "B" to the stream:

```
(send s :tyo #\B)
```

For binary output streams, the argument is a nonnegative number rather than specifically a character.

dbg:special-command-p *condition special-command* *Generic Function*

Returns **t** if *command-type* is a valid Debugger special command for this condition object; otherwise, returns **nil**.

The compatible message for **dbg:special-command-p** is:

:special-command-p

:bottom *bottom-edge* (for **tv:sheet**) *Init Option*

Specifies the y-coordinate of the bottom edge of the window.

14.1.4 Variables

Descriptions of variables ("special" or "global" variables) look like this:

typical-variable *Variable*

The variable **typical-variable** has a typical value....

14.2 The Lisp Top Level

These functions constitute the Lisp top level and its associated functions.

si:lisp-top-level *Function*

This is the first function called in the initial Lisp environment. It calls **sys:lisp-reinitialize**, clears the screen, and calls **si:lisp-top-level1**.

sys:lisp-reinitialize &optional (*called-by-user* **t**) *Function*

This function restarts the Lisp system, resetting the values of various global constants and initializing the error system.

si:lisp-top-level1 *stream* *Function*

This is the actual top-level loop. It reads a form from **zl:standard-input**, evaluates it, prints the result (with slashification) to **zl:standard-output**, and repeats indefinitely. If several values are returned by the form, all of them will be printed. Also the values of *, +, -, **zl:/**, ++, **, +++, and *** are maintained.

prinl *Variable*

The value of this variable is normally **nil**. If it is non-**nil**, then the read-eval-print loop uses its value instead of the definition of **prinl** to print the values returned by functions. This hook lets you control how things are printed by all read-eval-print loops – the Lisp top level, the **zl:break** function, and any utility programs that include a read-eval-print loop. It does not affect output from programs that call the **prin1** function or any of its relatives such as **print** and **zl:format**; to do that, you need more information on customizing the printer. See the section "Output Functions" in *Reference Guide to Streams, Files, and I/O*. If you set **prin1** to a new function, remember that the read-eval-print loop expects the function to print the value but not to output a Return character or any other delimiters.

14.3 Logging in

After cold booting, you are in a window named Dynamic Lisp Listener 1. You are now ready to log in. If your login name is KJones, you can log in in any of the following ways: (Note that the examples are given in upper and lower case, but the machine is not case sensitive. You can use all upper case, all lower case, or mixed case as you prefer.)

- To log into the default host machine, using your init file, type
```
Login KJones
```
- To log into your machine, without your init file, type
```
Login KJones :init file none
```
- To log into another machine "sc3", using your init file, type
```
Login KJones :host sc3
```

If the host machine you log in to is a timesharing computer system, you must have a directory and account on that host machine.

For more information about logging in: See the section "Login Functions and Variables".

For more information about how to write init files: See the section "Customizing Genera", page 93.

14.4 Some Utility Functions

zwei:save-all-files &optional (*ask* **t**) *Function*

This function is useful in emergencies in which you have modified material in Zmacs buffers that needs to be saved, but the editor is partially broken. This function does what the editor command Save File Buffers (m-X) does, but it stays away from redisplay and other advanced facilities so that it might work if other things are broken.

zwei:zmail-save-all-files is similar, but saves mail files from Zmail.

ed &optional *thing* *Function*

ed is the main Lisp function for entering Zmacs. Select Activity Zmacs is the command for entering Zmacs.

(ed) or **(ed nil)** enters Zmacs, leaving everything as it was when you last left Zmacs. If Zmacs has not yet been used in the current session, it is initialized and an empty buffer created.

(ed t) enters Zmacs, and creates and selects an empty buffer.

If the argument is a pathname or a string, the **ed** function enters Zmacs, and finds or creates a buffer with the specified file in it. This is the same as the Edit File command.

If the argument is a symbol that is defined as a function, Zmacs will try to find the source definition for that function for the user to edit. This is the same as the Edit Definition command.

Finally, if the argument is the symbol **zwei:reload**, Zmacs will be reinitialized. All existing buffers will be lost, so use this only if you have to.

zl:dired &optional (*pathname* "") *Function*

Puts up a window and edits the directory named by *pathname*, which defaults to the last file opened. While editing a directory you may view, edit, compare, hardcopy, and delete the files it contains. While in the directory editor press the HELP key for further information. This is similar to the Edit Directory command, except that Edit Directory enters Zmacs and runs Dired (m-X).

zl:mail &optional *initial-destination initial-body prompt initial-idx* *bug-report* (*make-subject* **(zl:memq zwei:*require-subjects* (quote (t :init))))** *initial-subject* *Function*

Sends mail by putting up a window in which you can compose the mail.

initial-destination is a symbol or string that is the recipient.

initial-body is a string that is the initial contents of the mail. If these are unspecified they can be typed in during composition of the mail. Press the END key to send the mail and return from the **zl:mail** function.

prompt and *initial-idx* are used by programs, such as **zl:bug**, that call **zl:mail**. *prompt* is a string printed in the minibuffer of the mail window created by **zl:mail**. *initial-idx* positions point in the mail window.

zl:bug &optional (*system* **(quote zwei:lispm)**) *additional-body prompt point-before-additional-body* (*make-subject* **(zl:memq zwei:*require-subjects* (quote (t :init :bug)))**) *initial-subject* — *Function*

Reports a bug. This is the same as the Report Bug command. **zl:bug** is like **zl:mail** but includes information about the system version and what machine you are on in the text of the message.

system is the name of the faulty program (a symbol or a string). It defaults to **zl-user:lispm** (the Lisp Machine system itself). This information is important to the maintainers of the faulty program; it aids them in reproducing the bug and in determining whether it is one that is already being worked on or has already been fixed.

additional-body is user-supplied text appended to the information supplied by the system.

prompt is text supplied by the system printed in the minibuffer of the mail window concerning the bug-mail you are sending.

point-before-additional-body is a position for point supplied by the system.

zl:qsend &optional *destination message* — *Macro*

Sends interactive messages to users on other machines on the network.

destination is normally a string of the form *name@host*, to specify the recipient. If you omit the *@host* part and just give a name, **zl:qsend** looks at all of the Lisp Machines at your site to find any that *name* is logged into; if the user is logged into one Lisp Machine, it is used as the host; if more than one, **zl:qsend** asks you which one you mean. If you leave out *destination* altogether, doing just **(zl:qsend)**, Converse is selected as if you had pressed `SELECT C`.

message should be a string. For example:

```
(qsend kjones@wombat "Want to go to lunch?")
```

If *message* is omitted, **zl:qsend** asks you to type in a message. You should type in the contents of your message and press `END` when you are done.

The input editor is used while you type in a message to **zl:qsend**. So you get some editing power, although not as much as with full Converse (since the latter uses Zwei). See the section "Editing Your Input", page 134. **zl:qsend** predates Converse and is retained for compatibility.

15. Checking on What the Machine is Doing

15.1 Poking Around in the Lisp World

This section describes a number of functions, most of which are not normally used in programs, but are "commands", that is, things that you type directly at Lisp. They are useful for finding information about your current state or about the Lisp world in general.

who-calls *symbol* &optional *how* *Function*

who-calls tries to find all the functions in the Lisp world that call *symbol*. *how* may be **nil**, meaning all ways to call the symbol, a keyword, meaning only find *symbol* called as *keyword*, or a list of keywords. The permitted keywords are:

:variable
:function
:microcoded-function
:constant
:flavor
:instance-variable
:macro
:defined-constant
:condition
:flavor-component
:generic-function

who-calls takes a single symbol as its argument. It no longer takes a list of symbols. The package filtering has been removed.

who-calls prints one line of information for each caller it finds. It also returns a list of the names of all the callers.

who-calls works only on bound symbols. To locate unbound symbols: See the function **si:who-calls-unbound-functions**, page 184.

The compiler records, as part of its debugging-info property, which macros were expanded and which functions were optimized away, with the exception of basic parts of the language, such as **car** and **when**. This information is used by **who-calls** and similar functions. Thus you can use **who-calls** for macros. **who-calls** can also find callers of open-coded functions, such as substitutable functions.

The **who-calls** database is created at site configuration time using the function **si:enable-who-calls**.

si:enable-who-calls &optional *mode* *Function*

This command takes an argument which is the mode in which the database should be enabled. If you need the full database, use **si:enable-who-calls** during site configuration time. If you do not need a full database you can create an incremental database by choosing a suitable mode.

mode can be one of the following:

:all	Creates a full callers database. This takes many minutes and about 2000 pages of storage. **:all** also queries about the old state.
:all-remake	Creates a full callers database but does not query about the old state. This takes many minutes and about 2000 pages of storage.
:new	Creates a callers database that includes only new functions.
:all-no-make	Creates a callers database that includes only new functions. When you follow it with a **si:full-gc** the entire database is created. This takes many minutes and about 2000 pages of storage.
:explicit	Enables items to be added to the callers database explicitly by using **si:add-files-to-who-calls-database** or **si:add-system-to-who-calls-database**.

After you create the database, you should run **si:compress-who-calls-database.**

si:compress-who-calls-database *Function*

si:compress-who-calls-database makes the **who-calls** database more compact and more efficient. You should call this function after **si:enable-who-calls.** If you used **(si:enable-who-calls ':all),** **si:compress-who-calls-database** takes a long time to complete its job. However, it is faster than using **si:full-gc** and the result can be saved using incremental disk save.

The editor has a command, List Callers (m-X), that is similar to **who-calls.**

si:who-calls-unbound-functions *Function*

si:who-calls-unbound-functions Searches the compiled code for any calls through a symbol that is not currently defined as a function. This is useful for finding errors such as functions whose names you misspelled or forgot to write.

what-files-call *symbol-or-symbols* &optional *how* *Function*

Similar to **who-calls** but returns a list of the pathnames of all the files that contain functions that **who-calls** would have printed out. This is useful if you need to recompile and/or edit all those files. *how* may be **nil**, meaning all ways to call the symbol, a keyword, meaning only find *symbol* called as *keyword*, or a list of keywords. The permitted keywords are:

:variable
:function
:microcoded-function
:constant
:flavor
:instance-variable
:macro
:defined-constant
:condition
:flavor-component
:generic-function

zl:apropos *apropos-substring* &optional *pkg* (*do-packages-used-by* **t**) *do-packages-used* *Function*

Tries to find all symbols whose print-names contain *apropos-substring* as a substring. When it finds a symbol, it prints out the symbol's name; if the symbol is defined as a function and/or bound to a value, it tells you so, and prints the names of the arguments (if any) to the function. It checks all symbols in a certain set of packages. The set always includes *pkg*. If *do-packages-used-by* is true, the set also includes all packages that use *pkg*. If *do-packages-used* is true, the set also includes all packages that *pkg* uses. *pkg* defaults to the **zl-user:global** package, so normally all packages are searched. **zl:apropos** returns a list of all the symbols it finds. This is similar to the Find Symbol command, except that Find Symbol only searches the current package unless you specify otherwise.

where-is *pname* *Function*

Finds all symbols named *pname* and prints on **zl:standard-output** a description of each symbol. The symbol's home package and name are printed. If the symbol is present in a different package than its home package (that is, it has been imported), that fact is printed. A list of the packages from which the symbol is accessible is printed, in alphabetical order. **where-is** searches all packages that exist, except for invisible packages.

If *pname* is a string it is converted to uppercase, since most symbols' names use uppercase letters. If *pname* is a symbol, its exact name is used.

where-is returns a list of the symbols it found.

The **find-all-symbols** function is the primitive that does what **where-is** does without printing anything.

describe *anything* &optional *no-complaints* *Function*

Tries to tell you all the interesting information about any object except array contents). **describe** knows about arrays, symbols, all types of numbers, packages, stack groups, closures, instances, structures, compiled functions, and locatives, and prints out the attributes of each in human-readable form. Sometimes it describes something that it finds inside something else; such recursive descriptions are indented appropriately. For instance, **describe** of a symbol tells you about the symbol's value, its definition, and each of its properties. **describe** of a floating-point number shows you its internal representation in a way that is useful for tracking down roundoff errors and the like.

If *anything* is a named-structure, **describe** handles it specially. To understand this: See the section "Named Structures" in *Symbolics Common Lisp*. First it gets the named-structure symbol, and sees whether its function knows about the **:describe** operation. If the operation is known, it applies the function to two arguments: the symbol **:describe**, and the named-structure itself. Otherwise, it looks on the named-structure symbol for information that might have been left by **zl:defstruct**; this information would tell it the symbolic names for the entries in the structure. **describe** knows how to use the names to print out each field's name and contents.

describe describes an instance by sending it the **:describe** message. The default method prints the names and values of the instance variables.

This is the same as the Show Object command.

describe always returns its argument, in case you want to do something else to it.

inspect &optional *object* *Function*

A window-oriented version of **describe**. See the section "How the Inspector Works", page 204.

disassemble *function* *Function*

function is either a compiled function, or a symbol or function spec whose definition is a compiled function. **disassemble** prints out a human-readable version of the macroinstructions in *function*.

15.1.1 Variables for Examining the Lisp World

- *Variable*

While a form is being evaluated by a read-eval-print loop, - is bound to the form itself.

+ *Variable*

While a form is being evaluated by a read-eval-print loop, + is bound to the previous form that was read by the loop.

***** *Variable*

While a form is being evaluated by a read-eval-print loop, * is bound to the result printed the last time through the loop. If several values were printed (because of a multiple-value return), * is bound to the first value. If no result was printed, * is not changed.

zl:/ *Variable*

While a form is being evaluated by a read-eval-print loop, **zl:/** is bound to a list of the results printed the last time through the loop.

++ *Variable*

++ holds the previous value of +, that is, the form evaluated two interactions ago.

+++ *Variable*

+++ holds the previous value of ++.

****** *Variable*

** holds the previous value of *, that is, the result of the form evaluated two interactions ago.

******* *Variable*

*** holds the previous value of **.

grindef &rest *fcns* *Special Form*

Prints the definitions of one or more functions, with indentation to make the code readable. Certain other "pretty-printing" transformations are performed:

- The **quote** special form is represented with the ' character.
- Displacing macros are printed as the original code rather than the result of macro expansion.
- The code resulting from the backquote (‘) reader macro is represented in terms of ‘.

The subforms to **grindef** are the function specs whose definitions are to be printed; ordinarily, **grindef** is used with a form such as **(grindef foo)** to print the definition of **foo**. When one of these subforms is a symbol, if the symbol has a value its value is prettily printed also. Definitions are printed as **defun** special forms, and values are printed as **setq** special forms.

If a function is compiled, **grindef** says so and tries to find its previous interpreted definition by looking on an associated property list. See the function **uncompile** in *Program Development Utilities*. This works only if the function's interpreted definition was once in force; if the definition of the function was simply loaded from a BIN file, **grindef** does not find the interpreted definition and cannot do anything useful.

With no subforms, **grindef** assumes the same arguments as when it was last called.

zl:break &optional *tag* (*conditional* **t**) *Special Form*

Enters a breakpoint loop, which is similar to a Lisp top-level loop. **(break** *tag***)** always enters the loop; **(break** *tag conditional***)** evaluates *conditional* and only enter the break loop if it returns non-**nil**. If the break loop is entered, **zl:break** prints out:

```
;Breakpoint tag; Resume to continue, Abort to quit.
```

The standard values for any variables are checked. If **zl:break** rebinds any of these standard variables, it warns you that it has done so. **zl:break** then enters a loop reading, evaluating, and printing forms. A difference between a break loop and the top-level loop is that when reading a form, **zl:break** checks for the following special cases: If the ABORT key is pressed, control is returned to the previous break or Debugger, or to top level if there is none. If the RESUME key is pressed, **zl:break** returns **nil**. If the list **(return** *form***)** is typed, **zl:break** evaluates *form* and returns the result.

Inside the **zl:break** loop, the streams **zl:standard-output**, **zl:standard-input**, and **zl:query-io** are bound to be synonymous to **zl:terminal-io**; **zl:terminal-io** itself is not rebound. Several other internal system variables are bound, and you can add your own symbols to be bound by pushing elements onto the value of the variable **sys:*break-bindings***. (See the variable **sys:*break-bindings***, page 188.)

If *tag* is omitted, it defaults to **nil**.

There are two easy ways to write a breakpoint into your program: **(zl:break)** gets a read-eval-print loop, and **(zl:dbg)** gets the Debugger. (These are the programmatic equivalents of the SUSPEND and m-SUSPEND keys on the keyboard.)

sys:*break-bindings* *Variable*

When **zl:break** is called, it binds some special variables under control of the list that is the value of **sys:*break-bindings***. Each element of the list is a list of two elements: a variable and a form that is evaluated to produce the value to bind it to. The bindings happen sequentially. You can **zl:push** things on this list (adding to the front of it), but should not replace the list wholesale since several of the variable bindings on this list are essential to the operation of **zl:break**.

dbg:*debugger-bindings* *Variable*

When the Debugger is entered, it binds some special variables under control of the list that is the value of **dbg:*debugger-bindings***. Each element of the list is a list of two elements: a variable and a form that is evaluated to produce the value to bind it to. The bindings happen sequentially. You can **zl:push** things on this list (adding to the front of it), but should not replace the list wholesale since several of the variable bindings on this list are essential to the operation of the Debugger.

15.2 Utility Functions

print-sends &optional (*stream* **zl:standard-output**) *Function*

Prints out all messages you have received (but not messages you have sent), in forward chronological order, to *stream*. Converse is more useful for looking at your messages, but this function predates Converse and is retained for compatibility.

zl:print-notifications &optional (*from* **0**) (*to* **(1- (zl:length tv:notification-history))**) *Function*

Reprints any notifications that have been received. The difference between notifications and sends is that sends come from other users, while notifications are asynchronous messages from the Lisp Machine system itself. If *from* or *to* is specified, prints only part of the notifications list.

Example: `(print-notifications 0 4)` prints the five most recent notifications.

This is the same as the Show Notifications command.

si:show-login-history &optional (*whole-history* **si:login-history**) *Function*

Prints one line for each time the login command has been used since the world was last cold booted. It also shows the logins done during the creation of the world load. Each line contains the name of the user who logged in, the name of the machine on which the world load was running at that time, and the date and time. This command also shows the name of an init file, if one was loaded. If you cold boot, log in, and then do Show Login History, the last line refers to your own login and all previous lines refer to logins that were done before doing Save World (or running **zl:disk-save**).

This information is useful to determine how many times a world load has been disk-saved, on what machines it was disk-saved, and who disk-saved it.

The first couple of lines do not contain any date or time, because they were made during the initial construction of the world load before it found out the current time. Names of users at other sites that are not in the local site's namespace search list are qualified with the site's namespace name and a vertical bar. The user LISP-MACHINE is the dummy user used by **si:login-to-sys-host** when new world loads are created.

This function replaces **si:print-login-history**.

zl:hostat &rest *hosts* *Function*

Asks each of the *hosts* for its status, and prints the results. If no hosts are specified, asks all hosts on the Chaosnet. Hosts can be specified by either name or octal number.

For each host, a line is displayed that either says that the host is not responding or gives metering information for the host's network attachments. If a host is not responding, probably it is down or there is no such host at that address. A Lisp Machine can fail to respond if it is looping inside **without-interrupts** or paging extremely heavily, such that it is simply unable to respond within a reasonable amount of time.

To abort the host status report produced by **zl:hostat** or `FUNCTION H`, press `c-ABORT`.

zl-user:uptime &rest *hosts* *Function*

Queries the specified *hosts*, asking them for their "uptime"; each host responds by saying how long it has been up and running. **zl-user:uptime** prints out the results. If **zl-user:uptime** reports that a host is "not responding", either the host is not responding to the network, or it does not support the UPTIME protocol.

The **zl-user:uptime** function is a variant of **zl:hostat**.

15.3 Dribble Files

Sometimes it is useful to have a more permanent record of what is happening on your screen when a program is running. Dribble files allow you to save the output from or interaction with a program in a file or a buffer. Formerly such files were called wallpaper files because the resulting long strips of paper output resembled wallpaper and were sometimes posted on the wall to demonstrate the operation of a program. Now that display consoles are in wide use, the files are referred to as dribble files because the output "dribbles" out of the running program.

zl:dribble-start *pathname* &optional *editor-p* (*concatenate-p* **t**) *Function*

Opens *filename* as a "dribble file". It rebinds **zl:standard-input** and **zl:standard-output** so that all of the terminal interaction is directed to the file as well as to the terminal. If *editor-p* is non-**nil**, then instead of opening *filename* on the file computer, **zl:dribble-start** directs the terminal interaction into a Zmacs buffer whose name is *filename*, creating it if it does not exist.

zl:dribble-end *Function*

Closes the file opened by **zl:dribble-start** and resets the I/O streams.

15.4 zl:status and zl:sstatus

The **zl:status** and **zl:sstatus** special forms exist for compatibility with Maclisp. Programs that are designed to run in both Maclisp and Zetalisp can use **zl:status** to determine in which one they are running. Also, **(zl:sstatus zl-user:feature zl-user:...)** can be used as it is in Maclisp.

zl:status *status-function* &optional (*item* **nil** *item-p*) *Special Form*

(zl:status zl-user:features) returns a list of symbols indicating features of the Lisp environment. The default list for the Lisp Machine is:

```
(:DEFSTORAGE :LOOP :DEFSTRUCT :LISPM :SYMBOLICS 3600 :CHAOS :SORT
 :FASLOAD :STRING :NEWIO :ROMAN :TRACE :GRINDEF :GRIND)
```

The value of this list will be kept up to date as features are added or removed from the Lisp Machine system. Most important is the symbol **:lispm**; this indicates that the program is executing on the Lisp Machine. The order of this list should not be depended on, and might not be the same as shown above.

The following symbols in the features list can be used to distinguish different Lisp implementations, using the #+ and #- reader syntax.

Three symbols indicate which Lisp Machine hardware is running:

:lispm	Any kind of Lisp Machine, as opposed to Maclisp
:cadr	An M.I.T. CADR
:3600	A 3600-family machine

One symbol indicates which kind of Lisp Machine software is running:

:symbolics	Symbolics software

See the section "Sharp-sign Reader Macros" in *Reference Guide to Streams, Files, and I/O.*

(status feature *symbol***)** returns **t** if *symbol* is on the **(zl:status zl-user:features)** list, otherwise **nil**.

(status nofeature *symbol***)** returns **t** if *symbol* is not on the **(zl:status zl-user:features)** list, otherwise **nil**.

(zl:status zl-user:userid) returns the name of the logged-in user.

(zl:status zl-user:tabsize) returns the number of spaces per tab stop (always 8). Note that this can actually be changed on a per-window basis: however, the **zl:status** function always returns the default value of 8.

(zl:status zl-user:opsys) returns the name of the operating system, always the symbol **:lispm**.

(zl:status zl-user:site) returns the name of the local machine, for example, **"mit-lispm-6"**. Note that this is not the same as the value of **zl:site-name**.

(zl:status zl:status) returns a list of all **zl:status** operations.

(zl:status zl:sstatus) returns a list of all **zl:sstatus** operations.

zl:sstatus *status-function item* *Special Form*

(sstatus feature *symbol***)** adds *symbol* to the list of features.

(sstatus nofeature *symbol***)** removes *symbol* from the list of features.

15.5 Using Peek

15.5.1 Overview of Peek

You start up Peek by pressing `SELECT P`, by using the Select Activity Peek command, or by evaluating **(zl:peek)**.

The Peek program gives a dynamic display of various kinds of system status.

When you start up Peek, a menu is displayed at the top, with one item for each system-status mode. The item for the currently selected mode is highlighted in reverse video. If you click on one of the items with the mouse, Peek switches to that mode. Pressing one of the keyboard keys as listed in the Help message also switches Peek to the mode associated with that key. The Help message is a Peek mode; Peek starts out in this mode.

Pressing the `HELP` key displays the Help message.

The `Q` command exits Peek and returns you to the window from which Peek was invoked.

Most of the modes are dynamic: they update some part of the displayed status periodically. The time interval between updates can be set using the `Z` command. Pressing *n*`Z`, where *n* is some number, sets the time interval between updates to *n* seconds. Using the `Z` command does not otherwise affect the mode that is running.

Some of the items displayed in the modes are mouse sensitive. These items, and the operations that can be performed by clicking the mouse on them, vary from mode to mode. Often clicking the mouse on an item gives you a menu of things to do to that object.

The Peek window has scrolling capabilities, for use when the status display is longer than the available display area. `SCROLL` or `c-V` scrolls the window forward (towards the bottom), `m-SCROLL` or `m-V` scrolls it backward (towards the top).

As long as the Peek window is exposed, it continues to update its display. Thus a Peek window can be used to examine things being done in other windows in real time.

15.5.2 Peek Modes

Processes (P)

In Processes mode, invoked by pressing `P` or by clicking on the [Processes] menu item, you see all the processes running in your environment, one line for each. The process names are mouse sensitive; clicking on one of them pops up a menu of operations that can be performed:

Arrest (or Un-Arrest)
: Arrest causes the process to stop immediately. Unarrest causes it to pick up where it left off and continue.

Flush
: Causes the process to go into the state Wait Forever. This is one way to stop a runaway process that is monopolizing your machine and not responding to any other commands. A process that has been flushed can be looked at with the debugger or inspector and can be reset.

Reset	Causes the process to start over in its initialized state. This is one way to get out of stuck states when other commands do not work.
Kill	Causes the process to go away completely.
Debugger	Enters the Debugger to look at the process.
Describe	Displays information about the process.
Inspect	Enters the Inspector to look at the process.

See the section "Introduction to Processes" in *Internals, Processes, and Storage Management.*

Areas (A)

Areas mode, invoked by pressing A or by clicking on [Areas], shows you information about your machine's memory. The first line is hardware information: the amount of physical memory on the machine, the amount of swapping space remaining in virtual memory, and how many wired pages of memory the machine has. The following lines show all the areas in virtual memory, one line for each. For each area you are shown how many regions it contains, what percentage of it is free, and the number of words (of the total) in use. Clicking on an area inserts detailed information about each region: its number, its starting address, its length, how many words are used, its type, and its GC status. See the section "Areas" in *Internals, Processes, and Storage Management.*

Meters (M)

Meters mode, invoked by pressing M or by clicking on [Meters], shows you a list of all the metering variables for storage, the garbage collector, and the disk. There are two types of storage and disk meters:

Timers	Timers have names that start with **zl-user:*ms-time-** and keep a total of the milleseconds spent in some activity.
Counts	Counts have names that start with **zl-user:*count-** and keep a running total of the number of times some event has occurred.

The garbage collector meters fall into two groups according to which part of the garbage collector they pertain to: the scavenger or the transporter. See the section "Operation of the Garbage Collector".

File System (F)

File System mode, invoked by pressing F or by clicking on [File System], provides you information about your network connections for file operations. For each host the access path, protocol, user-id, host or server unit number, and connection state are listed. For active connections information about the actual packet flow is also given. The various items are mouse sensitive. For hosts, you can get hostat information, do a file reset, log in remotely, find out who is on the remote machine, and send a message to the machine. You can reset, describe, or inspect data channels, and close streams.

Resetting an access path makes the server on a foreign host go away, which might be useful to free resources on that host or if you suspect that the server is not working correctly.

Windows (W)

Windows mode, invoked by pressing W or clicking on [Windows], shows you all the active windows in your environment with the panes they contain. This allows you to see the hierarchical structure of your environment. The items are mouse sensitive. Clicking on a window name pops up a menu of operations that you can perform on the window.

Servers (S)

Clicking on [Servers] or pressing S puts Peek in Servers mode. If your machine is a server (for example, a file server), Servers mode shows the status of each active server.

Network (N)

Network mode, invoked by pressing N or by clicking on [Network], shows information about the networks connected to your machine. For each network there are three headings for information:

Active connections	The data channels that your machine has opened to another machine or machines on the network.
Meters	Information about the data flow (packets) between your machine and other machines on the network.
Routing table	A list of all the subnets and for each the route to take to send packets to a host on that subnet.

To view the information under one of these headings, you click on the heading. The hosts and data channels in the list of active connections are mouse sensitive.

For hosts, you can get hostat information, do a file reset, login remotely, find out who is on the remote machine, and send a message to the machine. You can reset, describe, or inspect data channels.

Information about the hardware network interface is also displayed, as well as metering variables for the networks.

Hostat (H)

Clicking on [Hostat] or pressing H starts polling all the machines connected to the local network. For each host on the network a line of information is displayed. Those machines that do not respond to the poll are marked as "Host not responding". You terminate the display by pressing c-ABORT.

Help and Quit

Clicking on the [Help] menu item or pressing HELP displays the help information that is displayed when Peek is selected the first time.

Clicking on [Quit] or pressing Q buries the Peek window and returns you to the window from which you invoked Peek.

16. Tools for Lisp Debugging

16.1 Overview of the Debugger

Genera: The Symbolics Software Environment offers you a host of powerful debugging tools. The most comprehensive of these tools is the Symbolics interactive Debugger and its window-oriented counterpart, the Window Debugger.

Other debugging tools, also known as debugging aids, are:

- The *Trace* facility, which performs certain debugging actions at the time a function is called or at the time a function returns. See the section "Tracing Function Execution" in *Program Development Utilities.*

- The *Advise* facility, which modifies the behavior of a function. See the section "Advising a Function" in *Program Development Utilities.*

- The *Step* facility, which allows you to execute forms in your program, one at a time, so that you can examine what is happening when execution suspends at every "step." See the section "Stepping Through an Evaluation" in *Program Development Utilities.* The Debugger's :Single Step command also performs stepping. See the section "Single Step Command" in *Program Development Utilities.*

- The *evalhook* facility, which allows you to get a particular Lisp form whenever the evaluator is called. The Step facility also uses evalhook. See the section "**evalhook**" in *Program Development Utilities.*

Another tool related to debugging is the *Inspector*, which is a window-oriented program that lets you inspect data objects and their components. See the section "Using the Inspector" in *Program Development Utilities.*

In the Genera software environment, unlike more traditional programming environments, you do not have to explicitly include the Debugger when you compile your programs. Generally, you can debug your code as you write it without having to perform a series of complicated compiling, loading, and executing procedures between source code development and debugging.

Because Symbolics Dynamic Windows and other user-interface features allow you to many Symbolics activities simultaneously – Zmacs, Zmail, the file system, the Dynamic Lisp Listener, and so on – debugging becomes an easy task, regardless of how many system activities you are using. You can move in and out of the Debugger as easily as you can move in and out of any other process in Genera. For example, the Debugger command, :Edit Function, brings up a function for editing in a Zmacs editor window. This is useful when you have found the function that caused the error and want to edit that function immediately. Another command, :Mail Bug Report, brings up a mail message window and puts a backtrace into the message to be mailed as a bug report. While composing the bug report, you can switch back and forth between the Debugger and the mail-sending window.

As in any other process, you can suspend the Debugger or use split-screen windows to run two or more processes simultaneously.

The Symbolics Debugger is there whenever you need it. The Debugger is signalled whenever an error occurs in your program's execution or the execution of a system function. That is, your machine brings you into the Debugger whenever it encounters an error that is not bound to a condition handler, for example, when you reference an unbound variable. See the section "Entering and Exiting the Debugger" in *Program Development Utilities.* Once in the Debugger, you are given a choice of actions that can correct the error. These actions are called *proceed* and *restart options.* See the section "Proceeding and Restarting in the Debugger" in *Program Development Utilities.*

You can also enter the Debugger explicitly, at any time, by pressing `m-SUSPEND` or `c-m-SUSPEND`, or make your program signal the Debugger by inserting the **cl:break** or **zl:dbg** function into your program code. See the section "Entering and Exiting the Debugger" in *Program Development Utilities.*

Upon Debugger entry, besides selecting one of the proceed and restart options, you can enter any of the Debugger's 58 commands. These commands are full-form English commands, built on normal command processor (CP) commands. In fact, several Debugger commands are also CP commands. For more detailed information on Debugger commands: See the section "Entering a Debugger Command" in *Program Development Utilities.*

In the Debugger you can also evaluate a form in the lexical, user-program context of the current frame. This context is referred to as the Debugger's *evaluation environment.* You can think of the Debugger's evaluation environment as a special read-eval-print loop that not only evaluates forms but also evaluates local variables while the execution of its lexical function is suspended. For more detailed information on the evaluation environment: See the section "Evaluating a Form in the Debugger" in *Program Development Utilities.*

Like all other output in the Genera software environment, Debugger output is mouse sensitive, so you can perform many useful Debugger operations using the mouse. For more detailed information on mouse capabilities: See the section "Using the Mouse in the Debugger" in *Program Development Utilities.*

The Debugger also provides some online help facilities. For more detailed information on help facilities: See the section "Getting Help with Debugger Commands".

For complete information on the uses of these features and other Debugger features – plus a list of descriptions for all Debugger commands: See the section "Using the Debugger" in *Program Development Utilities.* For information on the Window Debugger: See the section "The Window Debugger".

In general, you would use the Debugger when:

- Your program triggers the Debugger because garbage – an unbound variable or too many arguments perhaps – was passed to a function, and you want to find out where the garbage came from. See the section "Analyze Frame Command" in *Program Development Utilities.*

- You want to see what's happening in the sequence of function calls just executed, including a history of these function calls, the argument values passed, the local-variables values, the source code, and the compiled code. See the section "Show Backtrace Command" in *Program Development Utilities.* Also: See the section "Debugger Commands for Viewing a Stack Frame" in *Program Development Utilities.*

- You want to find out who or what is referencing a special variable or any other location in memory. See the section "Monitor Variable Command", page 247.

- You want to remember all the Debugger's key-binding command accelerators, like `c-B` and `c-m-F`, and you wish they were associated with real English commands, like :Show Backtrace and :Show Function. See the section "Debugger Command Descriptions" in *Program Development Utilities.*

- You want to perform debugging operations using the mouse. See the section "Using the Mouse in the Debugger" in *Program Development Utilities.*

- You want to continue program execution, proceed from the error, restart a function, return from a function, or throw through a function. See the section "Debugger Commands to Continue Execution" in *Program Development Utilities.*

- Your condition handler breaks, and you want to call the Debugger when this handler is encountered so that you can debug it. See the section "Enable Condition Tracing Command" in *Program Development Utilities.*

- You want to edit your function's source code in Zmacs immediately after you have found the error using the Debugger. See the section "Edit Function Command" in *Program Development Utilities.*

- You want to put a Debugger backtrace into a mail message and send this message as a bug report. See the section "Mail Bug Report Command" in *Program Development Utilities.*

- You want to use Debugger breakpoint commands, instead of using the Trace facility or inserting a function in your code, to set Debugger breakpoints. See the section "Commands for Breakpoints and Single Stepping".

16.1.1 Overview of Debugger Commands

The Debugger comprises 58 full-form English commands, which are implemented as CP commands. Debugger commands are entered inside the Debugger at the Debugger's command prompt, a right arrow (→). Commands fall into eight general categories:

Commands for viewing a stack frame

Commands for stack motion

Commands for general information display

Commands to continue execution

Trap commands

Commands for breakpoints and single stepping

Commands for system transfer

Miscellaneous commands

Most Debugger commands have corresponding key-binding accelerators, which means you can press a combination of one or more keys in place of the command. For example, you can type the accelerator c-E instead of the command :Edit Function.

Like CP commands, most Debugger commands have keywords that you can use to modify the command's behavior.

There are 13 Debugger commands that share the global command table with CP commands. Therefore, you can enter these commands in the CP as well as the Debugger. They are:

:Clear All Breakpoints

:Clear Breakpoint

:Disable Condition Tracing

:Edit Function

:Enable Condition Tracing

:Monitor Variable

:Set Breakpoint

:Set Stack Size

:Show Breakpoints

:Show Compiled Code

:Show Monitored Locations

:Show Source Code

:Unmonitor Variable

Note, however, that you must type a preceding colon with every command entered in the Debugger; for example, you must type :Set Breakpoint in the Debugger.

For complete information on Debugger commands: See the section "Entering a Debugger Command" in *Program Development Utilities.*

16.1.2 Overview of Debugger Evaluation Environment

In the Debugger, you can evaluate a form as easily as you can in a Dynamic Lisp Listener read-eval-print loop. Evaluating a form in the Debugger, however, is particularly useful because you are evaluating the form in the context of a user program and the current stack frame. This means you can see the value of Lisp objects at the point in program execution where an error occurred or at the precise place in your program where you explicitly suspend execution and signal the Debugger. You can even see the values of lexical (local) variables at the point where execution suspends.

Evaluating a form in the Debugger is a simple task. If you type a character other than the first character in a Debugger command – a colon or accelerator key – the

Debugger immediately brings you into its evaluation environment. In other words, just type the form. Evaluation happens automatically.

For complete information on how to evaluate a form in the Debugger: See the section "Evaluating a Form in the Debugger" in *Program Development Utilities.*

16.1.3 Overview of Debugger Mouse Capabilities

When the output generated by Debugger commands is displayed on a Dynamic Window, the output is mouse sensitive. You can perform several useful debugging operations simply by using the mouse to click on something. Some of these operations include: setting a breakpoint, monitoring a variable or another location in memory, evaluating a form, editing a function, setting the current frame, and choosing a proceed or restart option.

Besides performing certain mouse operations by clicking directly on displayed Debugger output, you can use menus to perform the usual large variety of other types of operations on Debugger output, just as you can with other output generated anywhere else in the Genera software environment.

For more information on using the mouse in the Debugger: See the section "Using the Mouse in the Debugger" in *Program Development Utilities.*

16.1.4 Overview of Debugger Help Facilities

The Debugger provides online help for Debugger commands and their components, such as keywords. You can get help for all Debugger commands by typing `c-HELP`, which displays brief command descriptions and available key-binding accelerators. For more information about Debugger help: See the section "Getting Help for Debugger Commands" in *Program Development Utilities.*

16.2 Flavor Examiner

The Flavor Examiner enables you to examine flavors, methods, generic functions, and internal flavor functions defined in the Lisp environment. It is a congenial environment for using the Show Flavor commands.

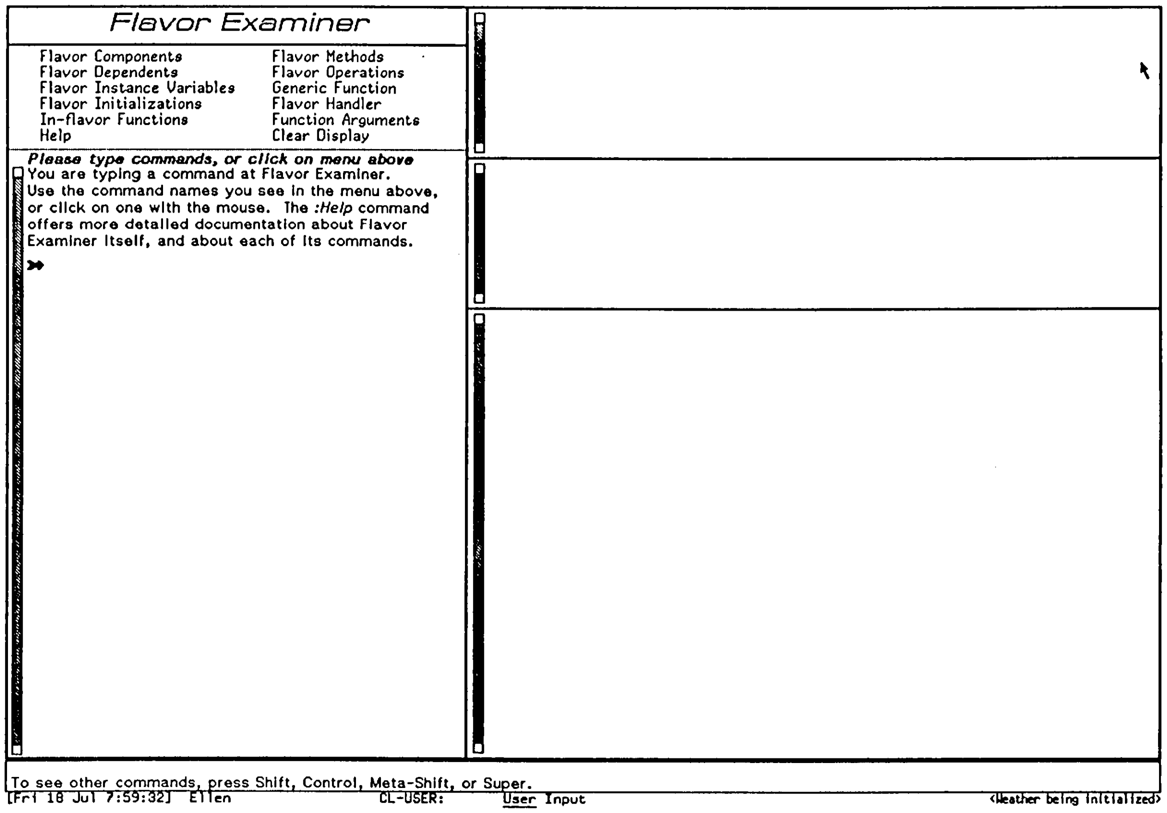
You can select the Flavor Examiner with `SELECT X`, or with the Select Activity Flavor Examiner command.

Figure 22 shows the initial window.

The Flavor Examiner window is divided into five panes.

Menu of Commands – the top-left pane

The top-left pane offers a menu of flavor-related commands, such as `Flavor`



```
Please type commands, or click on menu above
You are typing a command at Flavor Examiner.
Use the command names you see in the menu above,
or click on one with the mouse. The :Help command
offers more detailed documentation about Flavor
Examiner itself, and about each of its commands.
```

```
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Fri 18 Jul 7:59:32]  Ellen            CL-USER:      User Input               <Weather being initialized>
```

Figure 22. Flavor Examiner Window

`Components`; this is the same as the Show Flavor Components command. You can choose one of these commands by clicking left or right. Clicking left makes the command appear in the Command Input Pane. Clicking right makes the command appear and also displays the command's arguments, in a form that you can edit.

The `HELP` command displays documentation on the flavor-related commands. The `HELP` key provides information on all the CP commands you can enter.

The Flavor Examiner offers two commands for clearing and refreshing the display. The `CLEAR DISPLAY` command clears the display from the three output panes; it first asks for confirmation. The `REFRESH DISPLAY` command displays the information on the screen again.

When you click left or right on a command name, the command appears in the Command Input Pane.

Command Input Pane – the bottom-left pane

The bottom-left pane is a command processor window. If you click on commands in the Menu of Commands, the commands appear in this window. You can enter arguments (or commands) by typing them at the keyboard. This pane saves the history of all commands entered. You can click on the scroll bar to show different parts of the history.

You are not restricted to the commands in the Menu of Commands. You can give any command processor command.

The output of all commands appears in the Main Command Output Pane.

Main Command Output Pane – the bottom-right pane

Each command's output appears here. This pane saves the history of the output of all flavor-related commands. You can use the scroll bar to show different parts of the history.

Parts of the output of flavor-related commands are mouse-sensitive. You can make use of that by clicking on a flavor name or method name to enter it as an argument to another command.

If you give commands that are not flavor-related (such as the Show Host command), the output appears in a typeout window in the Main Command Output Pane. This kind of output is not saved in the history of this pane. The typeout window is itself a dynamic window with its own history.

When the output of the current command appears in the Main Command Output Pane, the output of the previous command is copied to the Previous Command Output Pane.

Previous Command Output Pane – the middle-right pane

This pane displays the output of the previous command. This pane does not save a history, but the second-to-last command is copied to the Second-to-Last Command Output Pane.

Second-to-last Command Output Pane – the top-right pane

This pane displays the output of the second-to-last command. This pane does not save a history. When another command is given, the contents of the Previous Command Output Pane are copied to this pane. Similarly, the contents of the Main Command Output Pane are copied to the Previous Command Output Pane.

16.3 How the Inspector Works

The Inspector is a window-oriented program for inspecting data structures. When you ask to inspect a particular object, its components are displayed. The particular components depend on the type of object; for example, the components of a list are its elements, and those of a symbol are its value binding, function definition, and property list.

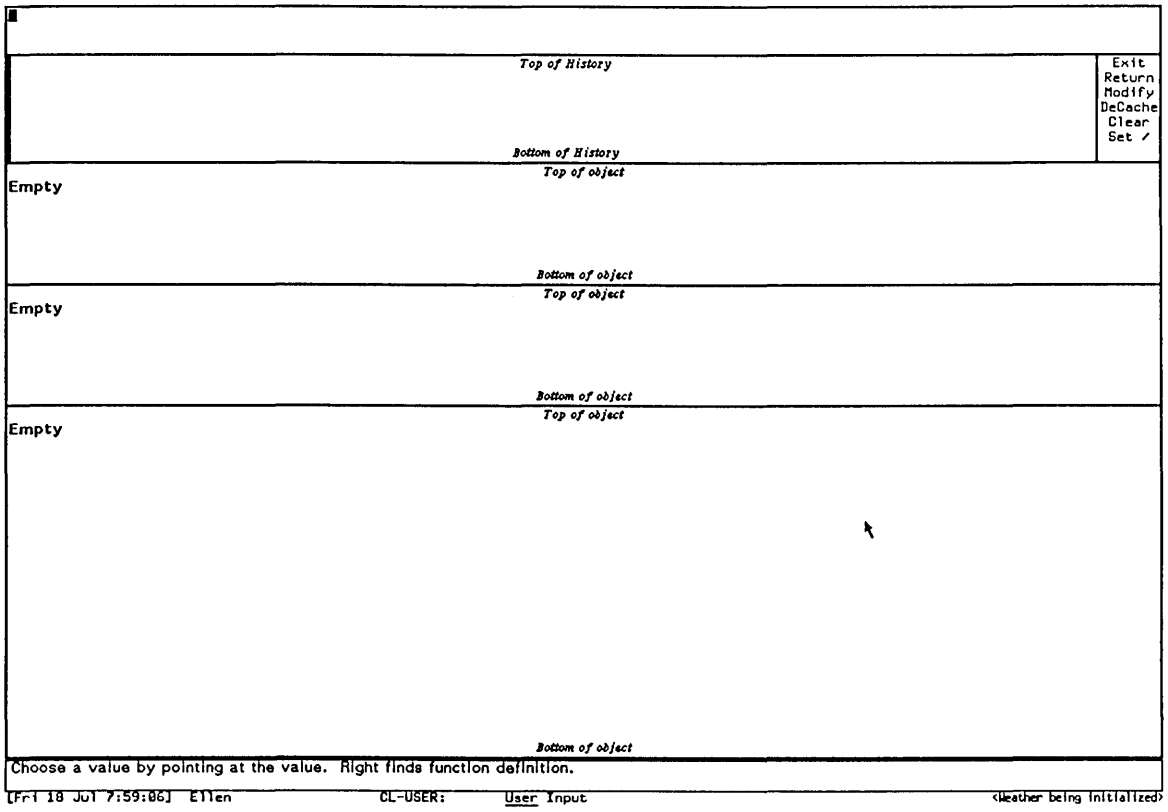
The component objects displayed on the screen by the Inspector are mouse-sensitive, allowing you to do something to that object, such as inspect it, modify it, or give it as the argument to a function. Choose these operations from the menu pane at the top-right part of the screen.

When you click on a component object itself, that component object gets inspected. It expands to fill the window and its components are shown. In this way, you can explore a complex data structure, looking into the relationships between objects and the values of their components.

The Inspector can be part of another program or it can be used standalone; for example, the Window Debugger can utilize some of the panes of the Inspector. Note, however, that although the display looks the same as that of the standalone Inspector, the handling of the mouse buttons depends upon the particular program being run.

Figure 23 shows the standalone Inspector window. The display consists of the following panes, from top to bottom:

- A small interaction pane
- A history pane and menu pane
- Some number of inspection panes (three by default)



```
Choose a value by pointing at the value.  Right finds function definition.
[Fri 18 Jul 7:59:06]  Ellen          CL-USER:      User Input                 <Weather being initialized>
```

Figure 23. The Inspector

16.4 Entering and Leaving the Inspector

You can enter the standalone Inspector via:

- Select Activity *Inspector*
- `SELECT I`
- [Inspect] in the System menu
- The Inspect command, which inspects its argument, if any
- The **inspect** function, which inspects its argument, if any

Warning: If you enter with the Inspect command or the **inspect** function, the Inspector is not a separate activity from the Lisp Listener in which you invoke it. In this case you cannot use `SELECT L` to return to the Lisp Listener; you should *always* exit via the [Exit] or [Return] option in the Inspector menu. If you forget and exit the Inspector by selecting another activity, you might need to use `c-m-ABORT` to return the Lisp Listener to its normal state.

See the section "The Inspector" in *Program Development Utilities.*

17. Quick Reference

17.1 General Help Facilities

c-ABORT	Aborts the function currently executing.
c-G	Aborts a command while it is being entered, unselects the region, or unmerges a kill; that is, resets "state".
HELP A *string*	Shows every command containing *string* (try HELP A Paragr or HELP A Buffer).
HELP C *x*	Explains the action of any command (try HELP C c-K as an example).
HELP D *string*	Describes a command (try HELP D Query Rep).
HELP L	Displays the last 60 keys pressed.
SUSPEND	Starts a Lisp Listener (return from it with RESUME).

17.2 Zmacs Help Facilities

Undo (m-X)	Reverts to buffer before last kill, unkill, fill, sort, or similar complex command.
c-Y	Yanks back the last thing killed.
m-Y	After a c-Y, yanks back things previously killed; used after a c-Y to cycle through the kill ring.

17.3 Extended Commands

Extended commands (the m-X commands) put you in a small area of the screen with full editing capabilities (a *minibuffer*) for entering names and arguments. Several kinds of help are available in a minibuffer.

COMPLETE	Completes as much of the current command as possible.
HELP	Gives information about special characters and possible completions.
c-?	Shows possible completions for the command currently being entered.
END or RETURN	Complete the command, and then execute it.
c-/	Does an apropos on what has been typed so far.

17.4 Writing Files

c-X c-S	Writes the current buffer into a new version of the current file name.
c-X c-W	Writes the current buffer into a file with a different name.
Save All Files (m-X)	Offers to save each file whose buffer has been modified.

17.5 Buffer Operations

c-X c-F	Gets a file into a buffer for editing.
c-X B	Selects a different buffer (prompts; default is the last one).
c-X c-B	Displays a menu of available buffers; lines are mouse-sensitive.
c-X K	Kills a buffer (prompts for which one; default is current one).
m-<	Moves to the beginning of the current buffer.
m->	Moves to the end of the current buffer.

17.6 Character Operations

c-B	Moves left (back) a character.
c-F	Moves right (forward) a character.
c-P	Moves up (previous) a character.
c-N	Moves down (next) a character.
RUBOUT	Deletes a character left.
c-D	Deletes a character right.
c-T	Transposes the two characters around point; if at the end of a line, transposes the two characters before point, ht -> th.

17.7 Word Operations

m-B	Moves left (back) a word.
m-F	Moves right (forward) a word.
m-RUBOUT	Kills a word left (c-Y yanks it back at point).
m-D	Kills a word right (c-Y yanks it back at point).
m-T	Transposes the two words around point (if only -> only if).
m-C	Capitalizes the word following point.
m-L	Lower-cases the word following point.
m-U	Upper-cases the word following point.

17.8 Line Operations

c-A	Moves to the beginning of the line.
c-E	Moves to the end of the line.
c-O	Opens up a line for typing.
c-X c-O	Closes up any blank lines around point.
CLEAR-INPUT	Kills from the beginning of the line to point (c-Y yanks it back at point).
c-K	Kills from point to the end of the line (c-Y yanks it back at point).

17.9 Sentence Operations

m-A	Moves to the beginning of the sentence.
m-E	Moves to the end of the sentence.
c-X RUBOUT	Kills from the beginning of the sentence to point (c-Y yanks it back at point).
m-K	Kills from point to the end of the sentence (c-Y yanks it back at point).

17.10 Paragraph Operations

m-[	Moves to the beginning of the paragraph.
m-]	Moves to the end of the paragraph.
m-Q	Fills the current paragraph (see HELP A Auto fill).
n c-X F	Sets the fill column to *n* (example: c-6 c-5 c-X F).

17.11 Screen Operations

SCROLL or c-V	Shows next screen.
m-SCROLL or m-V	Shows previous screen.
c-0 c-L	Moves the line where point is to line 0 (top) of the screen.
c-m-R	Repositions the window to display all of the current definition, if possible.
c-m-L	Selects the most recently selected buffer in this window.

17.12 Region Operations

`c-SPACE`	Sets the mark, a delimiter of a region. Move the cursor from mark to create a region; (the editor underlines to show the region). Use with region commands `c-W`, `m-W`, and `c-Y`.
`c-W`	Kills region (`c-Y` yanks it back at point).
`m-W`	"Saves" region (`c-Y` yanks it back at point).
`c-Y`	Yanks back the last thing killed.

17.13 Window Operations

`c-X 2`	Splits the screen in two windows, using the current buffer and the previously selected buffer (the one that `c-m-L` would select).
`c-X 1`	Resumes single window, using the current window.
`c-X O`	Moves cursor to other window.
`c-m-V`	Shows next screen of the buffer in the other window; with a numeric argument scrolls that number of lines – positive for the usual direction, negative for the reverse direction.
`c-X 4`	Splits the screen into two windows and asks what should be shown in the other window.

17.14 Search and Replace

`c-S` *string*	"Incremental" search; searches while you are entering the string; terminate search with `END`.
`c-R` *string*	"Incremental" backward search; terminate search with `END`.
`c-%` *string1* `RETURN` *string2* `RETURN`	Replaces *string1* with *string2* throughout.
`m-%` *string1* `RETURN` *string2* `RETURN`	Replaces *string1* with *string2* throughout, querying for each occurrence of *string1*; press `SPACE` meaning "do it", `RUBOUT` meaning "skip", or `HELP` to see all options; (see `HELP C m-%`).

18. Quick Summary of Mouse Functions

18.1 Mouse Cursor Shape

These are some of the more common mouse cursors:

A thin arrow pointing North by Northwest (up and to the left). This is the default mouse cursor. The mouse documentation line indicates any special commands. If the mouse is over a partially exposed window, clicking left will select that window. Clicking `sh-Mouse-Right` will get you the System menu.

A thin arrow pointing North by Northeast (up and to the right). This means the mouse is in an editor window. There are a number of editor commands on the mouse buttons. See the section "Mouse Documentation Line in Zmacs" in *Text Editing and Processing.*

A slightly thicker arrow pointing North (straight up). The editor uses this to show that it is asking you for the name of a function or for a symbol. If you point the mouse at a function name, and stop moving it, the name will light up and you can click to select it.

A small `x`. This is used when the mouse cursor should be unobtrusive, for instance in menus.

18.2 Mouse Gestures on Dynamic Windows

`Mouse-Left`	On a directory listing, does a Show File of the file. In Other contexts yanks the command line.
`sh-Mouse-Left`	Like Left, but also activates. Click `sh-Mouse-Left` on a command line to yank and activate the command.
`c-Mouse-Left`	Marks a region, `s-W` pushes marked region on the kill ring.
`Mouse-Middle`	On a Lisp Object does a **describe** of the Object.
`c-Mouse-Middle`	Yank the word the mouse is over. Useful for using arbitrary text to compose commands, for example after a Show Mail, click c-Mouse-Middle on a pathname mentioned in a mail message as an argument to Show File.
`Mouse-Right` on an object	Pops up a menu of possible operations on the object.
`m-sh-Mouse-R`	Gets the menu of window operations.

`m-Mouse-Left` in Zmacs
: Edit Definition. Hold down the left button and move the mouse around to see what is mouse sensitive.

`m-Mouse-Middle` in Zmacs
: Evaluate form. Hold down the middle button and move the mouse to see what is mouse sensitive.

18.3 Scrolling with the Mouse

Windows display "contents" that are too big to fit entirely in the window. When this is the case, you see only a portion of the contents, and you can scroll the contents up and down using the mouse.

Dynamic windows all have scroll bars along one side. The default position is along the left side. You can position the scroll bar along the right edge if you prefer.

When the mouse is moved over the scroll bar its cursor becomes a double-headed arrow. A gray area in the scroll bar indicates what portion of the window's contents is visible. The vertical position of the gray area within the scroll bar shows the position of the visible portion of the window's contents relative to the whole. At the top and bottom of the scroll bar are small boxes.

Clicking the mouse in the box at the top scrolls by lines. The mouse clicks are:

Left	Scroll by one line (next line)
Middle	Scroll to the top (first available screen) of the window.

Clicking the mouse in the box at the bottom scrolls by screens. The mouse clicks are:

Left	Scroll to the next screen.
Middle	Scroll to the end of the window (last available screen).
Right	Scroll to the previous screen.

Clicking the mouse in the scroll bar beside the middle of window scrolls proportionally. The mouse clicks are:

Left	Next screen, making this line the top line of the screen.
sh-Left	Previous screen, making this line the last line of the screen.
Middle	Put window here.

Right	Previous screen, placing the line that is currently the top line on the screen here.

The Lisp Listener window can also be scrolled horizontally. `s-SCROLL` scrolls the window to allow you to see what is to the right. `m-s-SCROLL` scrolls to the left.

Other scrolling conventions with the mouse are:

- A fat arrow, pointing up or down. This indicates you are in a scrolling zone. Moving the mouse slowly in the direction of the arrow scrolls the window, revealing more of the text in the direction the arrow points.

- Scrolling zones often say *more above* or *more below* in small italic letters. Clicking on one of these legends scrolls the window up and down by its height, thus you see the next or previous windowful. When the top or bottom of the window contents is reached, so that it is not possible to scroll any farther in one direction, the legend in the scrolling zone changes to indicate this.

19. Index of Special Function Keys

19.1 Introduction

This is a quick reference guide to the Symbolics 3600-family special function keys. Most of these keys have the same function in any window. However, a few of them perform differently in different contexts.

19.2 ABORT Key

ABORT	When read by a program, the program stops what it is doing and returns to its "command loop". Lisp Listeners, for example, respond to ABORT by throwing back to the read-eval-print loop (top level or **zl:break**). Note that ABORT takes effect when it is read, not when it is pressed; it does not stop a running program.
c-ABORT	Aborts the operation currently being performed by the process to which you are typing, immediately (not when it is read). For instance, this forces a Lisp Listener to abandon the current computation and return to its read-eval-print loop.
m-ABORT	When read by a program, the program stops what it is doing and returns through all levels of commands to its "top level". Lisp Listeners, for example, throw completely out of their computation, including any **zl:break** levels, then start a new read-eval-print loop.
c-m-ABORT	A combination of c-ABORT and m-ABORT, this immediately throws out of all levels of computation and restarts the process to which you type it.

19.3 BACKSPACE Key

In a Lisp Listener, BACKSPACE moves the cursor back one character, as does c-B, so that you can insert additional text or edit. In Zmacs, Converse, and Zmail message windows, it inserts a backspace character into the buffer. In the main Zmail window it scrolls the current message backward (as do m-SCROLL and m-V).

19.4 CLEAR INPUT Key

Usually erases the expression you are typing. In an editor buffer CLEAR INPUT erases from the location of your cursor to the beginning of the current line. If you are at the beginning of the line, it erases the previous line.

19.5 COMPLETE Key

Completes as much as possible of partially typed commands. In the Document Examiner, COMPLETE works on topic names. In Zmacs and Zmail, when completion is available for a command or pathname in the minibuffer, the word (Completion) appears in the mode line.

m-COMPLETE in the command processor displays a menu of the arguments and keywords for the command you are typing. You can then specify the arguments and keywords from the menu using the mouse or the keyboard. See the section "Using Menus", page 29.

19.6 END Key

Marks the end of input to many programs. Single-line input can be terminated with RETURN or END. END terminates multiple-line input where RETURN is used to separate lines. When you are typing Lisp input, balanced parentheses terminate expressions and END is not used. However, if you use the input editor to yank a previous command or expression, END terminates it. See the section "Editing Your Input", page 134.

19.7 ESCAPE Key

Displays the input editor history. c-ESCAPE displays the global kill history. Sends Escape/Altmode (octal 033) in the Terminal program.

19.8 FUNCTION Key

This key is a prefix for a family of commands relating to the screen, which you can type at any time, no matter what program you are running.

19.8.1 Display and Hardcopy Commands

The FUNCTION commands that control screen display and hardcopying are:

RUBOUT	Does nothing; press this key to cancel FUNCTION if you typed the latter by accident.
CLEAR INPUT	Discards typeahead.
REFRESH	Clears and redisplays all windows.
A	Arrests the process shown in the status line. FUNCTION - A resumes the process.
B	Buries the currently selected window, if any – that is, it moves it underneath all other windows. This brings up the previously selected window, which is automatically selected.
C	Complements the entire screen. An argument of 1 means white-on-black; an argument of 0 means black-on-white.
c-C	Complements the selected window, with the same argument as FUNCTION C.
m-C	Complements the mouse documentation line, with the same argument as FUNCTION C.
F	Shows users logged in on other machines on your network. Arguments can be assigned to shows users logged in on various machines at your site. FUNCTION 0 F prompts you for a specific user or host to show. Giving a user name followed by /w displays the information in the user's namespace entry.
H	Shows status of network hosts. With an argument, it prompts for hosts.
M	Controls global MORE processing. No argument means toggle, 0 means turn off, 1 means turn on.
c-M	Controls MORE processing for the selected window. The arguments are the same as for FUNCTION M.
O	Selects another exposed window.
Q	Hardcopies the entire screen.

`c-Q`	Hardcopies the selected window.
`m-Q`	Hardcopies the entire screen, minus the status and mouse documentation lines.

19.8.2 Selection and Notification Commands

The `FUNCTION` commands that control window selection and notification are:

`S` Selects the most recently selected window. With an argument *n* (default is 2), it selects the *n*th previously selected window and rotates the top *n* windows. An argument of 1 rotates through all windows (a negative argument rotates in the other direction); 0 selects a window that requires attention (for example, to report an error).

`T` Controls the selected window's input and output notification characteristics. If an attempt is made to output to a window when it is not exposed, one of three things can happen:

- The program can simply wait until the window is exposed.
- It can send a notification that it wants to type out and then wait.
- It can quietly type out "in the background"; when the window is next exposed the output becomes visible.

Similarly, if an attempt is made to read input from a window that is not selected (and has no typed-ahead input in it), the program can either wait for the window to become selected, or send a notification that it wants input and then wait.

The `FUNCTION T` command controls these characteristics based on its argument, as follows:

no argument	If output notification is off, turns input and output notification on; otherwise turns input and output notification off. This essentially toggles the current state.
0	Turns input and output notification off.
1	Turns input and output notification on.
2	Turns output notification on, and input notification off.
3	Turns output notification off, and input notification on.
4	Allows output to proceed in the background, and turns input notification on.

	5	Allows output to proceed in the background, and turns input notification off.

W — Controls the status line. With no argument, the status line is redisplayed. The arguments control the process the status line watches. The options are:

0	Gives a menu of all processes, and freezes the status line on the process you select. When the status line is frozen on a process, the name of that process appears where your user ID normally would (next to the date and time), and the status line does not change to another process when you select a new window.
1	The status line watches whatever process is using the keyboard, and changes processes when you select a new window. This is the default initial state.
2	Changes the status line so that it displays the name of the process instead of the name of the user. This also freezes the status line on that process; normally the status line switches to display a different process whenever the window system tells it to. Use this if you see an unexpected state in the status line. It will help you find out what process is in that state; you might find that you are not talking to the process you think you should be.
3	Rotates the status line among all processes.
4	Rotates the status line in the other direction.

19.8.3 Recovering From Stuck States

The following FUNCTION commands should all be used with caution.

ESCAPE	Helps you recover from stuck states such as "Output Hold" and "Sheet Lock".
c-A	Arrests all processes except the one shown in the status line and critical system processes, such as the keyboard and mouse processes. FUNCTION - c-A resumes all processes arrested by this command.
SUSPEND	Gets to the cold-load stream.

[illegible]

c-T	Deexposes temporary windows. This is useful if the system seems to be hung because there is a temporary window on top of the window that is trying to type out.
c-CLEAR INPUT	Clears window system locks. This is a last resort, although not as drastic as warm booting. Use this when none of the windows will talk to you, when you cannot get a System menu, and so on.

19.9 HELP Key

The key labelled HELP looks up context-dependent documentation.

HELP	Shows documentation available for the current activity. In some programs, c-HELP, m-HELP, and so on, provide additional documentation.
c-HELP	Shows a list of input editor commands (when typed at a Lisp Listener).
sy-HELP	Shows a list of the special function keys and the special character keys.
SELECT HELP	Shows programs and utilities that you can select using the SELECT key.
FUNCTION HELP	Shows a list of useful functions that you can invoke using the FUNCTION key.

19.10 LINE Key

The function of this key varies considerably. It is used as a command by the Debugger, and sends a Line Feed character in the Terminal program. In the editor it behaves like a RETURN followed by a TAB to the indentation level appropriate to the mode of the editor. See the section "TAB Key", page 225.

19.11 LOCAL Key

On 3670 and 3640 consoles this key controls local console functions:

LOCAL-G	Rings the bell.

LOCAL-D	Makes the screen dimmer.
LOCAL-B	Makes the screen brighter.
LOCAL-Q	Makes the audio quieter.
LOCAL-L	Makes the audio louder.
LOCAL-*n* LOCAL-C	Changes the contrast of the screen. *n* is a digit between 1 and 4. 4 is greatest contrast.

Related Lisp functions:

See the function **tv:screen-brightness**, page 97.
See the function **sys:console-volume**, page 98.

19.12 NETWORK Key

This key is used to get the attention of the Terminal program. You must be connected to a host via the Terminal program before you can use this key. See the section "Connecting to a Remote Host Over the Network", page 170.

Once connected, commands are given by pressing NETWORK and another single character.

The following commands are available:

NETWORK HELP	Display the list of options for the NETWORK key.
NETWORK A	Send an ATTN (in Telnet, a new Telnet "Interrupt Process").
NETWORK D	Disconnect without logging out first.
NETWORK L	Log out of remote host, and break the connection.
NETWORK Q	Quit, by disconnecting and deselecting this window.
NETWORK M	Toggle More processing.
NETWORK X	Enter an extended network command; see below.

More complicated commands are entered with the extended command, NETWORK X. This command invokes a Choose Variable Values window.

NETWORK X provides the capability to tailor the following:

- The escape character. The default is NETWORK.
- Whether characters overstrike or erase. Characters erase by default.

- Whether More processing is enabled. More processing is enabled by default.
- Whether to enable the *wallpaper* facility, which logs host output to a file. By default, wallpaper is not enabled.
- The filename of the wallpaper file.
- For Telnet, what level of filtering and interpretation is placed on the characters; for example, whether Imlac terminal codes are interpreted in host output.

19.13 PAGE Key

In Zmacs (in searches and after c-Q) this key inserts a page separator character, which displays as PAGE in a box.

19.14 REFRESH Key

Usually erases and redisplays the selected window.

19.15 REPEAT Key

Repeats the key pressed while the REPEAT key is held down. You can press and hold down a key and then press the REPEAT key, or you can hold down the REPEAT key and press a key. Once the repetition starts, it continues until you lift your finger from the REPEAT key. You can lift your finger from the first key and press another while still holding down REPEAT and that key starts to repeat. Pressing the REPEAT key without pressing any other key does nothing. The REPEAT key is enabled by default, but you can disable and re-enable it by setting the variable **si:*kbd-repeat-key-enabled-p*** with **setf**.

si:*kbd-repeat-key-enabled-p* *Variable*

Controls whether or not the REPEAT key is enabled. The default is **t**. It can be set using **setf**:

```
(setf si:*kbd-repeat-key-enabled-p* nil)
```

Setting **si:*kbd-repeat-key-enabled-p*** to **nil** turns off repeating using the REPEAT key.

There are two variables to control the frequency of the repetition and the delay before repetition starts.

si:*kbd-repetition-interval* *Variable*

Controls the speed of repetition of characters when the REPEAT key is held down, in sixtieths of a second. Its default is 2, which is a thirtieth of a second between repeated characters.

si:*kbd-repeat-key-initial-delay* *Variable*

Controls how long you must hold down a key before repetition with the REPEAT key starts, in sixtieths of a second. The default is 0, meaning repetition starts as soon as the REPEAT key and another key are depressed.

In addition to the REPEAT key, you can have keys repeat if they are held down. See the section "Auto-repeat", page 9.

19.16 RESUME Key

Continues from the **zl:break** function and the Debugger. In the Terminal program this sends a Backspace character.

19.17 RETURN Key

"Carriage return" or end of line. The exact significance of carriage return varies according to context.

19.18 RUBOUT Key

Usually erases the last character typed.

19.19 SCROLL Key

Scrolls the display forward. m-SCROLL scrolls it backward.

c-SCROLL	Initiates scrolling of the history for the typeout window of current window. You use this in conjunction with c-m-SCROLL to make use of the history in your Zmacs or Zmail windows.
c-m-SCROLL	Used after c-SCROLL, scrolls backward through the history of the current typeout window. As with any dynamic window, previous commands and arguments are mouse sensitive and can be re-executed or used in composing new commands.

19.20 SELECT Key

This key is a prefix for a family of commands, generally used to select a window of a specified type, such as a Lisp Listener or Zmail. The current list is:

C	Converse
D	Document Examiner
E	Editor
F	File system maintenance
I	Inspector
L	Lisp
M	Zmail
N	Notifications
P	Peek
Q	Frame-Up
T	Terminal
X	Flavor Examiner

SELECT c- creates a new window of the specified type.

19.21 SUSPEND Key

SUSPEND Usually forces the process to which you are typing into a **zl:break** read-eval-print loop, so that you can see what the process is doing, or stop it temporarily. The effect occurs when the character is read, not immediately. Press RESUME to continue the interrupted computation (this applies to the three modified forms of the SUSPEND key as well). While you are in the break, elements of your history in the other window remain mouse sensitive so you can yank them into the break for experimentation.

c-SUSPEND Like SUSPEND, but takes effect immediately rather than when it is read. Press RESUME to continue the interrupted computation.

m-SUSPEND Forces the process to which you type it into the Debugger when it is read. It should type out ">>BREAK:", any proceed options, and the Debugger prompt "→". You can examine the process, then press RESUME or s-A to continue.

c-m-SUSPEND

Forces the process to which you type it into the Debugger, whether or not it is running.

19.22 SYMBOL Key

Acts as a modifier key to produce special characters. Pressing sy-HELP produces a display of special function and special character keys.

19.23 TAB Key

This key is only sometimes defined. Its exact function depends on context, but in general it is used to move the cursor to an appropriate point to the right. The LINE key is related to TAB in that LINE does a RETURN followed by a TAB. If you change the behavior of the TAB key, the behavior of LINE can be affected.

19.24 Keys Not Currently Used

The following key currently has no function:

MODE LOCK

The following keys are reserved for use by the user (for example, for custom editor commands or keyboard macros):

CIRCLE
SQUARE
TRIANGLE
HYPER

20. Dictionary of Command Processor Commands

Add Paging File Command

Add Paging File *pathname :prepend*

Adds a pathname as a paging file.

pathname	The pathname of the new paging file. The default pathname is the disk unit from which you most recently booted. For example, if you most recently booted from FEP1:>, the default paging file might look like: `FEP1:>.page`
keywords	:prepend
:prepend	{*yes no*} Yes means to put the paging file at the beginning of the list of swap space to use when new space is needed. This makes the new paging file used almost immediately. No, which is the default, puts the paging file at the end of the list of paging files. Consequently, this new paging file will not be used until the previous swap space is completely used.

20.1 *Clear* Commands

Clear All Breakpoints Command

:Clear All Breakpoints *compiled-function-spec*

Clears all breakpoints in the current frame's function or in any other compiled function.

compiled-function-spec
The name of a compiled function in which you want to clear breakpoints. (Default clears all breakpoints in the current frame's function.)

Clear Breakpoint Command

:Clear Breakpoint *compiled-function pc*

Clears a breakpoint.

compiled-function	The name of a *compiled function* in which you want to clear a breakpoint.
pc	The PC (program counter) line at which you want to clear a breakpoint.

Suggested mouse operations

- To clear a breakpoint in a compiled function: Display disassembled code with the :Show Compiled Code command, point the mouse at a PC (program counter) line, and press `c-m-Mouse-Middle`.
- To clear a breakpoint in a code fragment: Display the code with the :Show Source Code command, point the mouse at a code fragment, and press `c-m-Mouse-Middle`.

Clear Output History Command

Clear Output History *window*

Discards the history for the specified window. This is useful for when you want to clean up and do an incremental garbage collection.

window	The window whose history to clear. The default is the current window.

20.2 *Compile* Commands

Compile File Command

Compile File *file-spec keywords*

Compile the file designated in *file-spec.*

file-spec	The pathname of the file to compile. The default is the usual file default.
keywords	:Compiler, :Load, :Query
:Compiler	{Lisp, use-canonical-type} The compiler to use. The default is use-canonical-type.
:Load	{yes, no, ask} Whether to load the file after compiling. The default is yes.

:Query	{yes, no, ask} Whether to ask for confirmation before compiling. The default is no.

Compile System Command

Compile System *system keywords*

Compile the files that make up *system.*

system-spec	name of the system to compile. The default is the last system loaded.
keywords	:Batch, :Condition, :Load, :New Major Version, :Query, :Redefinitions Ok :Silent, :Simulate, :Update Directory
:Batch	{yes, no} Whether to save the compiler warnings in a file instead of printing them on the console. The default is no, to just print them on the console. Adding the keyword :batch to your Compile System command is the same as :batch yes.
:Condition	{always, new-source} Under what conditions to compile each file in the system. Always means compile each file. New-source means compile a file only if it has been changed since the last compilation. The default is new-source.
:Load	{Everything, Newly-Compiled, Only-For-Dependencies, Nothing} Whether to load the system you have just compiled into the world. The default is Everything. The mentioned default is Newly-Compiled.
:New Major Version	{yes, no, ask} Whether to give your newly compiled version of the system the next higher version number. The default is yes. Giving the choice no will ask you to confirm that you really want to "prevent incrementing system major version number".
:Query	{Everything, Confirm-only, No} Whether to query before compiling. Everything means query before compiling each file. Confirm-only means create a list of all the files to be compiled and then ask for confirmation before proceeding. No means just go ahead and compile the system without asking any questions. The default is Confirm-only. The mentioned default is Everything.

:Redefinitions Ok
: {yes, no} Controls what happens if the system asks for confirmation of any redefinition warnings during the compilation. Yes means assume that all requests for confirmation are answered yes and proceed. No means pause at each redefinition and await confirmation. The default is No. The mentioned default is Yes. This allows you to start a compilation that you know will take a long time and leave it to finish by itself without interruption for questions such as "Warning: *function-name* being redefined, ok? (Y or N)".

:Silent	{yes, no} Whether to suppress output to the console. The default is no, to allow output. Adding the :silent keyword to your Compile System is the same as :silent yes.
:Simulate	{yes no} Print a simulation of what compiling would do. The default is no. Adding this keyword to your Compile System command string is the same as :simulate yes.
:Update Directory	{yes, no} Whether to update the directory of the system's components. The default is yes.

20.3 *Copy* Commands

Copy File Command

Copy File [from *file-spec*] [to *destination-spec keywords*]

Makes a copy of a file.

file-spec	The pathname of the file you want to copy.
destination-spec	The pathname of the location you want to put the file.
keywords	:Byte Size, :Copy Properties, :Create Directories, :Mode, :Query

:Byte Size	{*number*} Byte size in which to do the copy operation.
:Copy Properties	{*list of file properties*} The properties you want duplicated in the new file. The default is author and creation-date.
:Create Directories	{yes error query-each} What to do if the destination directory does not exist. The default is query-each.
:Mode	{character binary default} The mode in which to perform the transfer. The default is default.

:Query	{yes no ask} Whether to ask for confirmation before copying each file. The default is no.

Copy Microcode Command

Copy Microcode {*version or pathname*} *destination keywords*

Installs a version of microcode.

version or pathname

Microcode version number or pathname to copy. *version* is a microcode version number (in decimal). *pathname* rarely needs to be supplied. It defaults to a file on FEP*n*:> (where *n* is unit number of the boot disk) whose name is based on the microcode name and version. (The file resides in the logical directory sys:l-ucode;.) The *version* actually stands for the file *appropriate-hardware*-MIC.MIC.*version* on FEP*n*:>. (See the Section "Genera 7.0 Microcode Types" in *Software Installation Guide*)

destination	FEP file specification. The pathname on your FEP*n*:> directory. The default is created from the microcode version.
keywords	:update boot file

:update boot file

{FEP-file-spec none}. The default is the current default boot file name.

Copy Output History Into Editor Command

Copy Output History Into Editor

Creates an editor buffer and copies the history of your interaction with the Lisp Listener into it. This is useful if you want to edit your interaction for saving as a text file or for hardcopying.

Copy World Command

Copy World *file destination keywords*

Makes a copy of a world load.

file	FEP file spec. *file* is required (no default).
destination	FEP file spec. Required when copying a world from the local

	host to another host. When copying a world to the local host the default is same as the source file specification.
keywords	:Block Count, :Start Block, :Update Boot File
:Block Count	Number of blocks to copy. The default is the length of the band, meaning copy the entire band.
:Start Block	Number of the block to start with. The default is 0, meaning begin at the beginning.
:Update Boot file	{FEP-file-spec none}. The default is the current default boot file name if *destination* is the local host.

20.4 *Create* Commands

Create Directory Command

Create Directory *file-spec*

Creates a directory for storage of files on a file system specified in *file-spec*.

file-spec	The pathname of the directory to be created.

Create FEP File Command

Create FEP File *file-spec size*

Creates a file on a FEP directory on your machine.

file-spec	The pathname of the file to create. The default is FEP0:>temporary.temp.
size	The size in FEP blocks of the file. You must supply this.

Use Create FEP File to do the following:

- To create an extra paging file. For example:

```
Create FEP File fep0:>aux.page 10000
```

- To allocate space into which to load a world load. For example:

```
Create FEP File fep0:>release-6-1.load 30000
```

Create Link Command

Create Link *pathname target keywords*

Creates an association between one pathname and a second pathname, called the target. The first pathname is linked to the target so that any references to the first pathname actually refer to the target.

pathname	The pathname you want to link from. The default is the standard file default.
target	The pathname you want to link to. This pathname must exist. There is no default.
keywords	:Type
:Type	{Read-Only, Read-Write, Create-Through, All, or Use-Default} The kind of link to create. The default is Use-Default.

20.5 *Delete* Commands

Delete File Command

Delete File *file-spec keywords*

Deletes or marks for deletion the file *file-spec.*

file-spec	Pathname of the file to delete. The default is the usual file default. The version defaults to newest.
keywords	:Expunge, :Keep, :Query
:Expunge	{yes, no, ask}. Whether to expunge the file. The default is no. Adding this keyword to your Delete File command is the same as :expunge yes.
:Keep	*N* versions. The number of versions to retain. The default is 0.
:Query	{yes, no, ask} Whether to ask for confirmation before deleting the file. The default is no.

Delete Printer Request Command

Delete Printer Request *printer-request*

Deletes the specified print request from the print queue.

printer-request	A string specifying the printer and the request. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.

20.6 *Disable* Commands

Disable Network Command

Disable Network *network(s)*

Disables the network on the local machine.

Network(s)	{*network*, All} The network(s) to disable. The possibilities are the networks to which your machine is connected, that is on which it is has addresses. The default is All, meaning disable all network service on your machine.

Disable Services Command

Disable Services *service-type*

Turns off service(s) on the local machine.

service-type	{All-Services, Append, Chaos-Status, Converse, Lgp-Status, Lispm-Finger, Telnet, Time} The service to turn off. The default is All-services.

20.7 *Edit* Commands

Edit Definition Command

Edit Definition *name*

Finds the definition of the object *name* and puts it in an editor buffer for you to edit. This is the same as the Zmacs command m-.. See the section "Edit Definition" in *Text Editing and Processing*.

name	Name of the object whose definition you want to edit.

Edit Directory Command

Edit Directory *directory-spec keywords*

Invokes the directory editor **zl:dired** in Zmacs. See the section "Dired Mode in Zmacs" in *Text Editing and Processing.*

directory-spec	Pathname of the directory to edit. The default is the usual file default.
keywords	:Order, :Property, :Version
:Order	{alphabetical chronological}
:Property	File properties to display.
:Version	{all newest *number*} The default is all.

Edit File Command

Edit File *file-spec keywords*

Enters the editor and reads in *file-spec.*

file-spec	The pathname of the file to edit. The default is the usual file default.
keywords	:Initialize
:Initialize	{yes, no} Mentioned default is yes, omitted default is no.

Edit Font Command

Edit Font *font*

Invokes the Font Editor with *font* loaded to be edited.

font	A font name. There is no default. Issuing the command with no arguments invokes the font editor with no font loaded.

Edit Namespace Object Command

Edit Namespace Object *class name keywords*

Create or modify an object in the namespace database.

class	{User Printer Network Host Site Namespace any} The kind of object to create or edit. The default is any.
name	Name of the object to create or edit. The default is any.
keywords	:locally
:locally	{yes no} Whether to edit only a local copy of the information for the object. The default is no, meaning to edit the object in the central namespace database. Mentioning :locally in your command means yes, edit only a local copy.

20.8 *Enable* Commands

Enable Services Command

Enable Services *service-type*

Turns on service(s) on the local machine.

service-type	{All-Services, Append, Chaos-Status, Converse, LGP-Status, Lispm-Finger, Telnet, Time} The service to turn on. The default is all-services.

Enable Network Command

Enable Network *network(s)*

Enables (turns on) the network(s) on the local machine.

network(s)	{*network*, All} The network(s) to enable. The possibilities are any of the networks to which your machine is connected, that on which it is has addresses. The default is All, meaning enable all network service on your machine.

20.9 *Expunge* Commands

Expunge Directory Command

Expunge Directory *file-spec keywords*

Expunges files marked for deletion.

The :query option is useful for directories containing subdirectories or if you have used a wildcard in the pathname.

file-spec	The pathname of the directory to be expunged. The default is the usual file default.
keywords	:query
:query	{yes no ask} Ask for confirmation before expunging the directory. The default is no.

20.10 *Find* Commands

Find Symbol Command

Find Symbol *name keywords*

Tries to find the symbol *name.*

name	The symbol to look for.
keywords	:Package, :Type
:Package	{all, *package-name*} The package to search for the symbol. The default is the current package.
:Type	{all-types, variable, function, flavor, resource, unbound} The type of the symbol. The default is all-types.

Format File Command

Format File *pathname keywords*

Displays the contents of one or more files in formatted form, using the Document Examiner formatter.

pathname	The file or sequence of files to be formatted.
keywords	:destination :page headings
:destination	{screen *printer*} The destination for formatted output. The default is screen. The mentioned default is the default hardcopy text printer (the value of **hardcopy:*default-text-printer***) if that printer can handle formatted output; otherwise, the mentioned default is the last printer used to produce formatted

output; otherwise, there is no mentioned default. Other possible values for *printer* are Remote Printer or a local supported printer. Completion is available over the set of local supported printers. If the value is Remote Printer, the user is prompted for the name of a printer at any accessible site, with completion available over the set of supported printers in the namespaces in the namespace search list.

:page headings	{yes, no} Whether to print a heading line at the top of the page. The default is yes. The heading line consists of the word Page followed by the page number.

Format File is similar to the Format File (m-X) Zwei command. The file to be formatted must be a text file. It can contain the same formatting commands and environments as files acceptable to Format File (m-X). See the section "Zmacs Commands for Formatting Text" in *Text Editing and Processing.*

For example, to display a file in formatted form on the screen you might do the following:

```
Format File (file) acme-blue:>project>documentation.mss
```

To send the same file to a printer you might do:

```
Format File (file) acme-blue:>project>documentation.mss (keywords)
:Destination (a destination for formatted output [default Our-Printer]) Our-Printer
```

20.11 *Halt* Commands

Halt commands shut down some activity in such a way that you can resume it.

Halt GC Command

Halt GC

Turns ephemeral and dynamic garbage collection off.

Halt Machine Command

Halt Machine

Halt Machine stops execution of Lisp and gives control to the FEP. You can now enter Fep commands, for example, to warm or cold boot the machine.

Halt Printer Command

Halt Printer *printer printer-request keywords*

Halts the specified printer.

printer	The name of the printer to halt.
printer-request	(Optional) If the printer is printing a request when the Halt Printer command is given, it displays the request and asks you to confirm the halt command. If you supply a *printer-request* argument and it matches the request that is printing, the printer is halted immediately without requiring confirmation. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.
keywords	:Disposition, :Extent, :Reason, :Starting From, :Urgency
:Disposition	{Delete, Hold, Restart} What to do with the request that is printing. Delete deletes the request from the queue; you must request it again to have it printed. Hold retains the request in the queue but does not print it when the printer restarts. Restart restarts printing the interrupted request from the beginning when the printer restarts. The default is Delete.
:Reason	{*string*} The reason for the shutdown. This appears in the display from Show Printer Status to explain what is happening to users. The default is "Printer suspended by operator."

The following three keywords are related and interact to control precisely when the printer halts.

:Extent	{Entire, Copy} The extent of the request to be cancelled. Entire refers to the whole request. Copy refers to a single copy. In a request for one copy of a document, Entire and Copy are synonymous. The default is Entire.
:Starting From	{*number*} The copy number. If :Extent is Entire, this has no meaning. If :Extent is Copy, this is the number of the copy after which to halt the printer. The default is 0.
:Urgency	{Asap, After-Extent} When to halt. Asap means instantly, ignoring any settings of :Extent and :Starting From. After-Extent means halt based on the settings of the :Extent and :Starting From keywords. The default is Asap.

20.12 *Hardcopy* Commands

Hardcopy File Command

Hardcopy File *file-spec printer keywords*

Sends a file to a hardcopy device.

file-spec	The pathname of the file to be printed. The default is the usual file default.
printer	The printer to use to output the file. The default is determined from your init file or from the default-printer attribute for the host in the namespace database.
keywords	:Body Character Style :Copies :Delete :Ending Page :File Types :Heading Character Style :Orientation :Running Head :Starting Page

:Body Character Style	The character style to use for printing the text of the file and against which to merge any character styles in the file. The default is the *null style*, **(nil nil nil)**, meaning use the default for the printer.
:Copies	{*number*} The number of copies to print. The default is 1.
:Delete	{yes, no} Whether to delete the file after it is printed. The default is no, not to delete. Adding the :delete keyword to your hardcopy command string is the same as :delete yes.
:Ending Page	{*number*} The last physical page to print. The default is the last page of the file. A page is defined by the presence of a PAGE character or form feed in the file. Thus plain text files with no page markers in them are treated as a single page, although they take up several sheets of paper. Press format files, on the other hand, have form feeds or PAGE characters in them. It is important to remember that these are physical pages and do not necessarily correspond to the page numbering appearing in the heading. For example, the first physical page of a press file is probably a title page and the second physical page might be numbered *i* so the page numbered 1 might be the third physical page.
:File Types	{Text, Suds-Plot, Press, Lgp, Lgp2, Dmpl, Xgp or use-canonical-type} The internal format of the contents of the file, to

interpret for printing. The default is use-canonical-type, meaning that the type is determined from the extension to the file name.

:Heading Character Style	The character style to use for the running head supplied by the :running head keyword.
:Orientation	{landscape, portrait} Orientation on the paper for the output. Portrait is left to right across the short dimension of the paper. Landscape is left to right across the long dimension. The default is portrait.
:Running Head	{none, numbered} Type of running head to print on the top of each page. The default is numbered.
:Starting Page	{*number*} The first physical page to print. The default is the first page of the file. A page is defined by the presence of a PAGE character or form feed in the file. Thus plain text files with no page markers in them are treated as a single page, although they take up several sheets of paper. Press format files, on the other hand, have form feeds or PAGE characters in them. It is important to remember that these are physical pages and do not necessarily correspond to the page numbering appearing in the heading. For example, the first physical page of a press file is probably a cover page and the second physical page might be numbered *i* so the page numbered 1 might be the third physical page.
:Title	{*string*} Title to appear on the cover page to identify the output. The default is your user name.

20.13 *Help* Commands

Help Command

Help *keywords*

Displays a list of command processor commands.

keywords	:format
:format	{brief, detailed} The level of detail to show in the list. Brief means that only unique command names are shown in full and for the rest, the verb is shown with a number indicating the

number of commands that start with that verb. Detailed means the entire list of available commands is displayed. The default is brief. This is the same as pressing the HELP key.

20.14 *Initialize* Commands

Initialize Mail Command

Initialize Mail

Reloads Zmail without disturbing the rest of the system. The state of your mail on the machine is lost, so you only want to use this when your Zmail process is irreparably stuck.

Initialize Mouse Command

Initialize Mouse

Restarts the mouse process.

Initialize Time Command

Initialize Time *time keywords*

Resets the time. You can use this function to correct the time if it appears to be inaccurate.

time	A string representing date and time. The default is obtained from the network or from the calendar clock if there is no network available. If the calendar clock is also being initialized, you are prompted to enter the correct time from the keyboard.
keywords	:Clock
:Clock	{status-line, both, ask} The clock to update. Status-line is the little clock in the lower left hand corner of the screen. Both means update both the calendar clock and the status line clock. Ask is the default. It asks you which time source to use and updates the status line clock and asks if you also want to update the calendar clock (unless that is the time source you are using).

Initialize Time accepts most reasonable printed representations of date and time and converts them to the standard internal forms. The following are representative formats that are accepted by the parser:

```
"March 15, 1960" "15 March 1960" "3/15/60" "3/15/1960"
"3-15-60" "3-15" "15-March-60" "15-Mar-60" "March-15-60"
"1960-3-15" "1960-March-15" "1960-Mar-15"
"1130." "11:30" "11:30:17" "11:30 pm" "11:30 am" "1130" "113000"
"11.30" "11.30.00" "11.3" "11 pm" "12 noon"
"midnight" "m" "Friday, March 15, 1980" "6:00 gmt" "3:00 pdt"
"15 March 60" "15 March 60 seconds"
"fifteen March 60" "the fifteenth of March, 1960;"
"one minute after March 3, 1960"
"two days after March 3, 1960"
"Three minutes after 23:59:59 Dec 31, 1959"
"now" "today" "yesterday" "two days after tomorrow"
"one day before yesterday" "the day after tomorrow"
"five days ago"
```

Date strings in ISO standard format are accepted also. These strings are of the form `"yyyy-mm-dd"`, where:

`yyyy`	Four digits representing the year
`mm`	The name of the month, an abbreviation for the month, or one or two digits representing the month
`dd`	One or two digits representing the day

Following are some restrictions on strings to be parsed:

- You cannot represent any year before 1900.
- A four-digit number alone is interpreted as a time of day, not a year. For example, `"1954"` is the same as `"19:54:00"` or `"7:54 pm"`, not the year 1954.
- The parser does not recognize dates in European format. For example, `"3/4/85"` or `"3-4-85"` is always the same as `"March 4, 1985"`, never `"April 3, 1985"`. A string like `"15/3/85"` is an error. In such strings, the first integer is always parsed as the month and the second integer as the day.

20.15 *Inspect* Commands

Inspect Command

Inspect *object*

Displays the components of *object.* This is similar to Show Object but it uses the Inspector, a window oriented program for showing data structures. It allows you to do something to that object, such as inspect it, modify it, or give it as the argument to a function. You exit from the Inspector by clicking the mouse on `EXIT` in the Inspector menu.

object	Any Lisp object. The default is "unspecified".

See the section "The Inspector" in *Program Development Utilities.*

20.16 *Load* Commands

Load File Command

Load File *file-spec keywords*

Loads files into the current world.

file-spec	The pathname of the file to load. More than one pathname may be specified, separated by commas. The default is the usual file default.
keywords	:package, :query, :silently
:package	*package-name* The package into which to load the file. The default is the file's "home" package.
:query	{yes no ask} Whether to ask for confirmation before loading each file. The default is no.
:silently	{yes no} Whether to print a line as each file is loaded.

Load Patches Command

Load Patches *system keywords*

Loads patches into the current world for the indicated systems or for all systems. See the function **load-patches** in *Program Development Utilities.*

system	{All *system-name1, system-name2* ... } The default is All.
keywords	:Query, :Save, :Show
:Query	{yes no ask} Whether to ask for confirmation before loading each patch. The default is no.

:Save	{*file-spec*, Prompt, No-Save} The file in which to save the world with all patches loaded. Omitting this keyword means do not save the world. The default when this keyword is added to your command is Prompt which means save the world and then prompt for a pathname.
:Show	{yes no ask} Whether to print the patch comments as each patch is loaded. The default is yes.

Load System Command

Load System *system keywords*

Loads a system into the current world.

system	Name of the system to load. The default is the last system loaded.
keywords	:Condition :Load Patches :Query :Redefinitions Ok :Silent :Simulate :Version
:Condition	{always, never, newly-compiled} Under what conditions to load each file in the system. Always means load each file. Newly-compiled means load a file only if it has been compiled since the last load. The default is always.
:Load Patches	{yes, no} Whether to load patches after loading the system. The default is yes.
:Query	{Everything, Confirm-only, No} Whether to query before loading. Everything means query before loading each file. Confirm-only means create a list of all the files to be loaded and then ask for confirmation before proceeding. No means just go ahead and load the system without asking any questions. The default is Confirm-only. The mentioned default is Everything.
:Redefinitions Ok	{yes, no} Controls what happens if the system asks for confirmation of any redefinition warnings during the loading process. Yes means assume that all requests for confirmation are answered yes and proceed. No means pause at each redefinition and await confirmation. The default is No. The mentioned default is Yes. This allows you to start loading a system that you know will take a long time to load and leave it to finish by itself without interruption for questions such as "Warning: *function-name* being redefined, ok? (Y or N)".

:Silent	{yes, no} Whether to turn off output to the console while the load is going on. The default is no. Adding this keyword to your Load System command string is the same as :silent yes.
:Simulate	{yes, no} Print a simulation of what compiling and loading would do. The default is no. Adding this keyword to your Load System command string is the same as :simulate yes.
:Version	{released, latest, newest, use-default, *version-number*, *version-name*} Which version number to load. The default is use-default, that is latest.

Note: This command only loads a system. If you want to compile and load a system: See the section "Compile System Command", page 229.

20.17 *Login* and *Logout* Commands

Login Command

Login *user keywords*

Logs the *user* into Genera.

user	Any string. Your user ID.
keywords	:host :init file
:host	{local *any-host-name*} A particular host computer. `Local` as an argument to :host is particularly useful if your namespace system is down and you wish to log in to your Lisp Machine without having it try to use the namespace database. The default comes from `login` host for your user object in the namespace database.
:init file	{default-init-file none} Whether to load your init file. The default is to load your default init file. To avoid loading your init file, use :init file none.

If someone is already logged in when you give the Login command, that user is logged out. If this happens, you see the message

```
Warning -- You are logging out from program-name
```

Logout Command

Logout *keywords*

Logs you out of Genera.

keywords	:Save Buffers, :Save Mail
:Save Buffers	{yes, no, ask} Whether to write out modified editor buffers to disk. The default is yes.
:Save Mail	{yes, no, ask} Whether to write out modified mail buffers to disk. The default is yes.

Monitor Variable Command

:Monitor Variable *symbol keywords*

Monitors the access of a special variable. This command arranges for a trap to be signalled when any process accesses the monitored location. This command is used to answer the question: "What program or process is reading or writing this location in memory?". This is particularly useful when there are several processes sharing some data structures, and you want to debug the interactions between the processes.

symbol	The name of a symbol whose location in memory you want to monitor. Enter the name of a symbol and, optionally, its Value-Cell or Function-Cell. (See the :Cell keyword description below.)
keywords	:Boundp, :Cell, :Locf, :Makunbound, :Read, :Write
:Boundp	{Yes, No} Monitors the location for **boundp** operations. (Default is No.)
:Cell	{Value-Cell, Function-Cell} Specifies the cell that you want to monitor within the *location.* The Debugger gives you two choices: Value-Cell or Function-Cell. (Default is Value-Cell.)
:Locf	{Yes, No} Monitors the location for **locf** operations. (Default is No.)
:Makunbound	{Yes, No} Monitors the location for **makunbound** operations. (Default is No.)
:Read	{Yes, No} Monitors the location for reads. (Default is No.)
:Write	{Yes, No} Monitors the location for writes. (Default is Yes.)

Suggested mouse operations

- To monitor a location: Point the mouse at a locative, structure slot, or instance variable and press `c-m-sh-Mouse-Left`.
- To unmonitor a location: Point the mouse at a locative, structure slot, or instance variable that was previously monitored and press `c-m-sh-Mouse-Middle`.

Optimize World Command

Optimize World *keywords*

Optimizes the world that is currently loaded into your environment. Use this command if you load special programs or systems in addition to the distribution by reorganizing the world to improve paging performance.

keywords
:show

:show	Displays the progress of the optimization process on the screen.

20.18 *Rename* Commands

Rename File Command

Rename File *from-file to-file keyword*

from-file	The pathname of the file to be renamed. The default is the usual file default.
to-file	The new pathname. The default is the usual file default.
keyword	:Query
:Query	{yes, no, ask} Whether to ask for confirmation before renaming. The default is no.

Report Bug Command

Report Bug *system name*

Sends a bug report. It starts up a bug mail window with the message header `To: BUG-`*system name*. If *system name* is omitted, `BUG-LISPM` is used.

20.19 *Reset* Commands

Reset Network Command

Reset Network

Turns your network interfaces off and back on again, and also resets the namespace system. Turning the interfaces off and on can be useful if your network connections appear to be stuck and nothing is being transmitted or received. Resetting the namespace system can cure problems caused by information related to the namespace having been corrupted on your local machine. One symptom of that is your host repeatedly trying to connect to another host, without success.

The Reset Network command is the same as entering **(neti:general-network-reset)** followed by **(neti:enable)**.

Reset Printer Command

Reset Printer *printer printer-request keywords*

Resets a printer.

printer	The printer to reset.
printer-request	(Optional) If the printer is printing a request when the Reset Printer command is given, it displays the request and asks you to confirm the reset command. If you supply a *printer-request* argument and it matches the request that is printing, the printer is reset immediately without requiring confirmation. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.
keywords	:Disposition
:Disposition	{Delete, Hold, Restart} What to do with the request that is printing. Delete deletes the request from the queue; you must request it again to have it printed. Hold retains the request in the queue but does not print it when the printer restarts. Restart restarts printing the interrupted request from the beginning when the printer restarts. The default is Delete.

Resetting is like turning the printer off and then on again except that it is done remotely, you do not have to go over to the printer.

20.20 *Restart* Commands

Restart Printer Request Command

Restart Printer Request *printer-request*

Restarts a print request that has not yet finished. This is useful if something goes wrong with the printing, for example the paper is coming out crumpled.

printer-request	A string specifying the printer and the request. The print request should be selected with the mouse from the display of the Show Printer Status command. See the section "Show Printer Status Command", page 153.

20.21 *Save* Commands

Save File Buffers Command

Save File Buffers *keywords*

Saves your modified Zmacs file buffers on disk.

keywords	:Query
:Query	{yes, no, ask} The default is yes, meaning that it asks you about each buffer before writing it out.

Save Mail Buffers Command

Save Mail Buffers *keywords*

Saves on disk any modified mail buffers.

keywords	:expunge, :query
:expunge	{yes no ask} Whether to expunge each buffer before saving. The default is ask.
:query	{yes no ask} The default is yes, meaning that it asks you about each buffer before writing it out.

Save World Command

Save World (Complete or Incremental) *file-spec*

Saves the current world. The system prompts (Complete or Incremental) if Incremental Disk Save is enabled. You specify Complete to do a save of the entire world, or Incremental to do an Incremental Disk Save. If Incremental Disk Save is not enabled, the prompt is (Complete). You enter Complete by pressing `SPACE` or by typing `complete`.

file-spec	The pathname to use for the saved world. The default is the FEP file specification for the local machine, based on the version number of the current system and on whether the save is to be complete or incremental.

20.22 *Select* Commands

Select Activity Command

Select Activity *activity*

Selects *activity* and makes it current.

Activity can be:

Accept Values	Lisp
Converse	Mail
Document Examiner	Menu Program
Editor	Notifications
FEP-Tape	Peek
File Server	Presentation Inspector
File system operations	Terminal
Flavor Examiner	Zmacs
Frame-Up	Zmail
Inspector	

20.23 *Send* Commands

Send Mail Command

Send Mail *recipient keywords*

Prompts for the text of a message and sends it as electronic mail to *recipient.*

recipient	One or more electronic mail addresses of the form *user* or *user@host.* Multiple addresses should be separated by commas. If you supply only *user*, the namespace for your site is searched to locate the proper host for that user's mail.
keywords	:Cc, :Subject
:Cc	{*user*, *user@host*} One or more addresses to send a *Cc*, carbon copy, of the message.
:Subject	{*string*} A line to serve as the subject for the message.

Send Message Command

Send Message *recipient*

Sends a message to the specified recipient.

recipient	*user* or *user@host.* The person to whom to send the message. If *@host* is omitted, all Lisp machines on your network are polled to locate *user.*

Send Message prompts for text to send as a Converse message. END terminates and sends the message. This is equivalent to **zl:qsend**. See the section "Converse" in *Communicating with Other Users.*

20.24 *Set* Commands

Set commands set status variables.

Set Base Command

Set Base *number*

Sets the input and output bases to *number.*

The value of **zl:base** is a number that is the radix in which integers and ratios are printed in, or a symbol with a **si:princ-function** property. The initial value of **zl:base** is 10. **zl:base** should not be greater than 36 or less than 2.

The printing of trailing decimal points for integers in base ten is controlled by the value of variable ***print-radix***. See the section "Printed Representation of Rational Numbers" in *Symbolics Common Lisp.*

The following variable is a synonym for **zl:base**:

print-base

number	A number in base 10. The default is 10.

Set Breakpoint Command

:Set Breakpoint *compiled-function pc*

Sets a breakpoint.

compiled-function	The name of a *compiled-function* in which you want to set a breakpoint.
pc	The PC (program counter) line at which you want to set a breakpoint.
keywords	:Action, :Conditional
:Action	{Show-All, Show-Args, Show-Locals, *expression*} Specifies an action to take when the breakpoint is encountered. Show-All: Displays arguments and local variables. Show-Args: Displays arguments and no local variables. Show-Locals: Displays only local variables. Give an *expression* if you want it to be evaluated in the lexical context of the frame. (Default is no action. Mentioned default is Show-All.)
:Conditional	{Always, Mode-Lock, Never, Once, *expression*} Executes the breakpoint trap according to certain conditions. Always: The breakpoint is always taken. Mode-Lock: The breakpoint is taken only when the `MODE LOCK` key is depressed. Never: The breakpoint is never taken. Once: The breakpoint is taken only for the first time it is encountered. Give an *expression* if you want it to be evaluated in the lexical context of the frame. (Default is Always.)

Suggested mouse operations

- To set a breakpoint in a compiled function: Display disassembled code with the :Show Compiled Code command, point the mouse at a PC line, and press `c-m-Mouse-Left`.

- To set a breakpoint in a code fragment: Display the code with the :Show Source Code command, point the mouse at a code fragment, and press `c-m-Mouse-Left`.

Set Calendar Clock Command

Set Calendar Clock *time*

time	A date and time string.

Machines in the 3600 family have a calendar clock that operates independently of the other software timers. When you cold boot and the machine fails to get the time from the network, it asks you to type in the time. If the calendar clock has been set, it uses that time reading as the default. If the calendar clock has not been set, it offers to set it to the time you enter from the keyboard.

Set Calendar Clock allows you to set this clock if you need to. If you do not specify *time*, it prompts you to enter it from the keyboard or press `S` to use the time in the status line. Set Calendar Clock accepts the standard formats for date and time. See the section "Set Time Command", page 261.

Set Command Processor Command

Set Command Processor *mode prompt-string*

Sets the mode for the command processor.

mode

form-only	Anything typed is taken as a Lisp form to be evaluated.
form-preferred	Anything typed is taken as Lisp forms unless it is preceded by a colon (:), in which case it is taken as a command processor command.
command-preferred	Anything typed is taken as a command processor command unless it begins with a left parenthesis or, in the case of a variable to be evaluated, a comma (,) or any nonalphabetic character.
command-only.	Anything typed is taken as a command processor command.

The default is command-preferred.

prompt-string Any string. The default for command-preferred and command-only modes is `Command:` . The default for form-preferred and form-only modes is " ". To include a space in your prompt, you must enclose the string in double quotes. For example:

```
                    Set Command Processor Command-Preferred "Command: "
```

Setting the prompt for a mode makes the prompt you set the default for that mode. You can set the prompt for all modes in your init file by calling si:cp-on

Set File Properties Command

Set File Properties *pathname keywords*

Change the properties of a file.

pathname	The file whose properties you are modifying.
keywords	:Author, :Creation Date, :Delete, :Generation Retention Count :Modification Date, :Reap, :Reason
:Author	Who created this file.
:Creation Date	When this file was created.
:Delete	Whether or not this file can be deleted explicitly with the Delete File command.
:Generation Retention Count	How many versions of this file to keep unmarked for deletion. Extra copies of the file are marked for deletion.
:Modification Date	When this file was last modified.
:Reap	Whether or not this file can be reaped. Reaping is deletion under program control.
:Reason	Why this file cannot be deleted.

Set GC Options Command

Set GC Options

Select options for garbage collection from a menu. The menu looks like this:

```
:Set GC Options
Garbage collector status:
  Ephemeral GC: Off  On
Garbage collector report controls:
  Report the activity of the ephemeral GC: Yes  No
  Report the activity of the dynamic GC: Yes  No
  Report space reclaimed individually for each area:
        Yes  No  Dynamic only  Ephemeral only
  Enable warnings, for example that the GC should be turned on: Yes  No
  Minimum interval between repetitions of a GC warning: Forever
  Minimum adequate level of free space to suppress GC warnings: 1000000
  Minimum free space to suppress warnings when ephemeral GC is on: 200000
  Ratio of free space sizes on successive GC warnings: 0.75
  Disposition of GC reports: notification


Garbage collection options:
  Normal garbage collection mode: Incremental  Immediate
  Mode when free space is low: Turn GC off  Immediate
  Mode when collecting ephemeral objects: Incremental  Immediate
  Fraction of free space committed to the garbage collector: 1
  Minimum fraction of free space GC will accept: Same

Garbage collector process controls:
  Process priority for foreground operations: 5
  Process priority for background operations: 0
  Process priority for immediate garbage collection mode: 5
  Priority of GC daemon processes: 5
  Delay before warning that flipping is inhibited: 30 seconds
  Time that processes wait before inhibiting flips: 5 seconds

Scavenger performance options:
  Amount of "scavenger work" done with interrupts inhibited: 1024
  Maximum time (microseconds) with interrupts inhibited: 50000
  Number of words scavenged before turning to another region: 128
  Number of pages of Copy space to prefetch: 9
  Number of pages that refer to ephemeral objects to prefetch: 15
  Number of pages of Old space to prefetch: 5
  Number of words to look ahead for prefetchable
        Old space references: 2048
  Declare pages flushable from main memory after scavenging them: Yes  No

<ABORT> aborts, <END> uses these values
```

Set Input Base Command

Set Input Base *new-base*

Sets the input bases to *new-base.*

The value of **zl:base** is a number that is the radix in which integers and ratios are printed in, or a symbol with a **si:princ-function** property. The initial value of **zl:base** is 10. **zl:base** should not be greater than 36 or less than 2.

The printing of trailing decimal points for integers in base ten is controlled by the value of variable ***print-radix***. See the section "Printed Representation of Rational Numbers" in *Symbolics Common Lisp.*

The following variable is a synonym for **zl:base**:

   print-base

new-base	A number in base 10. The default is 10.

Since the Input Base is closely linked to the Output Base, if you set one of them, you should set the other to the same value.

Set Lisp Context Command

Set Lisp Context *lisp-syntax*

lisp-syntax	Zetalisp or Common-Lisp.

Sets the current context to use Zetalisp or Common Lisp syntax and sets the current package **zl-user** (for Zetalisp) or **cl-user** (for Common Lisp). The default is the current lisp syntax, so using the command with no arguments does not change anything.

Set Output Base Command

Set Output Base *new-base*

Sets the output bases to *new-base.*

The value of **zl:base** is a number that is the radix in which integers and ratios are printed in, or a symbol with a **si:princ-function** property. The initial value of **zl:base** is 10. **zl:base** should not be greater than 36 or less than 2.

The printing of trailing decimal points for integers in base ten is controlled by the value of variable ***print-radix***. See the section "Printed Representation of Rational Numbers" in *Symbolics Common Lisp.*

The following variable is a synonym for **zl:base**:

print-base

new-base	A number in base 10. The default is 10.

Since the Output Base is closely linked to the Input Base, if you set one of them, you should be sure to set the other to the same value.

Set Package Command

Set Package *package-name*

Makes *package-name* the current package; in other words, the variable **zl:package** is set to the package named by *package-name.*

package	The name of a package.

Set Printer Command

Set Printer *printer-name keywords*

Sets the default printer for hardcopy.

printer-name	The name of a supported printer that can be reached by your machine.
keywords	:Output Type
:Output Type	{text bitmap both} The type of output to send to that printer. Text means files and mail messages, bitmap means graphics and screen hardcopy. The default is both, meaning use the same printer for both types of output.

Set Screen Options Command

Set Screen Options

Allows you to customize your screen by selecting parameters from a menu containing all the screen options. You select options with the mouse.

The menu looks like this:

```
Documentation line options:
  Mouse documentation area height: 2
  Mouse documentation area background: [✓ Black] White
  Mouse documentation area character style: SANS-SERIF.ROMAN.NORMAL
  Interval between keyboard input and mouse motion after
         which to blank documentation line: None
  Interval before mouse documentation line goes blank after
         exiting input context: None


Status line options:
  Status line time display:
        [✓ [Mon 31 Jan 11:59:59]] 12/31/89 23:59:59  Mon 31 Jan 11:59pm
  Clock colon blink half period: 1 second
  Process display: [✓ User name] Process name
  Machine name: Visible [✓ Invisible]
  Progress area character style: FIX.EXTRA-CONDENSED.NORMAL
  Progress area: Wide bar  No display  [✓ Text and thin bar]

Screen options:
  Background gray pattern:
   None  5.5%  6%  7%  8%  9%  10%  12%  HES  25%  33%  50%  75%  Black  [✓ White]
  Partially exposed window gray pattern:
   None  5.5%  [✓ 6%]  7%  8%  9%  10%  12%  HES  25%  33%  50%  75%  Black  White
  Interval to wait before dimming screen: 20 minutes
  Dimness percentage: 0
  Beep mode: Beep only  Flash only  [✓ Beep and flash]
  Screen background: Black  [✓ White]

Global window defaults:
  End of screen action: [✓ Scroll]  Truncate  Wrap
  Amount to scroll (number of lines or screen fraction): 1
  Overlap between screens when scrolling (lines): 1
  Character style for prompts: NIL.NIL.NIL
  Highlighting mode for highlighting menus: Inverse video box
ABORT aborts, END uses these values
```

20.24.1 Set Site Command

Set Site *site name*

Starts a dialogue to set the current site to be *site name*. This command is used to configure the software and identify your machine before you use a new world load.

It should be the first thing you type to your machine after booting the new software.

When a new world is booted for the first time, the herald gives the machine name as *DIS-LOCAL-HOST*. You are prompted in the course of the Set Site dialogue for a name for the machine.

You need the following information to use the Set Site command:

Site name
: What you call the location of your machines. This might be the name of your company, or, if you are more whimsical-minded, it might be related to the machine names you have chosen. In the sample dialogues, we have chosen the site name Downunder.

Name of the local host
: The name of the Symbolics computer you are configuring. See the section "Why Do You Name Machines and Printers?", page 140. In the sample dialogues, we have chosen machine names Koala and Kangaroo.

Name of the namespace server
: The name of the machine where the namespace database is stored.

Chaosnet address of the namespace server
: The octal number that identifies the location of the namespace server on the network. You can use Show Host *machine name* or **(zl:hostat** "*machine name*"**)** to find this number.

If you are configuring a new site, you also need:

SYS host
: The machine where the sources are to be stored.

Host for bug reports
: The machine to which bug reports are to be sent.

SYS:SITE; directory translation
: The physical pathname that sys:site; translates to on the sys host. See the section "What is a Logical Pathname?", page 139. In the sample dialogues, this is

```
koala:>sys>site>
```

System account
: The user-id that the system uses when a server logs into a machine. In the sample dialogues, we have chosen Wombat.

Set Stack Size Command

:Set Stack Size *stack-type stack-size*

Sets the size of a stack.

stack-type	The type of the stack. Enter Control, Binding, or Data. (Default is Control.)
stack-size	The size of the stack. Enter a number of machine words that represents the stack size.

Set Time Command

Set Time *time*

Sets the local time on your machine. This allows you to set the time that appears in the lower left hand corner of the status line if you need to. If you do not specify a time, you are prompted to enter the time from the keyboard. The default is constructed from the status line time.

time	A date and time string.

Set Time accepts most reasonable printed representations of date and time and converts them to the standard internal forms. The following are representative formats that are accepted by the parser:

```
"March 15, 1960" "15 March 1960" "3/15/60" "3/15/1960"
"3-15-60" "3-15" "15-March-60" "15-Mar-60" "March-15-60"
"1960-3-15" "1960-March-15" "1960-Mar-15"
"1130." "11:30" "11:30:17" "11:30 pm" "11:30 am" "1130" "113000"
"11.30" "11.30.00" "11.3" "11 pm" "12 noon"
"midnight" "m" "Friday, March 15, 1980" "6:00 gmt" "3:00 pdt"
"15 March 60" "15 March 60 seconds"
"fifteen March 60" "the fifteenth of March, 1960;"
"one minute after March 3, 1960"
"two days after March 3, 1960"
"Three minutes after 23:59:59 Dec 31, 1959"
"now" "today" "yesterday" "two days after tomorrow"
"one day before yesterday" "the day after tomorrow"
"five days ago"
```

Date strings in ISO standard format are accepted also. These strings are of the form "yyyy-mm-dd", where:

yyyy	Four digits representing the year

`mm`	The name of the month, an abbreviation for the month, or one or two digits representing the month
`dd`	One or two digits representing the day

Following are some restrictions on strings to be parsed:

- You cannot represent any year before 1900.
- A four-digit number alone is interpreted as a time of day, not a year. For example, `"1954"` is the same as `"19:54:00"` or `"7:54 pm"`, not the year 1954.
- The parser does not recognize dates in European format. For example, `"3/4/85"` or `"3-4-85"` is always the same as `"March 4, 1985"`, never `"April 3, 1985"`. A string like `"15/3/85"` is an error. In such strings, the first integer is always parsed as the month and the second integer as the day.

Set Window Options Command

Set Screen Options

Displays a menu of the settable options for the current window. See the section "Using Menus", page 29.

```
More processing enabled: [✓ Yes]  No
Reverse video: Yes  No
Vertical spacing: 2
Deexposed typein action: Wait until exposed  Notify user
Deexposed typeout action: Wait until exposed  Notify user  Let it happen  Signal error  Other
ALU function for drawing: Ones  Zeroes  Complement
ALU function for erasing: Ones  Zeroes  Complement
Screen manager priority: None
Save bits: Yes  No
Default character style: FIX.ROMAN.NORMAL
Echo character style: NIL.NIL.NIL
Typein character style: NIL.NIL.NIL
End of screen action: Default  Scroll  Truncate  Wrap
Amount to scroll by: Default
<Abort> aborts, <End> uses these values
```

20.25 *Show* Commands

Show commands allow you to request informational displays of all kinds. These displays are mouse sensitive when appropriate and can be used in composing other commands. For example, after a Show Directory display, the individual pathnames in the directory can be selected as arguments to a Show File command.

Show Command Processor Status Command

Show Command Processor Status

Displays the current mode of the Command Processor and the current prompt. It also displays the prompts for the other modes. For example:

```
The command processor's current mode is
 Form Preferred: Interprets input starting with an alphabetic
                 character as Lisp;
                 type an initial : to force command interpretation.

The prompt string is "->".

The prompt strings for other modes are:
  Command Preferred:        "Command: "
  Form Only:                ""
  Command Only:             "Command: "
```

Show Directory Command

Show Directory *pathname keywords*

Displays a directory listing. The default for name, type, and version of *pathname* is **:wild**. The format of the listing varies with the operating system.

pathname	The pathname of the directory to list. The default is the usual file default.
keywords	:Before, :Order, :Output, :Since, :Size
:Before	a date. Show only those files created prior to this date.
:Order	{oldest-first smallest-first largest-first newest-first standard} Show files in this order. The default is standard, which is usually alphabetical.
:Output Destination	{*buffer file stream window*} Where to display the information. The default is the current window.

:Since	a date. Show only those files created after this date.
:Size	Show only those files the same size or larger than *N* blocks. The default is 0, meaning that all files will be listed, even if they are empty.

Show Disabled Services Command

Show Disabled Services

Shows you which services are disabled (with the Disable Services Command).

Show Documentation Command

Show Documentation *topic keywords*

Displays the documentation for *topic.* If you omit *topic,* you are prompted for it. If *topic* is more than one word, it must be enclosed in double-quotes:

```
    Show Documentation (for topic) "The Document Examiner"
```

keywords	:destination
:destination	{screen, "remote printer", or *printer name*}. Where to display (print) the documentation. Entering "remote printer" prompts for the name of a printer. You can also give the name of a local supported printer explicitly. Names containing spaces must be enclosed in double-quotes. The default is to display the documentation on the screen. Mentioning the :destination keyword changes the default to the default hardcopy device, if it is a supported printer, or to the supported printer most recently used for formatted output.

See the section "Using the Online Documentation System", page 43.

Show Expanded Lisp Code Command

Show Expanded Lisp Code *form keywords*

Shows the lisp definition of *form.*

form	{*macro name* None} The Lisp form whose code to expand. Specifying None causes the command to prompt you for a macro form to evaluate just as if you had used **(mexp)**. See the function **mexp** in *Symbolics Common Lisp.*
keywords	:As If Compiler :Constant Folding :Expand :Output Destination :Repeat :Whole Form

:As If Compiler	{yes no} Does everything the compiler would do to the form. The default is no. The mentioned default is yes.
:Constant Folding	{yes no} Does any constant computations, arithmetic for instance, that will not change and thus can be done now. The default is no. The mentioned default is yes.
:Expand	{macros inline-functions style-checkers optimizers constants} Does macro expansion. The default is to expand inline-functions and macros.
:Repeat	{yes no} Takes the output and operates on it again to expand it fully. The default is no. The mentioned default is yes.
:Whole Form	{yes no} Expands recursively until the innermost code loop is expanded. The default is yes.

Example:

```
Show Expanded Lisp Code (defmacro form ()())

(PROGN (SI:DEFMACRO-CLEAR-INDENTATION-FOR-ZWEI 'FORM)
       (MACRO FORM (SI:.FORM. SI:.ENV.)
         (DECLARE (ARGLIST))
         (BLOCK FORM
           (TAGBODY
                   (LET* NIL
                     (AND (CDR SI:.FORM.)
                          (GO #:G1227))
                     (IGNORE SI:.ENV.)
                     (RETURN-FROM FORM
                       (PROGN NIL)))
           #:G1227 (RETURN-FROM FORM
                    (SI:DEFMACRO-ARGUMENT-ERROR 'FORM 'NIL SI:.FORM.))))))
```

Show Expanded Mailing List Command

Show Expanded Mailing List *electronic-mailing-list keywords*

Shows the list of individuals who receive messages sent to a mailing list.

electronic-mailing-list	The name of a mailing list (from the file `>mail>static>mailboxes.text`)
keywords	:All Levels :Matching :Output Destination

:All Levels	{yes no} Whether to show the expansion of mailing lists that are included in the selected mailing list. The default is yes, expand all lists down to individuals.
:Matching	{*string*} Shows only those names on the mailing list matching *string*.
:Output Destination	{*buffer file stream window*} Where to display the information.

Show FEP Directory Command

Show FEP Directory *host unit keywords*

Displays a description of the FEP files on *unit*.

host	A host on the network. The default is local.
unit	{*disk-unit-number* All} The default is All. *disk-unit-number* is an integer, interpreted as a disk unit number on the specified host.

Show FEP Directory first displays an estimate of the number of free blocks and the proportion of blocks used on *unit*. It then displays a summary of the files on *unit*, one line per file. For each file, it displays the file name, the length in blocks, the creation date, the comment, and the author.

Show File Command

Show File *file-spec*

Displays a file on the screen. If there is more than one screenful, it pauses between screenfuls displaying `--More--` at the bottom.

file-spec	The pathname of the file to be printed. The default is the usual file default.

20.25.1 Show Flavor Commands

The following commands show attributes of a flavor, generic function, method, or handler. Only those keywords that are specific to each command (or have a different meaning for each command) are explained in the command descriptions. For an explanation of any keywords not covered in the command descriptions: See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp*.

Show Flavor Components Command

Show Flavor Components *flavor keywords*

Shows the order of the components of this flavor.

keywords :brief, :detailed, :duplicates, :functions, :initializations, :instance variables, :match, :methods, and :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*

:duplicates	{yes, no} Indicates whether or not to display duplicate occurrences of flavors. The default is no.
:brief	{yes, no} yes indicates that the output should not be indented to show the structure. The default is no.

The flavor components are ordered from top to bottom. The top flavor is the *most specific flavor* in the ordering. The indentation graphically represents which flavors are components of which other flavors. In the example below, **tv:minimum-window** has six direct components: **tv:essential-expose, tv:essential-activate, tv:essential-set-edges, tv:essential-mouse, tv:essential-window**, and **flavor:vanilla.**

When you use the :duplicates keyword and show the components of complex flavors, you notice special symbols in the display. For example:

```
Command: Show Flavor Components TV:MINIMUM-WINDOW :Duplicates
    --> TV:MINIMUM-WINDOW
          TV:ESSENTIAL-EXPOSE
            [TV:ESSENTIAL-WINDOW] ↓
          TV:ESSENTIAL-ACTIVATE
            [TV:ESSENTIAL-WINDOW] ↓
          TV:ESSENTIAL-SET-EDGES
            [TV:ESSENTIAL-WINDOW] ↓
          TV:ESSENTIAL-MOUSE
          TV:ESSENTIAL-WINDOW
            TV:SHEET
              SI:OUTPUT-STREAM
                SI:STREAM
          FLAVOR:VANILLA
```

Bracketed flavors are duplicates that are included by the parent flavor here, but are not ordered in this position because of some ordering constraint. They appear in another place in the display without brackets, in their correct order. All bracketed components have an arrow beside them. A down-arrow indicates that this component's position in the ordering is later in the display. An up-arrow indicates that this component's position in the ordering is earlier in the display; these occurrences are infrequent.

For example, the flavor **tv:essential-window** is a component of four other components: **tv:essential-expose, tv:essential-activate, tv:essential-set-edges**, and **tv:minimum-window** itself. Its correct position in the ordering is directly after **tv:essential-mouse**, where it appears without brackets.

You can read the order of flavor components by reading all unbracketed flavors from top to bottom, ignoring punctuation. If :duplicates is no, this is all that is displayed.

For information on how the order is determined: See the section "Ordering Flavor Components" in *Symbolics Common Lisp.*

Show Flavor Dependents Command

Show Flavor Dependents *flavor keywords*

Shows the names of flavors that are dependent on this flavor.

keywords	:brief, :detailed, :duplicates, :functions, :initializations, :instance variables, :levels, :match, :methods, :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*
:brief	{yes, no} yes indicates that the output should not be indented to show the structure. The default is no.
:duplicates	{yes, no} Indicates whether or not to display duplicate occurrences of flavors. The default is no.
:levels	{all, *integer*} Specifies how many levels of indirect dependency to display. The default is all, which shows all levels. For some flavors the output can be voluminous, and it is helpful to use :levels to pare it down.

A dependent flavor is a flavor that uses this flavor as a component (directly or indirectly). This is useful in program development or debugging, to answer the question "What flavors will be affected if I change the definition of this flavor?" For example:

```
Command: Show Flavor Dependents TV:SCROLL-WINDOW-WITH-DYNAMIC-TYPEOUT
     --> TV:SCROLL-WINDOW-WITH-DYNAMIC-TYPEOUT
           TV:BASIC-PEEK
             TV:PEEK-PANE
           TV:BASIC-TREE-SCROLL
             LMFS:AFSE-MIXIN
               LMFS:FSMAINT-AFSE-PANE
             LMFS:FSMAINT-HIERED-PANE
             TV:MOUSABLE-TREE-SCROLL-MIXIN
               TV:TREE-SCROLL-WINDOW
```

The output is indented to clarify which flavor is built on which component flavors. The structure of the output is the inverse of the output of Show Flavor Components. In this example, **tv:basic-peek** is a direct dependent of **tv:scroll-window-with-dynamic-typeout**, and **tv:peek-pane** is a direct dependent of **tv:basic-peek**.

Show Flavor Differences Command

Show Flavor Differences *flavor1 flavor2 keywords*

Shows the characteristics that two flavors have in common, and the characteristics in which they differ.

keywords	:match and :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp*.
:match	{*string*} Displays only those generic functions or messages that match the given substring.

This is most useful for two flavors that share many characteristics. Here is some sample output:

```
Command: Show Flavor Differences TV:ESSENTIAL-WINDOW TV:MINIMUM-WINDOW
     --> TV:ESSENTIAL-WINDOW and TV:MINIMUM-WINDOW have
         common components:
            flavors...
         TV:MINIMUM-WINDOW has other components:
            flavors...

         Differences in :ACTIVATE methods from TV:ESSENTIAL-WINDOW
                                                to TV:MINIMUM-WINDOW
            TV:SHEET before, primary,
            TV:ESSENTIAL-ACTIVATE after [added]
```

```
Differences in handling of :BURY
   Flavor TV:ESSENTIAL-WINDOW does not handle :BURY
   Methods of TV:MINIMUM-WINDOW:
      TV:ESSENTIAL-EXPOSE wrapper, TV:ESSENTIAL-ACTIVATE
   more differences...
```

First, the common components are displayed. Second, the extra components of either (or both) flavors are displayed. Third, any differences in handling of generic functions are displayed.

In this example, **tv:minimum-window** has one method for **:activate** that **tv:essential-window** does not have: an **:after** method provided by flavor **tv:essential-activate**. The term `[added]` indicates that this method is defined for the second flavor but not for the first flavor. If the command had been given such that *flavor-1* was **tv:minimum-window** and *flavor-2* was **tv:essential-window**, the term would have been `[deleted]`. To interpret which flavor "adds" or "deletes" a method, look at the line that defines the perspective: `"Differences in :ACTIVATE methods from TV:ESSENTIAL-WINDOW to TV:MINIMUM-WINDOW"`.

When comparing two complex flavors, the output can be voluminous. You can use :match to pare down the output so it answers a specific question. For example:

```
Command: Show Flavor Differences DYNAMIC-LISP-LISTENER SHEET :Match
         screen
      --> information about common and different components...
         Difference in handling of :FULL-SCREEN
           Method of DW::DYNAMIC-LISP-LISTENER: TV:ESSENTIAL-SET-EDGES
           Flavor TV:SHEET does not handle generic operation :FULL-SCREEN
         another difference...
         5 local functions found with no differences:
           TV:SCREEN-MANAGE-RESTORE-AREA
           TV:SCREEN-MANAGE-CLEAR-AREA
           TV:SCREEN-MANAGE-CLEAR-UNCOVERED-AREA
           TV:SCREEN-MANAGE-CLEAR-RECTANGLE
           TV:SCREEN-MANAGE-MAYBE-BLT-RECTANGLE

         120 differing local functions were found that did not
         contain the substring "screen".
```

Show Flavor Handler Command

Show Flavor Handler *generic-function flavor keywords*

Provides information on the handler that performs *generic-function* on instances of *flavor.*

keywords :code and :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*

:code {yes, no, detailed} Specifies whether the Lisp code of the handler should be displayed. The default is no. Yes displays a template that resembles the actual code of the handler. Detailed displays the actual code of the handler. This displays some internal functions and data structures of the Flavors system. For most purposes, yes is more useful than detailed.

If the handler is a single method (not a combined method), its function spec is given:

```
Command: Show Flavor Handler CHANGE-STATUS CELL
     --> The handler for CHANGE-STATUS of an instance of CELL is
         the method (FLAVOR:METHOD CHANGE-STATUS CELL).
         The method-combination type is :AND :MOST-SPECIFIC-LAST.
```

If the handler is a combined method, the method combination type and order of methods are displayed. In the following example, the methods used in the combined method are represented by the names of the flavors that implement them. Even in this abbreviated format, the representation of the method is mouse-sensitive.

```
Command: Show Flavor Handler CHANGE-STATUS BOX-WITH-CELL
     --> The handler for CHANGE-STATUS of an instance of
         BOX-WITH-CELL is a combined method, with
         method-combination type :AND :MOST-SPECIFIC-LAST.
         The methods in the combined method, in order of
         execution, are: CELL, BOX-WITH-CELL
```

For combined methods, :code yes is useful. It requests a template that resembles the actual code of the handler:

```
Command: Show Flavor Handler CHANGE-STATUS BOX-WITH-CELL :Code yes
     --> The handler for CHANGE-STATUS of an instance of
         BOX-WITH-CELL is a combined method, with
         method-combination type :AND :MOST-SPECIFIC-LAST.
         (DEFUN (FLAVOR:COMBINED CHANGE-STATUS BOX-WITH-CELL)
                 (SELF SYS:SELF-MAPPING-TABLE FLAVOR::.GENERIC.
                  &REST FLAVOR::DAEMON-CALLER-ARGS.)
           (AND call (FLAVOR:METHOD CHANGE-STATUS CELL)
                call (FLAVOR:METHOD CHANGE-STATUS BOX-WITH-CELL)))
```

Show Flavor Initializations Command

Show Flavor Initializations *flavor keywords*

Shows the initialization keywords accepted by **make-instance** of this flavor, and any default initial values.

keywords	:detailed, :locally, :match, :sort, and :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*
:detailed	{yes, no} The default is no, which requests the allowed initialization keywords that can be given to **make-instance** of this flavor, including init keywords and initable instance variables. If :detailed is yes, any additional instance variables are also shown; these are not initable instance variables. They are initialized by default values given in the **defflavor** form. Also, any initialization methods are shown. In other words, when :detailed is no, you see the initializations from an external perspective (useful for making an instance). When :detailed is yes, you see the initializations from an internal perspective and gain information about how the flavor is constructed internally.
:locally	{yes, no} If yes, inherited initializations are not shown. The default is no, which requests all initializations defined for this flavor or inherited by this flavor.
:match	{*string*} Requests only those initializations matching the given substring.

For example:

```
Command: Show Flavor Initializations BOX-WITH-CELL :Detailed
    --> Instances of BOX-WITH-CELL are created in the default area
        Another area can be specified with the keyword :AREA
        Initialization keywords that initialize
          instance variables:
           :BOX-X → BOX-X
           :BOX-Y → BOX-Y
           :SIDE-LENGTH → SIDE-LENGTH, default is *SIDE-LENGTH*
           :STATUS → STATUS, default is (IF (EVENP (RANDOM 2))
                                             ':ALIVE ':DEAD)
           :X → X
           :Y → Y
        Initialization method:
           MAKE-INSTANCE method: BOX-WITH-CELL
```

Show Flavor Instance Variables Command

Show Flavor Instance Variables *flavor keywords*

Shows the state maintained by instances of the given flavor.

keywords :detailed, :locally, :match, :output destination and :sort. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*

:detailed {yes, no} If yes, the attributes of the instance variables are shown, such as their accessibility or initializations. The default is no.

:locally {yes, no} If yes, inherited instance variables are not shown. The default is no, which shows all instance variables defined for this flavor or inherited by this flavor.

:sort {alphabetical, flavor} If flavor, each instance variable is displayed along with the component flavor that provides it. The default is alphabetical.

For example:

```
Command: Show Flavor Instance Variables CELL
    --> NEIGHBORS
        NEXT-STATUS
        STATUS
        X
        Y
```

Show Flavor Methods Command

Show Flavor Methods *flavor*

Displays all methods defined for the given flavor.

Keywords	:locally, :match, :output destination, and :sort. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*
:locally	{yes, no} If yes, inherited methods are not shown. The default is no, which shows all methods defined for this flavor or inherited by this flavor.
:match	{*string*} Requests only those methods for generic functions that match the given string.

Each line of output contains the name of the generic function, followed by the name of each flavor that provides a method for the generic function. If the method is not a primary method, its type is also displayed.

```
Command: Show Flavor Methods BOX-WITH-CELL
    --> ALIVENESS method: CELL
        CHANGE-STATUS: methods: CELL, BOX-WITH-CELL
        COUNT-LIVE-NEIGHBORS method: CELL
        :DESCRIBE method: FLAVOR:VANILLA
        others...
```

This command is similar to Show Flavor Operations. See the section "Show Flavor Operations Command", page 274. The difference between the two commands is in the perspective:

Show Flavor Methods displays information from an internal perspective, answering the question: What methods are defined for this flavor, or inherited from its component flavors?

Show Flavor Operations displays information from an external perspective, answering the question: What operations (generic functions and messages) are supported by instances of this flavor?

Show Flavor Operations Command

Show Flavor Operations *flavor keywords*

Shows all operations supported by instances of the given flavor, including generic functions and messages.

keywords	:detailed, :match, and :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*
:detailed	{yes, no} If yes, the display shows the arguments of each operation. The default is no.
:match	{*string*} Shows only those operations matching the given substring.

For example:

```
Command: Show Flavor Operations BOX-WITH-CELL
    --> ALIVENESS
        CHANGE-STATUS
        COUNT-LIVE-NEIGHBORS
        :DESCRIBE
        MAKE-INSTANCE
        SYS:PRINT-SELF (:PRINT-SELF)
        others...
```

One of the operations can be performed by using the generic function **sys:print-self** or sending the message **:print-self.** This operation was defined with **defgeneric,** using the **:compatible-message** option.

This command is similar to Show Flavor Methods. See the section "Show Flavor Methods Command", page 274. The difference between the two commands is in the perspective:

Show Flavor Operations displays information from an external perspective, answering the question: What operations (generic functions and messages) are supported by instances of this flavor?

Show Flavor Methods displays information from an internal perspective, answering the question: What methods are defined for this flavor, or inherited from its component flavors?

Show Flavor Functions Command

Show Flavor Functions *flavor keywords*

Shows internal flavor functions for the given flavor.

keywords	:locally, :match, :output destination, and :sort. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*
:locally	{yes, no} If yes, inherited internal flavor functions are not

	shown. The default is no, which shows all internal flavor functions defined for this flavor or inherited by this flavor.
:match	{*string*} Displays only those internal functions that match the given substring.

Internal flavor functions are defined by **defun-in-flavor**, **defmacro-in-flavor**, and **defsubst-in-flavor**. See the section "Defining Functions Internal to Flavors" in *Symbolics Common Lisp.*

```
Command: Show Flavor Functions TV:MAKE-WINDOW
    --> TV:ADJUST-MARGINS
        SI:ANY-TYI-CHECK-EOF
        SI:ASSURE-INSIDE-INPUT-EDITOR
        others...
```

Show Generic Function Command

Show Generic Function *operation keywords*

Shows information on the given *operation*, which can be a generic function or message.

keywords	:flavors, :methods and :output destination. See the section "Keyword Options for Show Flavor Commands" in *Symbolics Common Lisp.*
:methods	{yes, no} yes displays all methods for the generic function, and their types.
:flavors	{yes, no} yes displays the flavors that implement methods for the generic function.

For example:

```
Command: Show Generic Function CHANGE-STATUS
    --> Generic function CHANGE-STATUS takes arguments: (CELL-UNIT)
        This is an explicit DEFGENERIC in file SYS:EXAMPLES;FLAVOR-LIFE.
        Method-combination type is :AND :MOST-SPECIFIC-LAST.
```

Show Font Command

Show Font *font*

Displays all characters of the font. You can get a list of the fonts loaded by pressing HELP after typing Show Font or by clicking on List Fonts in the font editor. You enter the font editor by using the Edit Font command with no arguments.

font	Font name.

Show GC Status Command

Show GC Status

Displays statistics about the garbage collector.

Show Herald Command

Show Herald *keywords*

Displays the herald message. The herald is part of the screen display on a cold booted machine. It shows you the name of the FEP file or partition for the current world load, any comment added to the herald, a measure of the physical memory and swapping space available, the versions of the systems that are running, the site name, and the machine's own host name.

keywords	:Detailed,:Output Destination
:Detailed	{yes no} Whether or not to print the version information in full detail. The default is no.
:Output Destination	{*buffer file printer stream window*} Where to display the information. The default is the current window.

Show Hosts Command

Show Hosts *host-spec*

Asks each of the *hosts* for its status, and prints the results. If no hosts are specified, asks all hosts on the Chaosnet. Hosts can be specified by either name or octal number.

For each host, a line is displayed that either says that the host is not responding or gives metering information for the host's network attachments. If a host is not responding, probably it is down or there is no such host at that address. A Lisp Machine can fail to respond if it is looping inside **without-interrupts** or paging extremely heavily, such that it is simply unable to respond within a reasonable amount of time.

host-spec	A host or list of hosts (names or network addresses) or sites, separated by commas.

For example:

```
    Show Hosts Wombat
    Show Hosts chaos|23557
    Show Hosts Wombat,Kangaroo,Wallaby
```

The display looks like this:

```
Chaosnet host status report.  Type Control-Abort to quit.
Site  Name/Status Subnet    #-in  #-out  abort  lost  crc     ram bitc other
23557 Wombat          51 4615995  89709      0    22 8214524   07749784   919
```

Site	The chaosnet address of the *host*, in this case 23557.
Name/Status	The name of ths *host*, in this case Wombat.
Subnet	For sites with large networks, the number of the subnet on which *host* resides, in this case 51.
#-in	The number of chaosnet packets received by *host* since it was cold booted last.
#-out	The number of chaosnet packets transmitted by *host*.
abort	The number of packets *host* attempted to send but was unsuccessful due to network collisions.
lost	The number of incoming packets that *host* was forced to discard.
crc	Cyclic Redundancy Check. The number of packets that failed this check, because they were damaged either in transmission or by *host* when they were received.
ram	Random Access Memory. The number of hardware memory errors *host* has experienced, in this case 0.
bitc	Bit Count. The number of packets whose actual bit count did not match the bit count recorded for them.
other	Other errors.

Show Legal Notice Command

Show Legal Notice

Displays the Symbolics Legal Notices, such as copyrights and trademarks.

Show Lisp Context Command

Show Lisp Context

Displays the currently enabled lisp context (Zetalisp or Common-Lisp), the current package, and the current input and output bases. For example:

```
    Zetalisp syntax is now enabled.
    Package is USER; Input base is 10; Output base is 10.
```

Show Login History Command

Show Login History

Prints one line for each time the login command has been used since the world was last cold booted. It also shows the logins done during the creation of the world load. Each line contains the name of the user who logged in, the name of the machine on which the world load was running at that time, and the date and time. This command also shows the name of an init file, if one was loaded. If you cold boot, log in, and then do Show Login History, the last line refers to your own login and all previous lines refer to logins that were done before doing Save World (or running **zl:disk-save**).

This information is useful to determine how many times a world load has been disk-saved, on what machines it was disk-saved, and who disk-saved it.

The first couple of lines do not contain any date or time, because they were made during the initial construction of the world load before it found out the current time. Names of users at other sites that are not in the local site's namespace search list are qualified with the site's namespace name and a vertical bar. The user LISP-MACHINE is the dummy user used by **si:login-to-sys-host** when new world loads are created.

Show Machine Configuration Command

Show Machine Configuration *host*

Shows the board-level hardware information about any 3600-family machine on the same network as your machine.

host	The name of a 3600-family machine. The default is your machine.

This information is useful for service personnel. You might be asked for the machine serial number (in line 3) if you call Symbolics Software Support. The display from Show Machine Configuration looks like this:

```
:Show Machine Configuration (A host) acme-sling-shot
Chassis (PN 170219, Serial 230) in Chassis or NanoFEP:
    Manufactured on 1/10/85 as rev 1, functions as rev 1, ECO level 0
    Machine Serial Number: 4185
Datapath (PN 170032, Serial 1253):
    Manufactured on 9/20/83 as rev 3, functions as rev 3, ECO level 0
Sequencer (PN 170042, Serial 2576):
    Manufactured on 4/21/85 as rev 4, functions as rev 4, ECO level 0
Memory Control (PN 170052, Serial 1381) in Memory Control or IFU:
    Manufactured on 12/3/83 as rev 5, functions as rev 5, ECO level 0
Front End (PN 170062, Serial 2380) in FEP:
    Manufactured on 2/1/84 as rev 5, functions as rev 5, ECO level 0
512K Memory (PN 170002, Serial 1258) in LBus slot 00:
    Octal Base address: 0
    Manufactured on 3/2/84 as rev 2, functions as rev 2, ECO level 0
512K Memory (PN 170002, Serial 2572) in LBus slot 01:
    Octal Base address: 2000000
    Manufactured on 2/22/85 as rev 2, functions as rev 2, ECO level 0
512K Memory (PN 170002, Serial 140) in LBus slot 02:
    Octal Base address: 4000000
    Manufactured on 1/19/83 as rev 2, functions as rev 2, ECO level 0
IO (PN 170157, Serial 356) in LBus slot 03:
    Octal Base address: 6000000
    Manufactured on 9/22/84 as rev 6, functions as rev 6, ECO level 0
512K Memory (PN 170002, Serial 2932) in LBus slot 04:
    Octal Base address: 10000000
    Manufactured on 4/11/85 as rev 2, functions as rev 2, ECO level 0
FEP Paddle Card (PN 170069, Serial 943) in FEP -- PADDLE side:
    Manufactured on 3/21/85 as rev 1, functions as rev 1, ECO level 0
IO Paddle Card (PN 170245, Serial 3) in LBus slot 03 -- PADDLE side:
    Manufactured on 4/20/84 as rev 1, functions as rev 1, ECO level 0
    Ethernet Address: 08-00-05-03-18-00
    Monitor Type: Philips
```

Show Mail Command

Show Mail *file-spec*

Displays your mail inbox on the screen. If there is more than one screenful, it pauses between screenfuls displaying `--More--` at the bottom.

file-spec	The pathname of the mail inbox to be read. The default is the default inbox.

Show Monitored Locations Command

:Show Monitored Locations

Displays all of variables and other locations in memory that you are monitoring via the :Monitor Variable command, the **dbg:monitor-location** function, and so on.

Show Namespace Object Command

Show Namespace Object *class name keywords*

Shows the information in the namespace database for *name.*

class	The type of object. Possible types are: Host, Site, Namespace, User, Network, and Printer.
name	The name of the object, that is a user-id, the name of a machine, or the name of the site or namespace.
keywords	:Locally
:Locally	{yes, no} Whether to show the information cached in the local machine or to consult the namespace server. The default is no, to consult the server. The mentioned default is yes.

Here is what the namespace object for a user might look like:

```
Command: Show Namespace Object User KJONES
View in namespace ACME:
USER KJONES
LISPM-NAME KJones
PERSONAL-NAME "Jones, Kingsley"
HOME-HOST ACME
MAIL-ADDRESS kjones ACME
LOGIN-NAME kjones VAX01
NICKNAME King
WORK-ADDRESS "Building 3-701"
WORK-PHONE 5891
BIRTHDAY "19 June"
PROJECT Database
SUPERVISOR "Finklestein"
USER-PROPERTY :USUAL-LOGIN-HOST wombat
```

Show Notifications Command

Show Notifications *keywords*

Re-displays any notifications that have been received. Notifications are asynchronous messages from Genera.

keywords	:before,:from,:matching,:newest,:oldest,:since,:through
:before	A date to serve as one limit for notifications to show: `:before 11/1/84`
:from	A number to use as the first notification to show.
:matching	A string to search for. Only show notifications that contain that string: `:matching hardcopy`
:newest	A number of notifications to show, for instance, the ten most recent ones: `:newest 10` Using this keyword without a number is the same as `:newest 1`.
:oldest	A number of notifications to show, for instance, the ten earliest ones: `:oldest 10` Using this keyword without a number is the same as `:oldest 1`.
:since	A date to serve as one limit for notifications to show.
:through	A number to use as the last notification to show: `:through 17`

Using the Show Notifications command with no keyword arguments means show all notifications in reverse chronological order (most recent first).

Show Object Command

Show Object *name keywords*

Show Object tries to tell you all the interesting information about any object (except for array contents). Show Object knows about areas, structures, packages, pathnames, systems, variables, functions, flavors, and resources. It displays the attributes of each. Show Object *symbol* will tell you about *symbol*'s value, its definition, and each of its properties.

name	Any Lisp object.
keywords	:Type
:Type	{all, area, structure, partition, package, logical-host, pathname system, variable, function, flavor, resource}. The default is all.

Show Printer Defaults Command

Show Printer Defaults

Displays the current default printer(s). If you send all your hardcopy output to one printer, this is displayed as

```
Default Printer: printer-name
```

If you use a different printer for text and screen hardcopy, this is displayed as

```
Default Text Printer: printer-name1
Default Bitmap Printer: printer-name2
```

Show Printer Status Command

Show Printer Status *printer*

Displays the print queue for the specified printer or printers.

printer	The name of a printer or printers (separated by commas) whose print queue to show, or All to show all the queues for all printers at your site. The default is your current printer default. If your text printer and your bitmap printer are different, your text printer is used as the default for Show Printer Status.

The display of requests is mouse sensitive and can be clicked on to select arguments for the Delete Printer Request and Restart Printer Request commands. This is only true for print spoolers running Release 7.0.

Show Source Code Command

:Show Source Code *compiled-function-spec* `c-X c-D`

Displays the source code for a function. This command works only when your code resides in an editor buffer. The output is mouse sensitive only when the function is compiled with source locators. When you specify a compiled function for which you want to see source code – for example, **myfunction** – the Debugger displays the source code for **myfunction** beneath the following message:

Source code for `MYFUNCTION:`

If **myfunction** were not compiled with source locators, the Debugger would still display the source code, but the output would not be mouse sensitive. The Debugger would display the source code only after giving you this message:

```
Function MYFUNCTION has no source locators; the code will not be sensitive.
```

compiled-function-spec
: The name of a compiled function for which you want to see source code. (Default is the function in the current frame.)

Suggested mouse operations

When a function has been compiled using source locators – mapping source code to PCs via the editor's `c-m-sh-C` command – you can perform the following mouse operations:

- To use this command with the mouse: Type in the :Show Source Code command. When the Debugger asks you for a compiled-function-spec, point the mouse at the name of a compiled function previously displayed in the output of another command, such as :Show Backtrace, and click `Mouse-Left`.

- To set a breakpoint: Point the mouse at a form (a code fragment) in the displayed source code and press `c-m-Mouse-Left`.

- To clear a breakpoint: Point the mouse at a form (a code fragment) in the displayed source code and press `c-m-Mouse-Middle`.

- To evaluate a code fragment: Point the mouse at a form in the displayed source code and press `m-Mouse-Middle`.

Show System Definition Command

Show System Definition *system keywords*

Displays a the system definition of *system* including its current patch level, status (experimental or released), and the files that make up the system.

system	A loaded system.
keywords	:Detailed
:Detailed	{Yes,No} Whether to list all the component systems of the system or not. The default is no, the mentioned default is yes.

Show System Modifications Command

Show System Modifications *system-name keywords*

With no arguments, Show System Modifications lists all the systems present in this world and, for each system, all the modifications that have been loaded into this world. For each modification it shows the major version number (which will always be the same since a world can only contain one major version), the minor version number, and an explanation of what the modification does, as entered by the person who made it.

If Show System Modifications is called with an argument, only the modifications to *system-name* are listed.

system-name	The system for which to show modifications. The default is All.
keywords	:Author, :Before, :From, :Matching, :Newest, :Number, :Oldest, :Since, :Through

:Author — A name. Show modifications by a particular person. For example:

```
:show modifications system :author kjones
```

would only show those modifications made by the person whose user ID is kjones.

:Before — A date to serve as one limit for modifications to show:

```
:before 11/1/84
```

:From — A number to use as the first modification to show.

:Matching — A string to search for in the comments and only show modifications whose comment contain that string:

```
:matching namespace
```

:Newest — A number of modifications to show, for instance the ten most recent ones:

```
:newest 10
```

Using this keyword without a number is the same as `:newest 1`.

:Number — A number. Show only this particular modification. For example:

```
Show Modifications :number 6
```

would show modification number 6.

:Oldest — A number of modifications to show, for instance the ten earliest ones:

```
:oldest 10
```

Using this keyword without a number is the same as `:oldest 1`.

:Since — A date to serve as one limit for modifications to show.

:Through — A number to use as the last modification to show:

```
:through 17
```

Show Users Command

Show Users *user-spec keywords*

Shows the users logged into *host.*

user-spec — {*user user@host user@site @host @site*} The user or users to locate and the host or site to search. The default is to show all users logged in at the local site.

keywords — :Format :Order :Output Destination

:Format — {brief normal detailed} How much information to display.

Brief — Gives the user's login name, the host name and the idle time only.

```
kjones wombat 1:12
```

Normal — Gives the user's full name and information about the world they are running in addition to the information displayed by brief.

```
kjones  Kingsley Jones  Wombat 1:12 Computer Room (x515) (Release 7.0)
```

Detailed — Displays the information in the user's namespace object.

The default when only one user is specified is full, if more than one is specified the default is brief.

:order — {host idle user} The order in which to sort the entries in the display.

host	Sorts alphabetically by host name.
user	Sorts alphabetically by user login name.
idle	Sorts by idle time, from active to greatest idle time.

:output destination

	{*buffer file stream window*} Where to display the information. The default is the current window.

20.26 *Start* Commands

Start GC Command

Start GC *keywords*

Turns on the garbage collector.

keywords	:Dynamic, :Ephemeral, :Immediately
:Dynamic	{yes, no} Dynamic Level of incremental GC.
:Ephemeral	{yes, no} Ephemeral Level of incremental GC.
:Immediately	{yes, no} Perform a complete garbage collection right now.

Unmonitor Variable Command

:Unmonitor Variable *symbol keyword*

Stops monitoring one or all special variables in memory.

symbol	{*location*, RETURN} A *location* specifies one location that you want to stop monitoring. Enter the name of a symbol and, optionally, its Value-Cell or Function-Cell. (See the :Cell keyword description below.) Press the RETURN key if you want to stop monitoring all locations.
keyword	:Cell
:Cell	{Value-Cell, Function-Cell} Specifies which cell within the location you want to stop monitoring. The Debugger gives you two choices: Value-Cell or Function-Cell. (Default is Value-Cell.)

Suggested mouse operations

- To unmonitor a location: Point the mouse at a locative, structure slot, or instance variable that was previously monitored and press `c-m-sh-Mouse-Middle`.

20.27 *Undelete* Commands

Undelete File Command

Undelete File *file-spec keywords*

Undeletes a deleted file, if the host on which the file resides supports undelete. It prompts for the name of a file to undelete. It displays a message if the specified file does not exist.

file-spec	The pathname of the file to be undeleted. The default is the usual file default.
keywords	:Query
:Query	{yes, no, ask} Whether to ask for confirmation before removing the delete flag on the file. The default is no.

Index

	What is a World Load? 140
	What is Customizing? 93
Checking on	What the Machine Is Doing 183
	What to Do After Reading a Zmail Message 83
Finding Out	What You Have Typed 73
	When and How to Use the Garbage Collector 157
Why Can't I Write Out Files	When I Have Free Disk Space? 141
	where-is function 185
Init File Form: Fixing	White Space for Kill/Yank Commands 103
Init File Form:	White Space in Lisp Code 103
Keywords for	**who-calls** and **what-files-call** 183, 185
	who-calls function 183
si:	**who-calls-unbound-functions** function 184
Find	Whole Word Candidates 50
	Why Can't I Write Out Files When I Have Free Disk Space? 141
	Why Do You Name Machines and Printers? 140
Converse Pop-up Message	Window 89
Delete contents of	window 177
Document Examiner	Window 54
Erase	window 177
Headers	window 84
HELP Key in Any Zmacs Editing	Window 114
Message	window 84
Scrolling summary	window 83
Summary	window 80
Typeout	window 71
User options	window 111
Using the mouse to scroll the message	window 80
Zmail Message	Window 77
Zmail Summary	Window 77
Display status of	window area 192
	Window Operations 210
Set	Window Options Command 262
Default	Windows 13
Getting Acquainted with Dynamic	Windows 26
Interaction with Completion and Typeout	Windows 113
Introduction: Selecting and Creating	Windows 13
Mouse Gestures on Dynamic	Windows 211
Selecting and Creating	Windows 13
Typeout	Windows in Zmacs 117
	Windows (W) 195
Logging in	Without Processing Your Init File 110
Backward	Word 68
Forward	Word 68
Find Whole	Word Candidates 50
	Word Operations 208
How the Inspector	Works 204
A Brief Introduction to the Lisp	World 175
Poking Around in the Lisp	World 183
Variables for Examining the Lisp	World 186
Copy	World Command 231
Optimize	World Command 248
Save	World Command 251
What is a	World Load? 140
	World load disk-saves 189
	Write a breakpoint into a program 188
	Write Buffer (m-X) Converse Command 90
	Write Conversation (m-X) Converse Command 90
	Write File Zmacs command 75
Why Can't I	Write Out Files When I Have Free Disk Space? 141