MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 209 April, 1981

Operating the Lisp Machine

David A. Moon
Allan C. Wechsler

This document is a draft copy of a portion of the Lisp Machine window system manual. It is being published in this form now to make it available, since the complete window system manual is unlikely to be finished in the near future. The information in this document is accurate as of system 57, but is not guaranteed to remain 100% accurate.

This document explains how to use the Lisp Machine from a non-programmer's point of view. It explains the general characteristics of the user interface, particularly the window system and the program-control commands. This document is intended to tell you everything you need to know to sit down at a Lisp machine and run programs, but does not deal with the writing of programs. Many arcane commands and user-interface features are also documented herein, although the beginning user can safely ignore them.

Table of Contents

1. The Console

This chapter describes the basic logical characteristics of the devices that are used to talk to the Lisp Machine. These include one or more bit-raster displays, a specially extended keyboard, and a graphical input device called a *mouse*. Collectively these form a complete and extremely flexible user interface, called the *console*.

1.1 The Screen

The Lisp Machine generally displays its output on one or more bit-raster displays. The *window system* controls how display space is allocated to various programs, systems, menus, and messages, allowing the user to shift his attention between these easily. This is discussed more fully later, in the chapter called "The Window System".

1.2 The Keyboard

We are in the process of changing over from the old Knight keyboard to a new type. Most of the current software can deal with both kinds of keyboards. In particular, the new keyboards can generate all the characters that the old keyboards can. However, transition problems may be encountered over the next few months. New software may have been written with specific features of the new keyboards in mind, and thus may be difficult to use from the old keyboards. Most system software can still be used fairly easily from the old keyboards, but exceptions exist and will probably become more frequent.

These problems will be resolved with time as programs and people become adapted to the new keyboards.

This document will assume that a new keyboard is being used. Where applicable, the sequence that would have to be typed on the old keyboards to get the same result will also be given.

There are 100 physical keys on the new keyboards. The keyboard has unlimited rollover, meaning that a keystroke is sensed when the key is depressed, no matter what other keys are depressed at the time.

Actually, the hardware can tell exactly what physical keys are being pressed at any given moment: it knows when any key is depressed and when it is released. This means that the Lisp Machine *could* be programmed to interpret the keyboard in any manner whatsoever: in this sense the keyboard is completely "soft". But the Lisp Machine has already been programmed to interpret the keyboard input in a useful way, and such reprogramming would be necessary only for the most special needs.

The keys are divided into three groups: function keys, character keys, and modifier keys. Character keys are generally small and gray, while function keys are generally large and blue.

Function Keys ABORT, BREAK, CALL, CLEAR-INPUT, CLEAR-SCREEN, DELETE, END, HELP, HOLD-OUTPUT, LINE, MACRO, NETWORK, OVER-STRIKE, QUOTE, RESUME, RETURN, RUBOUT, STATUS, STOP-OUTPUT, SYSTEM, TAB,

TERMINAL

In addition to these, there are some gray function keys: four on the right with fingers pointing in various directions, called HAND-UP, HAND-DOWN, HAND-RIGHT, and HAND-LEFT; and four on the left with the roman numerals I, II, III, and IV.

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 : - = { } ( ) ' \ ; " , . /
In addition, ALT-MODE and the space bar are character keys, despite being blue.

Modifier Keys MODE-LOCK, CAPS-LOCK, REPEAT, ALT-LOCK, left TOP, left GREEK, left SHIFT, left HYPER, left SUPER, left META, left CTRL, right TOP, right GREEK, right SHIFT, right HYPER, right SUPER, right META, right CTRL

1.2.1 Modifier Keys

Modifier keys are intended to be held down while typing another key.

None of the software distinguishes between the left and right versions of TOP, GREEK, SHIFT, HYPER, SUPER, META, and CTRL. When one of these is specified, either physical key (or both) will work. The incantations used for warm-booting and cold-booting (CTRL/META/CTRL/META/RETURN and CTRL/META/CTRL/META/RUBOUT, respectively) are the only exceptions; these require that both control keys and both meta keys be held down.

The MODE-LOCK, CAPS-LOCK, and ALT-LOCK keys hold themselves down once depressed, and must be explicitly released by striking them again.

In this document, the action of holding down some modifier keys while striking some other key will be represented with a slash notation. For example, if you are told to type HYPER/META/J, you will accomplish this by holding down the HYPER and META keys while you strike "j".

The SHIFT, TOP, and GREEK keys are intended to modify character keys to produce printed characters. Some printed characters can be produced in more than one way. The software never distinguishes between the possible ways of producing a particular printed character. For example, typing the "9" key while holding down SHIFT produces an open parenthesis, and so does striking the "(" with no modifiers. There is no difference between these two parentheses.

GREEK is also called FRONT since it causes a character key to generate the character written on the front of the key. These characters are greek letters on the alphabetic keys and special punctuation symbols on the other keys.

The HYPER, SUPER, META, and CTRL keys are used by programs like the Editor whose commands are mostly single characters. They are intended to modify other keys to produce commands for these programs.

The CAPS-LOCK key, when depressed, causes all typed letters (A through Z) to be interpreted as if the shift key were down. CAPS-LOCK does not affect the interpretation of non-alphabetic character keys.

The functions of the ALT-LOCK and MODE-LOCK keys have not been assigned as of this writing.

1.2.2 Function Keys

The function keys allow the user to do certain useful operations in very few keystrokes. Some of these operations may be performed anytime, no matter what the Lisp Machine is doing at the moment. Other operations are only defined in certain contexts, and the keys that perform these operations may do different things at different times, or may be ignored if the associated operation is not relevant to what the Lisp Machine is doing.

The four finger keys are function keys, but functions have not yet been assigned to them. HAND-UP and HAND-DOWN may eventually be used for answering yes-or-no questions. FRONT-shifting a finger key gives a printing character: circle-plus, circle-minus, circle-cross, and circle-slash are typed this way.

The roman-numeral keys are not used by the system. One useful thing you can do with them is to put your own editor commands or keyboard macros on them.

The operations performed by the various function keys are summarized in the Index of Function Keys (see chapter 9, page 35).

1.3 The Mouse

The mouse is a pointing device that can be moved around on a flat surface. These motions are sensed by the Lisp Machine, which usually responds by moving a cursor around on the screen in a corresponding manner. The shape of the cursor varies, depending on context. See chapter 10, page 40.

There are three buttons on the mouse, called Left, Middle, and Right. They are used to specify operations to be performed. Typically the user points at something with the mouse and specifies an operation by clicking the mouse buttons. Rapid double clicks are conventionally distinguished from single clicks. Thus, in any specific context, there are up to six operations that can be performed with the mouse, invoked by Left, Left Double, Middle, Middle Double, Right, and Right Double 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 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. "Left Double Hold" means to click the left mouse button twice, holding it down the second time until the operation is complete.

Occasionally a long click is distinguished from a short one, as a Morse Code dash is distinguished from a dot. In these cases it doesn't matter exactly how long the button is held

down, as long as it is perceptibly longer than the usual rapid strike. Such a click will be described by the word "Long", as in "Right Long".

The mouse is completely "soft", like the keyboard. The Lisp Machine can be programmed to interpret the mouse in any desired fashion. The protocol that has been chosen, however, is extremely general and should suffice for almost all needs.

1.4 Booting the Machine

The Lisp machine can be "booted" by typing special combinations of keys on the keyboard. There are two kinds of booting; cold-booting completely reinitializes the machine, while warm-booting crashes it and then restarts it. Cold-booting is what you do when you have just switched on power to the machine. It is also a good idea to cold-boot when done with the machine so as to leave it in a completely clean state for the next person.

Warm-booting is a kludgey escape mechanism which allows you to restart the machine after it has crashed or when it is in some run-away state that for some reason you can't get out of by using the normal function keys. Warm-booting is not guaranteed to work, since it may have to restart the machine from any arbitrary, inconsistent state. However, warm-booting makes a reasonable attempt to get the system to come up so that you can continue, or save your work and then cold-boot.

Both kinds of booting start by resetting the machine hardware and reloading the microcode from the disk. Cold-booting then reloads the entire contents of virtual memory from a fresh copy on the disk, while warm-booting recovers the virtual memory you were working in. The Lisp system then comes up and runs for a few seconds, performing various initialization tasks, puts a greeting message at the top of the screen, and leaves you in a Lisp listener waiting for you to type in Lisp forms to be evaluated. See chapter 3, page 19.

The incantations used for warm-booting and cold-booting involve holding down all four control and meta keys simultaneously (the two to the left of the space bar and the two to the right of the space bar), and striking RUBOUT for a cold-boot or RETURN for a warm-boot. This combination of keys is extremely difficult to type accidentally.

On terminals where the mouse is plugged into the keyboard rather than the base of the display monitor, after a boot the mouse will not work until you first type a key. This is to avoid disturbing the machine while it is booting, which could make it forget whether it was to do a cold or warm boot. Normally you will not notice this, as the first thing you ought to do after cold-booting is to log in, which involves typing on the keyboard.

2. The Window System

The *window system* is responsible for the appearance of the display. It allocates display space and allows the user to shift his attention between many programs quickly and easily. Usually, all input and output are mediated by the window system.

2.1 The Geography of the Display

2.1.1 Windows

The display shows one or more *windows*, which are independent sub-displays. Most windows have *borders* (black outlines) around them. Some have a *label* which is usually in the lower left-hand corner. If the display is completely taken up by one window, the borders and label are often omitted.

Often a window will function as a *stream*, that is, a place that a program can read from and write to. In this way, a window provides a communications channel whereby the user can talk to a program.

Usually one of the windows on the screen is *selected*. This means that that window is the focus of your attention, and keyboard input is directed to it, so that a program reading from that window will read what you type. Most often the selected window will have a blinking cursor and the other windows on the display (if any) will not. Usually, what you type will be printed in the selected window at the place marked by the blinking cursor.

Windows can be *exposed*, meaning that they are fully visible on the screen, or else *deexposed*. A *deexposed* window may either be partially visible and partially covered by other windows, or entirely invisible. Deexposed windows can be brought back to the display in many ways, described later. The selected window is always exposed.

2.1.2 The Who-line and Run-lights

At the bottom of the display is the *who-line*. Here are displayed several pieces of status information. From left to right the who-line shows the date and time, your login name, the current package, the state of the process that is connected to the keyboard, and the state of an open file or the console idle time, the time since the keyboard was last typed upon or the mouse last clicked upon.

Many things can appear in the process state field; here are some of the most common and least self-explanatory. Don't expect to understand everything in this table (nor process states you see that are not in this table) completely. The process state in the who-line is there to give you a hint as to what is going on, but often has to do with internal details of the particular program being run, or with the more esoteric features of the Lisp machine process system.

RUN The process is running (as opposed to waiting for something).

TYI The process is waiting for you to type on the keyboard.

Output Hold — The process is trying to display on a window which cannot display right now. For instance, the window may not be exposed. [This needs to be explained somewhere in this document.]

LOCK — The process is waiting for some other process to unlock a lock. This is typical of a set of process states whose interpretation depends on knowledge of the particular program running.

NIL — The who-line is not looking at any process. Typically this is because no window is selected. The who-line normally looks at the process associated with the selected window.

ARREST — The process is *arrested*, which means it is not allowed to run. See the TERMINAL A command (page 36).

STOP — The process is not running because it does not have a "run reason".

OPEN, NETI, NETO, Net Wait, File Finish, etc. — The process is waiting for service from another machine, over the Chaosnet. Typically the other machine is a file server such as the AI Lab timesharing system.

Underneath the who-line are three *run-lights*, small horizontal bars which flicker on and off. The one on the right, approximately under the process state, is there when the processor is running. The one in the middle is there when the disk is running. The one to the left lights up when the garbage-collector is running.

Above the who-line there is a line of mouse-documentation, which is displayed in inverse video to make it easy to move your eyes to it from someplace else on the screen. This line tells you what the buttons on the mouse would do if you clicked them with the mouse where it currently is. If the line is blank, it means the default mouse buttons are in effect; clicking Left will select the window pointed-to, and clicking Right will get you the system menu (these are explained later).

2.1.3 Blinkers and Cursors

Scattered around the display are markers of various shapes and dynamic characteristics. They are all called *blinkers* for historical reasons, although only some of them blink.

One blinker is associated with the mouse: when you slide the mouse along a surface, that blinker moves in a corresponding direction. When the mouse is moved very rapidly, the mouse blinker gets big like Godzilla in order to maintain visibility. Small children should be taken out of the room before demonstrating this frightening feature.

Each window on the display normally keeps track of a position called the *cursor*, which is the place at which text will next be displayed in the window. The cursor is almost always marked by a rectangular blinker. In the selected window, this blinker flashes with a period of about half a second. This is how you tell at a glance which window is selected.

Sometimes when the Lisp Machine is very busy, the blinkers will falter because the program responsible for maintaining them is not getting run regularly. This does not indicate a malfunction but is part of the normal behavior of the Lisp Machine.

2.2 Using the Mouse

One blinker is always associated with the mouse, and whenever the mouse is moved, this blinker moves in a corresponding fashion. The blinker controlled by the mouse is often called the *mouse cursor*.

At any given moment, some program is listening to the mouse and is responsible for the appearance of the mouse cursor, the way in which it moves around, and what happens when the mouse buttons are pressed. You can tell who is listening to the mouse by looking at the shape of the mouse cursor. The cursor may change shape as you move it around the display, indicating that jurisdiction over the mouse is passing from one program to another. Also, individual programs may vary the shape of the mouse cursor to show exactly what functions are available in a particular context. See chapter 10, page 40.

2.2.1 Pointing at Something with the Mouse

The mouse is almost always used as a pointing device. One uses the mouse to indicate something on the display, and then one clicks the mouse buttons to specify an operation to be performed on or with the thing indicated.

Graphics-oriented programs may simply use the mouse as a device for indicating positions on the screen. One could imagine using the mouse to specify two points to be connected by a line, for example.

More often, however, the mouse is used to point to distinctly displayed objects on the screen. In many cases, an object thus indicated responds by changing its appearance in some way. Such objects are *mouse-sensitive*. The system convention is that when the mouse is pointing at or near a mouse-sensitive object, an outline is drawn around that object.

2.2.2 Clicking the Mouse

When you click a mouse button while the mouse is pointing at a mouse-sensitive object, the response to the mouse buttons depends on that object. Otherwise the response depends on the window that the mouse is pointing at. Often the shape of the mouse cursor is used as a clue to what the mouse buttons will do. Generally the mouse-documentation line at the bottom of the screen will also give a brief reminder of what the mouse buttons will do.

Sometimes, clicking the mouse buttons does not do something to the object indicated by the mouse, but rather calls up a *menu* of available operations. In this case options offered by the menu may pertain to the object that the mouse was pointing at, or they may be more general operations. Menus are discussed in detail below.

Although the usage of the mouse buttons varies depending on what the mouse is pointing at, there are system-wide conventions that most programs adhere to. Generally the Left button on the mouse will do something simple, while the Right button will do something more complicated, offering options and choices that the beginning user probably does not need to worry about. The use of the Middle button is less standardized.

If there are several things you might mean by pointing to a mouse-sensitive object and clicking, typically the Right button will give you a menu of operations from which you can choose, and the Left button will do the most "obvious" thing. If there is no "obvious" choice, it will generally do the last thing you chose in the same circumstances with the Right button. If there is only one thing you could mean by clicking, no distinction will be made among the three buttons.

The system convention is that if there isn't anything better for the mouse buttons to do, clicking Left selects the window the mouse is pointing at (see section 2.3.4, page 11) and clicking Right invokes the System Menu (see section 2.3.2, page 9). In this case, the mouse-documentation line at the bottom of the screen will be blank.

The following sections discuss the things you can do by clicking the mouse in more detail.

2.3 Manipulating Windows

When you step up to a free Lisp Machine, a window of the Lisp Listener type will fill the whole display. You can tell from the rectangular blinker near the upper left corner that this window is selected. The Lisp Listener window is used to talk to the Lisp interpreter. Its use is described in the chapter "The Lisp Listener" (chapter 3, page 19). If you want to start using Lisp as soon as possible, you can read that chapter now, as it does not depend on any of the others.

As you use the Lisp Machine, you may create new windows, reselect old ones, and move windows around the screen or reshape them to suit your taste. This section describes how these operations may be performed.

2.3.1 Menus

A particularly common and useful kind of window is the *menu*. Menus are windows that contain the names of several options. These options are mouse-sensitive, and you select one by pointing at it with the mouse and clicking.

Many menus are invisible unless it is time to select an option from them. Then they *pop up*, or appear suddenly on top of some previous display, obscuring what was there before. After you select an option with the mouse, the menu disappears and the operation that you specified is performed. If you don't want to select any of the options, you can simply move the mouse far out of the menu and it will disappear. When this is not the case, the menu will contain the word ABORT; clicking on that will make it disappear.

2.3.2 The System Menu

One important and useful menu is the *system menu*. It has a repertoire of operations that mostly have to do with windows. You can almost always conjure up a system menu by pointing at some place on the screen and clicking Double Right. The system menu will appear at the spot you pointed to. The mouse cursor appears on the system menu as a little cross.

If the mouse cursor is an arrow pointing North by Northwest you can get the system menu by clicking either Right Single or Right Double.

To select an operation from the system menu, point at it with the mouse and click Left or Right. If you call up the system menu by mistake, as with most menus you can dismiss it without selecting anything by simply moving the mouse far off the menu. This also works for most other kinds of pop-up menus.

This subsection describes what the various options on the system menu do. Since this is designed as a quick reference, unfamiliar vocabulary may appear. If you don't understand something, it is probably explained elsewhere in this document.

Create This allows you to create a new window. The system will ask you what kind of window you want and where to put it on the screen. See section 2.3.6, page 13 for the details.

Select The system menu is replaced by a menu whose options are all the windows currently selectable. When you pick one with the mouse, that window becomes selected.

Inspect The Inspector is invoked. See chapter 5, page 27.

Trace The function-tracing system is invoked (see trace in the Lisp machine manual). First a small window appears in which you are asked for the name of the function to be traced. Then a menu of options to the trace system appears. After selecting which trace features you want, click on **Do It** to do it. If you decide you don't want to do it after all, click on **Abort**.

Split Screen This is a convenient way to divide the screen area among several windows. See section 2.3.8, page 14 for the details.

Layouts The system menu is replaced by a menu containing at least the options **Just Lisp** and **Save This**. If you choose **Just Lisp**, an idle Lisp Listener will be picked, expanded to cover the whole screen, and selected. If you choose **Save This**, the current configuration of exposed windows will be remembered. You will be prompted for a name to remember the configuration under.

All of the previously remembered layouts will appear in future Layouts menus. If you pick that option from the Layouts menu, that layout will be restored to the screen.

Edit Screen Invokes the screen editor, which allows you to move windows around in various ways. See section 2.3.12, page 16.

Other The system menu is replaced by another menu with more options on it. This menu is an extension of the system menu, and its options are documented fully in the remaining part of this table.

Arrest — Often a window has a process associated with it in some way. Pointing the mouse at a window, calling up a system menu, and clicking on "Arrest" halts the process associated with that window.

Un-Arrest — Pointing at a window, calling a system menu, and clicking on "Un-Arrest" starts the process associated with that window from where it left off, if it was stopped by mousing "Arrest" from the system menu. There are other reasons for a process to be stopped, however, and this will not undo all of them. See "Arrest" above, and also see the "Index to Function Keys" under "TERMINAL A" (page 36).

Reset — Pointing at a window, calling a system menu, and clicking on "Reset" starts the process associated with that window from scratch, re-evaluating that process' initial form. Whatever program the process had been running is thrown out of. Before the reset actually happens, a window will pop up asking you to confirm the operation; answer yes by clicking the mouse or no by moving it away from the confirmation window.

Kill — Destroys the window that the mouse was pointing to when the system menu was summoned. Before the kill actually happens, a window will pop up asking you to confirm the operation; answer yes by clicking the mouse or no by moving it away from the confirmation window.

Emergency Break

Clicking on "Emergency Break" is the same as typing "TERMINAL CALL". It gets you to the cold-load stream where there is a Lisp interpreter running that does not depend on the window system. See the "Index to Function Keys" under "TERMINAL CALL" (page 37). This function is accessible from both the keyboard and the mouse so that if you break the software for one of them you still have a chance of getting to the cold-load stream and fixing it.

Refresh — Refreshes the display on the window that the mouse was pointing to when the system menu was summoned. Useful when something dark and sinister has munged your screen.

Set Mouse Screen

The window system's jurisdiction extends not only to the main black-and-white monitor, but to any other bit-raster monitors that are connected to the Lisp Machine, such as a color monitor. However, in order to manipulate windows on another screen, the mouse must somehow be moved to that screen. If you pick the Set Mouse Screen option from the system menu, and there is more than one screen connected to the Lisp Machine, the mouse will be moved onto another screen. If you click Left on Set Mouse Screen, the system will pick another screen (this is useful when there are only two). If you click Right, the system menu will be replaced by a menu whose options are the names of the various screens. When you pick one of these screens, the mouse will move to that screen.

2.3.3 Notification

When certain asynchronous events occur, unrelated to what you are currently doing with the selected window, the system notifies you by beeping and displaying an explanatory message. Such an event might be an error in a process whose window is not exposed, an error or other attempt to type out by a "background" process which has no associated window, or an attempt to type out on a deexposed window of a kind which notifies rather than just waiting for you to expose it. The system notifies you in one of two ways, depending on what windows are currently on your screen.

One way that you can be notified is by the appearance of a message enclosed in square brackets. This method is used when the selected window is a Lisp listener, or any other type of window that accepts notifications. For instance, in the editor notifications are printed this way in the "echo area" below the mode line. If the notification informs you of a window waiting to type out or to tell you about an error incurred by some program, then you can select that window at any time by typing TERMINAL 0 S, or ESC 0 S on the old keyboards. You can return from there to your original window by typing TERMINAL S, or ESC S on the old keyboards. See the section on "Selecting a Window" (section 2.3.5, page 12) for further details.

The other way you can be notified is by the popping-up of a small window with the message displayed in it. This happens when there is no good place on the screen to print the message. In this case you point the mouse at the notification window and click the Left button, at which point the notification will disappear and the associated window which is waiting to type out (if any) will appear. Typing any normal key (typically space) will get rid of the notification and return you to the window you were in when the notification popped up. Alternatively, TERMINAL 0 S will select the interesting window, then TERMINAL S will reselect the window that you were typing at when the notification occurred. Selecting that window with the mouse by clicking Left at it also works.

If a notification pops up while you are typing, the system saves your typing in the window you were typing at before the notification popped up. After beeping, it gives you a second or two to notice and stop typing before it listens to the keyboard; at this point if you hit a key this means that you have read the notification and want it to go away now.

Notifications are saved. If you want to see old notifications again, call the function tv:print-notifications, which will print each notification with the time that it occurred. This can be useful when a notification is accidentally erased before you have had time to read it.

2.3.4 Selecting a Window

There are several ways to cause a particular window to become selected. If any part of the desired window is visible on the screen, you can select it by pointing at it with the mouse and clicking Left. If the desired window is completely invisible, you can call up a system menu and pick the "Select" option. The system menu will be replaced by a menu of all the currently selectable windows. Pick the one you want by clicking Left on it.

When a window is selected, it will become exposed if it was not exposed already. If the selected window has a standard rectangular blinker, the blinker will wake up and start to flash. The window is now fully awake and anything you type will be directed to it.

2.3.5 More Advanced Window Selection

All the currently selectable windows are arranged in a kind of stack with the selected window on top. This has no relation to the arrangement of windows on the display, but rather refers to the way the window system keeps track of selectable windows. When you select a window with the mouse, it is dredged up and put on top of the stack. The windows are thus arranged with the most recently selected ones near the top of the stack. If you type TERMINAL 1 S, the currently selected window will be moved to the bottom, and the next most recently selected window will come to the top and be selected. Repeatedly typing TERMINAL 1 S will select each of the selectable windows in turn.

Typing TERMINAL - S (or TERMINAL -1 S: the two are equivalent) will drag the window on the bottom of the stack to the top. Repeatedly typing TERMINAL - S will select each of the selectable windows in reverse order. Note that TERMINAL 1 S and TERMINAL - S do not alter the cyclic order of the selectable windows.

You can select any selectable window with some variant of the TERMINAL S command. To select the *n*th window in the stack, where the currently selected window is considered the first, type TERMINAL *n* S. This is just like selecting that window with the "Select" option to the system menu. The window in question is extracted from the stack and pushed on top.

Typing TERMINAL 2 S repeatedly flips back and forth between the two top windows on the stack. When you type TERMINAL S with no argument, the argument defaults to 2, and this is the behavior you get. Typing TERMINAL 3 S repeatedly cycles through the top three windows on the stack, and so on. If there are *k* selectable windows, giving TERMINAL S an argument larger than *k* is the same as giving an argument of *k*, which is the same as giving an argument of -1; it brings up the "oldest" window.

Giving TERMINAL S a negative argument (other than -1, which was discussed above) of -*k* takes the currently selected window and stashes it in the *k*th slot down, bringing the *k*-1 windows beneath it up. The window that was in second position becomes Top Dog and is selected. Repeated *k* times, this cycles through the top *k* windows on the stack in reverse order. This is exactly the inverse of a positive argument. TERMINAL *n* S and TERMINAL -*n* S undo each other.

There is also a way to select a window that is trying to talk to you. When a deexposed window has a process doing something interesting in it, such as waiting to type out or waiting to tell you about an error it encountered, you can select it by typing TERMINAL 0 S. When a window goes into such a condition, it sends you a notification (see section 2.3.3, page 11). When there is no window in such a condition, TERMINAL 0 S does nothing. When there is more than one such window, the first one found in the stack will be selected. Repeatedly typing TERMINAL 0 S will get all of them.

2.3.6 Creating New Windows

Starting up new Editors, Lisp Listeners, Supdups, and so on, is done by creating new windows of the appropriate type. This section explains one way to create new windows of various types.

Call up a system menu and pick the "Create" option. The system menu will be replaced by a menu of window types to create. At present there are six kinds of windows on this menu: Supdup, Telnet, Lisp, Lisp (Edit), Edit, and Peek. There is also Any, which allows you to type in the flavor of window you want from the keyboard. User-defined windows may be added to this menu through the variable tv:default-window-types-item-list, so you may see more choices in the menu than those listed here. The various window types are explained in other sections of this document.

Click on the type of window you wish to create. The menu will vanish and the mouse blinker will change into an upper left corner bracket. With this corner bracket, point to the spot on the screen where you want the upper left corner of the new window to be and click Left. The bracket will freeze on that spot in order to mark it, and the mouse blinker will change into a lower right corner bracket. Use this bracket in the same way to define the lower right corner of the new window. The new window will take shape between the corners thus delimited.

To make the new window occupy the whole screen, simply place the lower right corner above or to the left of the upper left corner.

When you are giving the system a rectangle with the mouse in this way, clicking Left will place the bracket where it is now, while clicking Right will place it at the nearest "suitable" place. The exact definition of "suitable" is complicated, but it tries to put it at a nearby window edge or corner, if one is close enough. Note that the bracket will only move in the direction it points, thus you point at a corner of a rectangle from *inside* the rectangle.

Usually clicking Middle will abort the whole operation. Click Middle if you decide you don't want to create a window after all.

Whenever you create a new window, it is immediately selected, and pushed onto the stack of selectable windows. Usually the new window will stay around on that stack as a selectable window until it is explicitly destroyed.

Another way that windows get created is explained in the next section.

2.3.7 Selecting a System

The SYSTEM function key can be used to find a window of a particular type, and if one does not exist, to create one.

The SYSTEM key should be followed by one of these code letters:

E	Editor.
I	Inspector.
L	Lisp Listener.

M	Mail-reading system.
P	Peek.
S	Supdup.
T	Telnet.

When you type SYSTEM followed by one of these letters, the stack of selectable windows is scanned, from the top down, for a window of the specified type. As soon as one is found, it is selected and moved to the top of the stack. If there are no windows of the specified type, one is created.

If you hold down CTRL while typing the code letter, a new window of that type will be created even if one already exists.

In the event that the currently selected window is itself of the specified type, it is moved to the bottom of the stack before the scan begins. Typing SYSTEM L repeatedly, for example, cycles through all selectable Lisp Listener windows. If there is only one window of the specified type, and it is current, it remains current and the system beeps to tell you that you probably goofed.

Invoking the function ed from a Lisp Listener is almost exactly the same as typing SYSTEM E. The same Editor window gets selected in both cases. One minor difference is that the next time that particular Lisp Listener window is selected, the first thing that will happen is that the ed function will return t. (Actually, the ed function returns t right away, but the value can't be typed out until the Lisp listener window is exposed again.)

Typing CTRL/Z to an Editor is not the same as typing SYSTEM L. SYSTEM L gets you to the most-recently selected Lisp Listener, while CTRL/Z gets you to the last place you ran the ed function.

Additional code letters for the SYSTEM key can be added through the Split Screen command in the system menu, or via the variable tv:*system-keys*. Typing SYSTEM HELP will always tell you all the available choices.

2.3.8 Splitting the Screen

The Split Screen option of the system menu is a convenient way to divide the screen area among several windows. If you select Split Screen, the system menu is replaced by a menu whose options are detailed below. By selecting items from this menu, the user specifies a set of windows that are to share the screen. Typically these are newly-created windows, but there are also options to incorporate existing windows into a split screen arrangement. As the specification proceeds, a small diagram of the proposed display appears next to the Split Screen menu and is updated as you make selections.

Supdup	Incorporate a new Supdup window into the split-screen layout. See chapter 8, page 34.
Telnet	Incorporate a new Telnet window into the split-screen layout. See chapter 8, page 34.

Lisp Incorporate a new Lisp Listener window into the split-screen layout. See chapter 3, page 19.

Lisp (Edit) Incorporate a new Editing Lisp Listener window into the split-screen layout. See chapter 3, page 19.

Edit Incorporate a new Editor window into the split-screen layout. See chapter 4, page 21.

Peek Incorporate a new Peek window into the split-screen layout. See chapter 7, page 32.

Any Will ask what flavor of new window you want.

Existing Window
A menu will pop up containing the names of all the selectable windows. Pick one with the mouse, and it will be incorporated into the split-screen layout.

Existing Lisp An idle Lisp Listener will be chosen and incorporated into the split-screen layout. Since it doesn't matter which one is incorporated, the user is not asked to pick one.

Plain Window Incorporate a new window with no interesting features at all.

Trace & Error Incorporate a new window which will be used for output from the trace package and interaction with the error-handler. This allows you to use those debugging facilities without interfering with the window in which you are running your program, which might contain a graphic display, for example.

Trace Incorporate a new window which receives just trace output.

Error Incorporate a new window which is used just for error-handler interaction.

Frame The default is to split the screen simply by creating windows with the appropriate sizes, shapes, and positions. If you select the Frame option, however, the windows in the split-screen layout will be bound together as inferiors of a frame (see section 2.3.10, page 16). The main effect of this is that they will be exposed and de-exposed together; selecting one of the windows, for instance with the Select operation in the system menu, will expose the whole set.

Selecting Frame pops up an additional window which lets you specify parameters for the frame: whether it should exist (so you can turn it off if you decide you don't want it), its name, and a key which can be typed after the SYSTEM key to select it. You can change one of these parameters by pointing the mouse at it, so that a box appears around it, and clicking.

Mouse Corners
Allows you to use the mouse to point to the upper-left and lower-right corners of the screen area to be divided among the windows you select, just as in the Create operation (see page 13). If you don't use Mouse Corners, the entire screen will be split.

Undo Remove the last window you added to the layout you are building up.

Do It Create all the windows that need to be created, and assemble them with the other specified windows into the split-screen layout. Expose the entire layout, and select the first selectable window that was specified.

Abort — Flush the proposed layout and go back to the previously selected window. Use this if you decide you didn't really want to split the screen after all.

2.3.9 Destroying a Window

To destroy a window, point at it with the mouse and click Double Right. This calls up a system menu. Pick the "Other" option. The system menu will be replaced with another menu containing additional options. This "Other" menu is in some sense the second page of the system menu. To kill the window you indicated, choose the "Kill" option from this auxiliary menu. You will be asked to confirm that you really wanted to kill that window by the popping-up of a small confirmation window. To answer yes, click the mouse on the confirmation window. To answer no, move the mouse away from the window so that it disappears.

2.3.10 Frames

Sometimes windows are grouped into *frames*. A frame is a window which acts like a screen; it can have several windows displayed on it. The purpose of frames is to group related windows together so that they can be manipulated as a unit, while still keeping them separate so that each window can do a different thing.

Programs that have a display made up of several windows always group them into a frame. The editor, the inspector, and the window error-handler are examples.

2.3.11 Invoking the Inspector

There are three ways to enter the Inspector system. You may type SYSTEM I, the function inspect may be invoked from a Lisp Listener, or the "Inspect" option may be chosen from the system menu. For full documentation of the inspector, see chapter 5, page 27.

2.3.12 The Screen Editor

The screen editor is a mouse-controlled program for manipulating the layout of your screen. It can be used to move windows around, to change the size and shape of windows, and to change which windows are displayed.

The screen editor is gotten from the system menu by clicking on Edit Screen. Ordinarily it will enter the screen editor immediately, editing the whole screen. However, if you select Edit Screen by clicking Right (indicating you want the hairier version of the command), and the window that the mouse is in is a frame, you have the option of editing that frame or the whole screen; another pop-up menu will appear asking you to choose.

The screen editor works by displaying a menu of commands. You select a command by clicking on it with the mouse and the menu disappears. If you need to point to any windows, edges, or corners needed as arguments to the command, the screen-editor will prompt you with a message in the mouse-documentation line above the who-line and will change the shape of the mouse cursor. When you are to point to a window, the mouse cursor changes to a small reticule, which should be positioned over the desired window.

When the screen editor is asking you to point to something, clicking Left will select what the mouse is pointing at. Clicking either of the other two buttons will abort the current command and bring back the screen editor's command menu.

After executing the command, the screen editor's menu will appear again. To exit the screen editor, choose "Exit".

This is a summary of the options in the Edit Screen menu.

Bury Bury the specified window, deexposing it and allowing whatever display it was obscuring to be seen.

Expose Completely expose the specified window. This is used for displaying windows that have been partially obscured by others.

Expose (menu) Like Expose but pops up a menu of all active deexposed windows. This is good for exposing a window which you can't point to because you can't see any part of it.

Create Create a new window inferior to the frame or screen you are editing. This is just like the Create operation in the system menu (see section 2.3.6, page 13), except that if you are editing a frame the choice of window types to create may be different, or there may be no choice at all. For instance, when screen-editing an editor frame, you are only allowed to create editor windows.

Create (expand) This is the same as Create, except that instead of prompting you for an upper-left corner and a lower-right corner, it only prompts you for a single point. The window is created in such a way that it occupies any unused area surrounding that point. An unused area is any part of the screen that doesn't contain an exposed window; it may contain nothing (blank) or it may contain a visible portion of a deexposed window.

Kill Destroy the specified window. Before the kill actually happens, a window will pop up asking you to confirm the operation; answer yes by clicking the mouse or no by moving it away from the confirmation window.

Exit Exit the Screen Editor.

Undo Attempt to reverse the last Edit Screen operation. Kill cannot be undone. Undoing Create simply buries the window rather than killing it. Undo can itself be undone.

Move Window Keeping its size and shape constant, move the specified window to another place on the display. After you pick a window, a rectangle the size and shape of the window will follow the mouse around. Click Left to move the window to where the rectangle is; click Middle or Right to abort and leave the window where it is. If the window cannot be moved to where you tell it, you will get a beep.

Reshape The user is asked to pick a window, and then is prompted to reposition the upper left and lower right corners of that window (see page 13).

Move Multiple The mouse blinker becomes a Move blinker which is used for pointing at exposed corners and edges of windows. (See Move Single.) Clicking Left at a feature adds that feature to a list of things to be moved. Features that are on the list are

highlighted. If a feature is already a member of the list, it is removed when you click at it.

When you have selected the features you wish to move, click Right (meaning "Do It"). All of the features are moved together, and nailed down in a new location at the next Left or Right click.

Clicking Left Long at a feature is the same as clicking Left, but it also commences the move.

If you want to abort the operation, either while selecting features or while moving them, click Middle. This returns to the screen editor's command menu and leaves all the features where they were originally.

When one of a set of coincident edges or corners is picked to be added to the list, the rest are also added. This facilitates the rearrangement of groups of adjacent windows. If the user does not desire to do this, the extra features can be deleted from the list of things to be moved by clicking Left at them in the usual way.

Move Single The mouse blinker becomes a Move blinker which is used for pointing at exposed corners and edges of windows. The Move blinker is a large arrow which points at the feature in question. It always points at features from inside the window with which that feature is associated. It has two states: pointing at an edge and pointing at a corner. When pointing at an edge, it points steadily in a direction perpendicular to that edge. When pointing at a corner, the arrow rotates smoothly as it is moved, so that it continues to point directly at the corner.

Using the Move blinker, select a feature by clicking Left at it. The feature will be highlighted in boldface, and the Move blinker will vanish. The feature will now follow the mouse to a new position. Another click Left fixes it in the new position. Clicking Middle aborts the move and leaves the feature where it was originally.

Expand Window

Reshape the specified window so that it occupies any unused area surrounding it. An unused area is any part of the screen that doesn't contain an exposed window; it may contain nothing (blank) or it may contain a visible portion of a deexposed window.

Expand All Reshape all the currently exposed windows so that together they occupy as much of the screen as possible, subject to the restriction that no window gives up any of its former territory (all its edges move outward if at all). There are several possible algorithms; the one implemented seems to do the right thing most of the time, and is stable with respect to itself, so that if Expand All is called twice in succession, the second invocation does nothing.

3. The Lisp Listener

A Lisp Listener window allows you to talk to the Lisp Machine Lisp interpreter. When you cold-boot a Lisp Machine, a newly-created Lisp Listener window is the first thing you see, filling the whole display except for the who-line at the bottom. When you type to a new Lisp Listener, you are typing to a read-eval-print loop of the usual kind. Most simple user programs are run from a Lisp Listener.

The mouse always points North by Northwest when it is pointing at a Lisp Listener window. This means that the system menu can be summoned by a single click Right and there are no unusual functions on the mouse buttons.

All of the interesting things that can be done from a Lisp Listener are done by typing Lisp forms. Consult the Lisp Machine Manual for further details.

3.1 The Editing Lisp Listener

The Editing Lisp Listener is a Lisp listener which allows you to edit your input with all the commands of the editor, rather than just Rubout. (The editor is discussed in the next chapter.) The Lisp (Edit) option of the Create system-menu command is used to create an editing Lisp listener (see section 2.3.6, page 13). If you type Lisp forms at this window, the forms will be evaluated and the value will be printed, exactly like the way a regular Lisp Listener works.

If you mistype a character, you can rub it out exactly as you would in an ordinary Lisp Listener. However, you may also use all the correcting features of the Editor: if you mistype a form, you can edit it with the editor functions, inserting characters, deleting words and forms, exchanging characters, words, and forms, and so on. You can go back and retrieve earlier elements (input *or* output) of your conversation with the interpreter and insert them into the form you are currently typing.

Each time a complete form is read, it is put into the editor's kill ring. This means that the CTRL/Y command will retrieve the last form. You may edit it and then evaluate the resulting modified form. As usual, previous forms may be retrieved by giving CTRL/Y a numeric argument or by typing META/Y.

You can also use the specific features of Lisp Mode, such as automatic indentation, printing of functions' argument names (CTRL/SHIFT/A), and parenthesis matching.

Input is complete and evaluation takes place whenever you finish typing a complete Lisp form, or a complete expression of whatever syntax is accepted by the program you are typing at. To "finish typing" is to type a printing character when the cursor is at the end of the buffer (not in the middle of editing something) and that character completes the expression (it might be a close parenthesis).

When you edit your input you may complete an expression by a means other than finishing typing it. For instance, you may delete an unmatched open parenthesis, causing a close parenthesis at the end of the buffer to complete an expression when it did not before. In this case input does not automatically complete, since you might not be finished editing. You might create a syntactically-complete S-expression temporarily while editing a larger expression, and you

would not want the Lisp listener to go off half-cocked and read it. For instance, you might yank in a complete expression with CTRL/Y, intending to edit it before sending it off to the reader.

Once you *are* done editing, you tell the editor that you are done with the END key (or CTRL/RETURN). The exact details depend on whether there is a region. If there is not, the cursor simply jumps to the end of the buffer and the characters that have not yet been read are released to the Lisp listener for reading.

If there is a region, it is copied to the end of the buffer, and the editing Lisp listener pretends that you had just typed it in. If you had been in the middle of typing a form at the end of the buffer, that old form is flushed and replaced by the region you specified. If the region does not contain a complete form, it is left for you to finish it after it has been copied to the end of the buffer. If the region contains exactly one complete form, it will be copied to the end of the buffer and evaluated. If the region contains several forms, they will be moved to the end of the buffer and evaluated one at a time, so that the results of the evaluations will be inserted between the forms.

All of the Lisp Mode commands for getting lines, forms, and DEFUNs into the current region are useful in conjunction with the END key.

If you use an editor command that requires the echo area, for example CTRL/S (Search), an echo area will temporarily pop up over the bottom several lines of the window.

You may save a transcript of part or all of your session in a file by marking the desired portion as a region and then issuing the META/X Write Region command.

All editors share the same kill ring, so text can be moved from one editing Lisp listener to another, or between an editing Lisp listener and the ZMacs editor, by putting it in the kill ring, selecting another window, and yanking it back.

4. The Editor

When an Editor window is selected, you are talking to ZMacs, the Lisp Machine's display-oriented editor. ZMacs is an extremely sophisticated program with hundreds of useful features. The beginning user can pretend that ZMacs is Emacs (the pdp-10 display editor) and be right most of the time.

The Lisp function ed selects an Editor window. One (at least) has already been created by default when you cold-boot the machine. Other ways to select an Editor window include typing SYSTEM E and calling up a system menu and explicitly selecting an Editor.

The Editor is an extremely large and rich system, and should get a document all its own. However, we discuss here some aspects of using the mouse in the Editor, so as to have most of the mouse functions under one roof.

4.1 Using the Mouse in the Editor

4.1.1 The Mouse and the Cursor

When the mouse is pointing to the central portion of an exposed Editor window, the mouse blinker takes the form of an arrow pointing North by Northeast. When the mouse cursor has this shape, there is always a character nearby that is flashing. A blank space or newline is flashed by blinking an underline underneath it. This is the character the mouse is pointing to.

You can bring the editor cursor to the mouse by clicking Left; alternatively you can bring the mouse to the cursor by clicking Left Double.

If you click Left Hold, not only will the cursor come to the moused character, but a mark will be set there as soon as the mouse is moved off the original character. As long as the left mouse button is held down, the cursor will continue to follow the mouse, but the mark will remain where it was placed. This enables the user to delimit an arbitrary region, so that the region-manipulation commands can be used. The current region is marked by underlining, so that you can see its exact scope.

If you click Left Double Hold (like a Morse Code "A": dit-dah), the mouse will move to the cursor and grab it, so that when the mouse moves, the cursor will move with it. This does not set any mark, but merely allows you to slide the cursor around the buffer. When you release the mouse button, the mouse will let go of the blinker.

4.1.2 The Mouse and the Region

Very often you will want to select some particular word, line, sentence, or Lisp form by setting the region around it. The middle mouse button is very useful for manipulating the region. Clicking Middle sets the region in various ways depending on what the mouse is pointing at.

To select a word, point to one of the letters of the word and click Middle. The Editor will always try to include a space in the region with the specified word. If it has a choice, the space to the right of the word will be included, but sometimes it will be forced to choose the space to the left, because the word appears just before some punctuation mark. If for some reason you especially want the space on the left included, rather than the space on the right, point to that space when you click Middle, rather than to a letter of the word.

To select a line, point at the end of the line and click Middle. To select a sentence, point to the period at the end of the sentence and click Middle. Clicking Middle on either of a pair of balanced parentheses will select all the text between (and including) them.

The only difference between clicking Middle Hold and clicking Middle is that the region moves with the mouse as long as the middle button is held down. The region chases the mouse, by turns encompassing words, sentences, lines, or parenthesized text, depending on where the mouse is pointing at any particular moment. When the region has been properly placed, release the middle button.

Clicking Middle Double performs some of the operations normally associated with CTRL/W, META/W, CTRL/Y, and META/Y. Its exact behavior is fairly involved, and depends on whether or not there is a region and on what the last command was.

If there is no region, the first Middle Double acts like a CTRL/Y and successive ones act like META/Y, trying to insert at the current cursor position all the pieces of text which have recently been deleted or saved, starting with the most recent and progressing backwards with each Middle Double click.

If there is a region, the first Middle Double acts like META/W. This has the effect of pushing the region onto the kill ring. At this point, the underline showing where the region is disappears, showing that there is now no region. The next Middle Double click deletes the text that had been underlined, like CTRL/W. From then on, Middle Double behaves as described in the preceding paragraph.

4.1.3 The Editor Menu

Clicking Right Double in an Editor window calls up the system menu, as it does almost everywhere.

Clicking Right (Single), however, calls up a menu of useful Editor operations. If you call up this menu by mistake, you can get rid of it simply by moving the mouse far off the menu. Below are listed the options offered by the Editor menu, and what they do when you click Left or Right on them.

Arglist All the function names in the Editor window become mouse sensitive, and the mouse cursor becomes an arrow pointing North. Clicking Left on the name of a function causes the argument list from that function's original definition to be

displayed below the mode line. You can also type the name of the function whose argument list you want to see.

Edit Definition Prompts you to point to or type a function name, just like the Arglist option, but actually reads the source file into an editor buffer, finds the definition of the function, and allows you to edit the code.

List Callers Prompts you to point to or type a function name, just like the Arglist option, and scans the current package for functions that call the specified function. The names of these functions are listed, and CTRL/. can be used to edit their definitions.

List Functions Prompts you for the name of an Editor buffer, and lists the names of all the functions defined in that buffer. The function names are mouse-sensitive; clicking Left jumps the editor cursor to that function definition. Clicking Right gives a menu of possible operations.

List Buffers Lists all your Editor buffers. The buffer names are mouse-sensitive; clicking Left selects that buffer, clicking Right gives a menu of things to do to that buffer.

Kill Or Save Buffers
Displays a window with one line for each editor buffer. To the right of each buffer name are three choice-boxes, labelled *Save*, *Kill*, and *UnMod*. Clicking the mouse on a choice-box selects that choice and puts an "X" in the box, or gets rid of the "X" if there is one already there. An "X" next to *Save* writes out the file, next to *Kill* kills the buffer, and next to *UnMod* marks the buffer as not needing saving, like the META/~ command. Note that all the buffers that need to be saved have an "X" in their *Save* box initially. At the bottom are two choice-boxes marked *Do It* and *Abort*. Clicking on *Do It* will go ahead and do the file-saving, buffer-killing, and buffer-unmoding specified by the choice-boxes you have marked. Clicking on *Abort* will forget the whole thing.

Split Screen This is just like the Split Screen option of the system menu (see section 2.3.8, page 14), but the options in this case are the names of all the Editor buffers, "New Buffer", "Find File", "Do It", and "Abort". "New Buffer" prompts for a buffer name and "Find File" prompts for a file name.

Compile Defun
Compiles the function the cursor is in. This is the same as CTRL/SHIFT/C.

Indent Region Does the same thing as the TAB key on each line of the region. In Lisp mode this corrects the indentation of all the lines.

Change Default Font
Changes the font of typed-in text. This is the same as CTRL/META/J. When prompted "Font ID:" you may type HELP for help.

Change Font Region
Changes the font of the text in the region. This is the same as CTRL/X CTRL/J. When prompted "Font ID:" you may type HELP for help.

Uppercase Region
Makes the alphabetic characters in the region all upper case.

Lowercase Region
Makes the alphabetic characters in the region all lower case.

Mouse Indent Rigidly

Clicking this and holding down the button will allow you to change the indentation of all the lines in the region. Moving the mouse to left or right while holding the button down decreases or increases the amount of indentation.

Mouse Indent Under

Indents the current line to a specified place. You may point with the mouse to the column where you want it indented and click Left. Or you may type a string on the keyboard; the line will be indented to line up with the beginning of the most recent occurrence of that string previously in the buffer.

4.1.4 Scrolling

The Editor window really functions as a "window" through which you can see a part of a larger object, the Editor buffer. Conceptually, the buffer continues invisibly above and below the actual window.

The Editor window, and some other windows which display only a part of their conceptual contents, have a feature called *scrolling* which allows the user to move the window's view so as to display any part of these contents.

The scrolling feature is really two distinct features, one for scanning slowly through the buffer and one for jumping around rapidly.

Toward the right side of the upper and lower edge of the Editor window is the *scrolling zone*. If you try to move the mouse out of the window across this part of the window's boundary, the mouse will not leave the window, but its cursor will change into a very fat arrow pointing at the edge you tried to go over. For example, if you tried to move the mouse out of the Editor window through the rightmost part of the upper edge, the mouse cursor would not leave the window, but would change into a fat arrow pointing up. If you now continued to move the mouse gently upwards, the mouse cursor would *not* move, but the text in the window would slide *down*. New text would appear at the top, and the text moving off the bottom edge of the window would be lost to view. When you pull the mouse away from this edge, the mouse cursor changes back to its normal Editor shape.

The rightmost part of the bottom edge of the window behaves the same way, but of course the fat arrow points down, and the text moves up as you scan downward.

The other scrolling feature is the *scroll bar*. The left edge of the Editor window resists violation in the same way as the scrolling zones do, but in this case the mouse cursor changes to a double-headed arrow pointing up and down. In addition, a thin line appears somewhere along the left edge, just inside the border and parallel to it. The length of this line and its position with respect to the entire left edge are in proportion to the length and position of the Editor window with respect to the entire Editor buffer.

See section 10.1, page 40 for more information on scrolling.

4.2 Ztop mode

A Ztop buffer is a hybrid between a Lisp listener and the ZMacs editor. It is an interactive alternative to more traditional programming environments. Traditionally, the Lisp programmer shifts his attention back and forth between an Editor and a Lisp Listener. With Ztop, the programmer has the power to manipulate his own conversation with the Lisp interpreter with all the flexibility that the editor provides. By using multiple buffers, the program source file can be in one buffer and the interaction can be in another buffer.

Ztop mode is very similar to the editing Lisp listener described above. The difference is that a Ztop buffer is an editor buffer, rather than a separate window, and therefore interacts more closely with the editor. It is largely a matter of personal preference which you use.

To enter Ztop mode, call up an Editor and get a new, empty buffer. Then invoke the extended command "Ztop Mode". The mode line will change to reflect the fact that this buffer is now in Ztop mode. If you type Lisp forms at Ztop, the forms will be evaluated and the value will be printed, exactly like the way a regular Lisp Listener works.

If you mistype a character, you can rub it out exactly as you would in a Lisp Listener. But Ztop has all the correcting features of the Editor: if you mistype a form, you can edit it with the editor functions, inserting characters, deleting words and forms, exchanging characters, words and forms, and so on. You can go back and retrieve earlier elements of your conversation with the interpreter and insert them into the form you are currently typing. Whenever you finish typing a form, Ztop will evaluate it and print the value.

You can also use the specific features of Lisp Mode, such as automatic indentation, printing of functions' argument names, and parenthesis matching.

4.2.1 The I-Beam

Sometimes Ztop will display a large slowly-blinking I-beam between two characters of the text. This is to show you how much of the typein has been read so far. Editing you do before the I-beam will not affect what the Lisp listener reads, while editing after the I-beam will. The I-Beam appears whenever you are actually editing rather than simply typing in, except that the I-Beam is suppressed if it would come out on top of the cursor.

Input is complete and evaluation takes place whenever you finish typing a complete S-expression (assuming you are talking to the normal read-eval-print loop.) However, when there is an I-Beam you may complete an S-expression by a means other than finishing typing it. For instance, you may delete an unmatched open parenthesis, causing a close parenthesis at the end of the buffer to complete an S-expression when it did not before. In this case input does not automatically complete, since you might not be done editing. You might create a syntactically-complete S-expression temporarily while editing a larger S-expression, and you would not want the Lisp listener to go off half-cocked and read it. For instance, you might yank in a complete S-expression with CTRL/Y, intending to edit it before sending it off to the reader.

Once you *are* done editing, you tell the editor that you are done with the END key (or CTRL/RETURN); the I-Beam is a reminder that you need to do this. The END key is used the same way as in the editing Lisp listener (see page 20).

4.2.2 Leaving Ztop

You can leave Ztop by going to another Editor buffer, or by leaving the Editor. In these cases, the Ztop buffer will be left just as it was and you can re-enter it and continue your session at any time. The extended command "Select Last Ztop Buffer" can be used to do just that from another editor buffer.

Alternatively, you could leave Ztop by changing your mode to Lisp or Text. This would end the Ztop session, and you could then edit your buffer normally. Of course, you can use the file-writing commands to write out your buffer at any time, whether you are in Ztop mode or not.

5. The Inspector

The Inspector is a window-oriented program for poking around in data structures. When you ask to inspect a particular object, its components are displayed. What the "components" are depends 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 objects displayed on the screen by the Inspector are mouse-sensitive; when you point the mouse at an object a box appears around it. Clicking a mouse button will do something to that object, such as inspecting it, modifying it, or giving it as the argument to a function.

The following documentation is on the current version. As this is still an evolving program, it may change without notice.

The Inspector can be part of another program, such as the Window Error-Handler, or it can be used "stand-alone". The stand-alone Inspector can be entered via the Inspect command in the system menu, or by the inspect function which inspects its argument, if any.

The stand-alone Inspector is a frame consisting of a small interaction window on the top, a history window and menu immediately below that, followed by some number of inspection windows (three by default). Each inspection window can inspect a different object. When you inspect an object it appears in the large window at the bottom, and the previously inspected objects shift upward.

Other programs, such as the Window Error-Handler, may utilize inspection and history windows, and though the display will look the same, the handling of mouse buttons may depend upon the particular program being run. The discussion below focuses primarily on the stand-alone Inspector.

The history window maintains a list of all objects that have been inspected. It is usually kept in LRU order—the least recently displayed object is at the top, and the most recently displayed object is at the bottom. Any mouse sensitive object in the history window can be inspected by clicking on it. There is a "line region" at the left hand side of the history window, which allows other operations. In the line region the mouse cursor changes to a rightward-pointing arrow. Clicking Left in the line region inspects the object. This is sometimes useful when the object is a list and it is inconvenient to position the mouse at the open paren. Clicking Middle deletes the object from the history. The history window has a scroll bar at the far left, as well as scrolling zones in the middle of its top and bottom edges. The last three lines of the history are always the objects being inspected in the three inspection windows.

The history window also maintains a cache allowing quick redisplay of previously displayed objects. This means that merely reinspecting an object will not reflect any changes in its state. Clicking Middle in the line region deletes the object from the cache as well as deleting it from the history window. The DeCache command in the menu may be used to clear everything from the cache.

At the top of an inspection window is a label, which is the printed representation of the object being inspected in that window, or else the words "a list" meaning that a list is being inspected. The main body of an inspection window is a display of the components of the object, labelled with their names if any. This display may be scrolled using the scroll bar on the left or

the "more above" and "more below" scrolling zones at the top and bottom.

Clicking on any mouse sensitive object in an inspection window inspects that object. The three mouse buttons have different meanings, though. Clicking Left inspects the object in the bottom window, pushing the previous objects up. Clicking Middle inspects the object, but leaves the source (namely, the object being inspected in the window in which the mouse was clicked) in the second window from the bottom. Clicking Right tries to find and inspect the function associated with the selected object (e.g. the function binding if a symbol was selected).

The inspection display that is chosen depends upon the type of the object:

Symbol — The name, value, function, property list, and package of the symbol are displayed. All but the name and package are modifiable.

List — The list is displayed ground by the system grinder. Any piece of substructure is selectable, and any "car" or atom in the list can be modified.

Instance — The flavor of the instance, the method table, and the names and values of the instance-variable slots are displayed. The instance-variables are modifiable.

Closure, Entity — The function, and the names and values of the closed variables are displayed. For an entity the type or class is displayed as well. The values of the closed variables are modifiable.

Named structure — The names and values of the slots are displayed. The values are modifiable.

Array — The leader of the array is displayed if present. For one dimensional arrays, the elements of the array are also displayed. The elements are modifiable.

FEF — The disassembled code is displayed.

Select Method — The keyword/function pairs are shown, in alphabetical order by keyword. The function associated with a keyword is settable via the keyword.

Stack Frame — This is a special internal type that is used by the window error handler. It is displayed as either interpreted code (a list), or as a FEF with an arrow pointing to the next instruction to be executed.

The interaction window is used to prompt the user and to receive input. If the user is not being asked a question, then a read-eval-inspect loop is active. Forms typed will be echoed in the interaction window and evaluated. The result is not printed, but inspected instead. When the user is prompted for input, usually due to having invoked a menu operation, any input being typed at the read-eval-inspect loop is saved away and erased from the interaction window. When the interaction with the user is over, the input is re-echoed and the user may continue to type the form.

Some special keyboard characters are recognized when not in the middle of typing in a form.

Control-Z — Exits and deactivates the Inspector.

Break — Runs a break loop in the typeout window of the lower inspection pane.

Quote — Reads a form, evaluates it, and prints the result instead of inspecting it.

The menu is for infrequently used but useful commands:

Exit	Equivalent to Control-Z. Exits the Inspector and deactivates the frame.
Return	Similar to Exit, but allows selection of an object to be returned as the value of the call to inspect.
Modify	Allows simple editing of objects. Selecting Modify changes the mouse sensitivity of items on the screen to only include fields that are modifiable. In the typical case of named slots, the names are the sensitive parts. When the field to modify has been selected, a new value may be specified either by typing a form to be evaluated or by selecting any normally mouse selectable object with the mouse. The object being modified is redisplayed. Clicking Right at any time aborts the modification.
DeCache	Flushes all knowledge about the insides of previously displayed objects and redisplays the currently displayed objects.
Clear	Clears out the history, the cache, and all the inspection windows.

6. The Debugger

6.1 The Error-Handler

Refer to section 26.2 of the Lisp Machine Manual, for now.

6.2 The Window Error-Handler

This section describes the window oriented error handler, which can be gotten from the standard keyboard error handler by typing Control-Meta-W to the → prompt.

The error handler window is divided into seven panes. At the bottom is a lisp window, which ordinarily provides a read-eval-print loop, similar to the regular keyboard error handler. More commands are available by using the mouse in the other windows as described below.

At the top is a display of the disassembled or ground code for the currently selected stack frame, depending on whether or not it is compiled. This is an inspection window. The window immediately below this is a history window as in the Inspector (see below). Clicking on any mouse sensitive item with the right button in either of these windows inspects it, and clicking with the middle button stuffs it into the lisp window, echoing it and making it the value of *.

Next are the args and locals windows, side by side, displaying the names and values of the arguments to the current stack frame and its local variables and special-variable bindings. These windows are grayed out if the frame has no variables of the corresponding type. They also have a scroll bar. Clicking the mouse on the name of a variable will print the name and the value in the lisp window. Clicking on just the value will print it in the lisp window. The mouse will highlight any relevant quantity that you are pointing to.

When something is printed in the Lisp window by pointing at it with the mouse, the variable * is set to the value printed, and the variable + is set to a locative to the stack slot containing the value, or (in the case of a special-variable binding) is set to the symbol. You can use * to feed the value to any Lisp function, and + to alter the value.

Next is the stack window, which displays in a pseudo-list format the functions and arguments on the stack. Clicking on a function or argument or a sublist of one will cause it to be printed in the lisp window as in the argument or local windows. Also, clicking the mouse to the left of a line containing a particular stack frame (when the cursor is a right-pointing arrow) will make the error handler select that frame, changing what the code, arguments, and locals windows show.

Below this, and above the lisp window, is the command menu for the error handler. The available commands and what they do are:

What error Reprints the error message for the current error, in the lisp window.

Exit Window EH
Returns to the regular error handler.

Abort Program
Like ABORT or CTRL/Z in the keyboard error handler. Throws back to command-level of the erring program.

Arglist	Asks for the name of a function, which can be typed on the keyboard, or moused if it is on the screen. Picking an actor or a closure will ask for the message name to that actor and print the arguments to its method for that message. Picking a line of a stack frame from the stack window will try to align the printout of the arguments with what value was supplied in that position in that frame.
Inspect	Asks for an object which can be pointed-to with the mouse or typed in, and inspects it in the upper window. See chapter 5, page 27 for information on the Inspector.
Edit	Reads the name of a function in the same fashion as the Arglist command and invokes the editor on that function. The editor will find the source code of the function, reading in the file if necessary, and let you edit it.
Retry	Attempts to restart the current frame, like the Control-Meta-R command.
Return a value	Asks for a value (which can be selected with the mouse) and returns it from the current frame, like Control-R.
Continue	Like Control-C, except that the mouse can be used to select the object it asks you for in order to continue.
Set arg	Select an argument or local with the mouse and type or mouse a new value to be substituted in.
Search	Like the Control-S command, except that the mouse can be used.
Throw	Like Control-T, it asks for a tag and a value and throws there; the mouse of course can be used.
T NIL	Clicking one of these supplies that symbol as an argument or value for other commands. These can also be used to answer yes-or-no questions.

7. Peek

The Peek program gives a dynamic display of various kinds of system status. When you start up a Peek, this table is displayed:

P	Active processes
M	Memory usage by area
C	Chaosnet connections
A	Areas
H	Hostat
%	Statistics counters
F	File system display
W	Window hierarchy
Q	Quit
nZ	Set sleep time between updates
?	Prints this message

Under this table is a message reading "Type any character to flush:"

The left column shows a repertoire of commands to Peek. The commands A, C, F, M, P, W, and % each place you in a different Peek subsystem, to examine the status of different aspects of the Lisp Machine system. The various subsystems are described in the sections that follow.

The commands H and ? do not place you in full subsystems, but merely display some information at the top of the Peek window, followed by the "Type any character to flush:" message. Typing a space restores you to the subsystem that you were in before you typed H or ?. Typing a subsystem command places you in that subsystem, as usual. The ? command displays the table described above. The HELP key does the same.

The Q command exits Peek and returns you to the window from which Peek was invoked.

Most of the subsystems are dynamic: they update some part of the displayed status periodically. The time interval between updates can be set using the Z command. Typing nZ, where n is some number, sets the inter-update time interval to n sixtieths of a second. Using the Z command does not otherwise affect the subsystem that is running.

Some of the items displayed in the subsystems are mouse-sensitive. These items, and the operations that can be performed by clicking the mouse at them, vary from subsystem to subsystem and are described in the sections that follow. Often clicking the mouse on an item will give you a menu of things to do to that object.

The Peek window has scrolling capabilities, for use when the status display overruns the available display area. For details on these capabilities, see section 10.1, page 40.

As long as the Peek window is exposed, it will continue to update its display. Thus a Peek window can be used to examine things being done in other windows in real time.

This document does not attempt to describe the individual Peek subsystems. They are generally more or less self-explanatory.

8. Network Programs

The supdup and telnet programs allow you to use the Lisp machine as a terminal to another host. Supdup acts like a display terminal, while Telnet acts like a printing terminal for systems that don't know about displays. There are window types called **supdup** and **telnet**, as well as SYSTEM keys and Lisp functions.

When you go into a Supdup or a Telnet, it will ask you what host to connect to. Once you are connected to a host, all the keys on the keyboard are sent through to that host, instead of having their normal functions, except for the three escape keys, TERMINAL, SYSTEM, and NETWORK. TERMINAL and SYSTEM retain their normal functions, while NETWORK is used to give commands to the Supdup or Telnet itself. These commands are:

CALL	Switch back to the previous window, leaving the connection open.
B	Call break, getting into a read-eval-print loop.
C	Change the escape character. This is normally BREAK. Typing the escape character is the same as typing NETWORK, for the benefit of old keyboards.
D	Disconnect from the host and ask for another host to connect to.
I	Turn Imlac simulation on or off. This applies to Telnet only, and is only used when talking to TOPS-20.
L	Log out of the foreign host, disconnect, and switch back to the previous window. This is the normal way of exiting the program.
M	Turn **MORE** processing on or off. This applies to Telnet only.
P	Switch back to the previous window, leaving the connection open.
Q	Disconnect from the foreign host and switch back to the previous window.
? or HELP	Prints documentation on these commands.

9. Index of Function Keys

ABORT By convention when this is read by a program it aborts 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 break) and typing a star. Note that ABORT takes effect when it is read, not when it is typed; it will not stop a running program.

CTRL/ABORT Aborts the operation currently being performed by the process you are typing at, immediately (not when it is read). For instance, this will force a Lisp listener to abandon the present computation and return to its read-eval-print loop. On the old keyboards, ABORT does not exist as a separate key, but it can be typed as TOP/CALL, and so CTRL/ABORT can be typed as CTRL/TOP/CALL.

META/ABORT By convention when this is read by a program it aborts 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 break levels, then start a new read-eval-print loop.

CTRL/META/ABORT
A combination of CTRL/ABORT and META/ABORT, this immediately throws out of all levels of computation and restarts the process you type at it.

BREAK Usually forces the process you are typing at into a break read-eval-print loop, so that you can see what it's doing, or stop it temporarily. The effect occurs when the character is read, not immediately. Type RESUME to continue the interrupted computation (this applies to the three modified forms of the BREAK key as well).

CTRL/BREAK This is like BREAK but takes effect immediately rather than when it is read.

META/BREAK By convention forces the process you type it at into the error handler, when it is read. It should type out ">>BREAK" and the error-handler prompt "→". You can poke around in the process, then type RESUME or CTRL/C to continue.

CTRL/META/BREAK
Forces the process you type it at into the error handler, whether or not it is running.

CALL Immediately stops the process you are typing at, and selects an idle lisp-listener (creating one if there aren't any). This is the key to use to get to "command level" without destroying the computation in progress. When the window called out-of is selected again, its process will be allowed to run once more.

CLEAR-INPUT Usually flushes the input expression you are typing. This command can be given on the old keyboards by typing CLEAR.

CLEAR-SCREEN
Usually erases and refreshes the selected window. On the old keyboards, which have no CLEAR-SCREEN key, the FORM key can be used for this. In the editor (in searches and after CTRL/Q) this key inserts a page separator character, which displays as "page" in a box.

DELETE This key is for some as yet unspecified form of deletion. In Supdup it substitutes for the VT key of the old keyboards.

END Marks the end of input to many programs. Input of a single-line nature may be ended with RETURN, but END will terminate multiple-line input where RETURN is useful for separating lines. The END key does not apply when typing in Lisp expressions, which are self-delimiting. The old keyboards have no END key; TOP/RETURN may be used as a substitute.

HELP Usually gets you some on-line documentation or programmed assistance. On the old keyboards, HELP does not exist as a separate key, but it can be typed as TOP/H. See SYSTEM HELP, TERMINAL HELP.

HOLD-OUTPUT Not used currently.

LINE The function of this key varies considerably. It is used as a command by the editor, and sends a "line feed" character in Supdup and Telnet.

MACRO Introduces a keyboard-macro command in programs, such as the editor, that have keyboard macros. The MACRO key is only defined while running such programs. The BACK NEXT key may be used for this function on the old keyboards.

NETWORK This key is used to get the attention of a running Supdup or Telnet. As such it functions as a command prefix. This replaces BREAK on the old keyboards. See chapter 8, page 34.

OVER-STRIKE Moves the cursor back so that you can superpose two characters, should you really want to. The key called BS will do the same thing on the old keyboards.

QUOTE Not currently used.

RESUME Continues from the break function and the error handler. In Supdup this sends a backspace character, which is used for a resume-like command by ITS DDT.

RETURN "Carriage return" or end of line. Exact significance may vary.

RUBOUT Usually erases the last character typed. It is not the same as DELETE.

STATUS Not currently used.

STOP-OUTPUT Not currently used.

SYSTEM This key is a prefix for a family of commands, generally used to select a window of a specified type, such as Lisp Listener or Editor. These commands can be given from the old keyboards by typing TOP/ESC instead of SYSTEM. For a detailed description see section 2.3.7, page 13.

TAB This key is only sometimes defined. Its exact function depends on context, but in general it is used to move the cursor right to an appropriate point.

TERMINAL This key is a prefix for a family of commands relating to the display, which you may type at any time, no matter what program you are running. These are documented below. Most of these commands can be given from the old keyboards by using the ESC key.

TERMINAL A

Striking TERMINAL A (or ESC A on the old keyboards) arrests the process whose state is currently being displayed in the who-line. TERMINAL - A un-arrests that process, whether it was originally arrested by typing TERMINAL A, or CALL, or whether it was arrested for some other reason.

TERMINAL ABORT
TERMINAL BREAK
These are the same as ABORT and BREAK except that they always work. The plain ABORT and BREAK keys are turned off by certain programs such as Supdup. The META modifier may be used, and the CTRL modifier is assumed whether or not you type it.

TERMINAL C Complements the inverse-video mode of the screen. With a numeric argument, complements the inverse-video mode of the mouse-documentation line just above the who-line.

TERMINAL CALL
Puts you into a break read-eval-print loop, using the "cold-load-stream". This is a way of getting to a Lisp read-eval-print loop that completely bypasses the window system, which can be very useful in debugging, since it does not interact with very much of the rest of the system. On the old keyboards, this command may be given as ESC CALL.

TERMINAL CLEAR-INPUT
Discards any typed-ahead keyboard input which has not yet been read by a program. On the old keyboards, this command may be given as ESC CLEAR.

TERMINAL CLEAR-SCREEN
Clears the screen and refreshes all the windows, including the who-line. Use this when something has been clobbered, e.g. by use of the "cold-load-stream". On the old keyboards, this command may be given as ESC FORM.

TERMINAL (n) F
Displays a list of the users logged in to a machine. With no numeric argument, shows the users logged in to AI. With a numeric argument of 1, shows the users on Lisp machines and which Lisp machines are free. With an argument of 2, shows the users on MC. With an argument of 3, shows the users on AI and MC both. With an argument of 0, asks for a command line, similar to the jcl of the :FINGER command on ITS. This command goes over the network to gets its information, and consequently may take a while if the target machine is slow.

TERMINAL H Displays the status of all the hosts on the Chaosnet.

TERMINAL HELP
Displays documentation on all of the function keys, including the terminal escape commands. Type a space to return to your previous window. On the old keyboards, this command may be given as ESC ?.

TERMINAL O Selects the window which is exposed on some screen and was selected least recently. Thus repeating this command cycles through all the "selectable" exposed windows. This is a lot like "CTRL/X O" in the editor.

TERMINAL (n) S
Switches you to another "selectable" window. The "(n)" represents an optional numerical argument. For full details, see section 2.3.5, page 12.

TERMINAL (n) 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, or it can quietly type out "in the background"; when the window is next exposed the output will become visible. Similarly, if an attempt is made to read input from a window which 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 TERMINAL T command controls these characteristics based on its numeric argument, as follows:

TERMINAL T If output notification is off, turns input and output notification on. Otherwise turns input and output notification off. This essentially toggles the current state.

TERMINAL 0 T
Turns input and output notification off.

TERMINAL 1 T
Turns input and output notification on.

TERMINAL 2 T
Turns output notification on, and input notification off.

TERMINAL 3 T
Turns output notification off, and input notification on.

TERMINAL 4 T
Allows output to proceed in the background, and turns input notification on.

TERMINAL 5 T
Allows output to proceed in the background, and turns input notification off.

You aren't really expected to remember all of these magic numbers. As always, typing TERMINAL HELP will print a brief reminder of the commands.

TERMINAL W Controls the who-line. What happens depends on the number typed before the "W". With no numeric argument, the who-line is redisplayed. The numeric arguments control what process the who-line watches. The options are:

0 Gives a menu of all processes, and freezes the who-line on the process you select. When the who-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 who-line does not change to another process when you select a new window.

1 The who-line watches whatever process is talking to the keyboard, and changes processes when you select a new window. This is the default initial state.

2 Freezes the who-line on the process it is currently watching. If you select a new window the process will not change.

3 Freezes the who-line on the next process in a certain order.

4 Freezes the who-line on the next process in the other direction.

These numbers are the same as on AI TV terminals, except that there is no "system who-line" on the Lisp machine.

TERMINAL HOLD-OUTPUT

If it says "Output Hold" in the who-line, indicating that the "current" process is trying to display on a window which is not exposed, typing this command will expose that window. Otherwise typing this will beep. Use TERMINAL S to return to the previously-selected window. This function is unfortunately not available on old keyboards.

TERMINAL CTRL/T

De-exposes all temporary windows. This is useful if the system seems to be hung because there is a temporary window on top of the window which is trying to type out and tell you what's going on.

TERMINAL CTRL/CLEAR-INPUT

Clears the locks on all the windows in the system, thus giving the window system a swift kick in the pants. This often works to unwedge a catatonic window system. This is a last resort, but not as drastic as warm booting. It should be used when none of the windows will talk to you, when you can't get a system menu, etc.

10. Quick Summary of Mouse Functions

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. It indicates that there no special commands on the mouse buttons. Clicking Left will select the window pointed-to. Clicking 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. You have several editor commands on the mouse buttons. See section 4.1, page 21.

A thin 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 wants to be unobtrusive, for instance in menus.

10.1 Scrolling

Some windows display a "contents" which may be too big to fit entirely in the window. The editor and the inspector are examples. When this is the case, you see only a portion of the contents, and you can scroll it up and down using the mouse.

The following mouse cursors indicate that the mouse is being used to control scrolling:

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 will scroll the window, revealing more of the text in the direction the arrow points, while moving the mouse quickly will let you out of the window.

Scrolling zones often say *more above* or *more below* in small italic letters. Clicking on one of these legends will scroll the window up and down by its height, thus you will see the next or previous windowfull. When the window is at the top or bottom of its contents, so that it is not possible to scroll any farther in one direction, the legend in the scrolling zone will change to indicate this fact.

A fat double-headed arrow. There will be a thin black bar nearby, the "scroll bar". The size of this bar relative to the edge of the window to which it is attached shows what portion of the window's contents is visible. The vertical position of the bar within the edge shows the position of the visible portion of the window's contents relative to the whole. The mouse commands in this case are

Left	Move the line next to the mouse to the top of the window.
Left Double	Move the line next to the mouse to the bottom of the window.
Right	Move the top line to where the mouse is.
Right Double	Move the bottom line to where the mouse is. Because of this command definition, you cannot get to the system menu while the mouse is displaying a

double-headed fat arrow.

Middle Jump to a place in the window contents as far, proportionally, from its beginning as the mouse is from the top of the window.