

# ***Artificial Intelligence Systems***

## ***Interlisp-D: A Friendly Primer***

**XEROX**

**Interlisp-D:  
A Friendly Primer**

**3102300  
November, 1986**

---

**Copyright (c) 1986 Xerox Corporation**

All rights reserved.

This publication may not be reproduced or transmitted in any form by any means, electronic, microfilm, xerography, or otherwise, or incorporated into any information retrieval system, without the written permission of Xerox Corporation.

---

This primer was developed for Xerox Artificial Intelligence Systems by Computer Possibilities, a company dedicated to making advanced computer systems available to businesses and organizations. For more information, contact Computer Possibilities, 320 South Pacific Avenue, Pittsburgh, PA. (412) 441-8949.

[This page intentionally left blank]

*It was dawn and the local told him it was down the road a piece, left at the best fishing bridge in the county, right at the apple tree stump, and onto the dirt road just before the hill. At midnight he knew he was lost.*

—Anonymous

Welcome to the Interlisp-D programming environment! The Interlisp-D environment truly must be one of the most sophisticated and powerful tools in use by human beings. Overall, it is flexible, well thought out, and full of pleasant surprises: "Wow, here are exactly the set of functions I thought I'd need to write." Unfortunately, along with the power comes mind-numbing complexity. The *Interlisp Reference Manual* describes the functions and some of the tools available in the Interlisp-D environment. To do this takes three large volumes. Other volumes are needed to document the library packages and other newly written tools. Needless to say, it is very difficult to learn such a huge amount of material when there is no way to determine where to start!

We developed this primer to provide a starting point for new Interlisp-D users, to enhance your excitement and challenge you with the potential before you. We assume you know a little about LISP, most likely received from taking a survey course in Artificial Intelligence (AI), and have seen a demonstration of how Interlisp-D runs on your 1186 or 1108. We further assume that your machine is not on a network system with a file server - though this is addressed, and that you will be working from floppy disks and the hard disk that is part of the machine. If this describes your situation, you are ready to sit down in front of your machine and follow the step-by-step examples provided in this primer.

The primer is broken into many small chapters, and these chapters are organized into five parts. You may want to read Parts 1 through 3 straight through, since they describe the basics of using the machine. Each chapter in Sections 4 and 5, however, can be used to learn a specific skill whenever you are ready to for it.

Part one, "Introduction", includes Chapters 1 and 2. Part two, "Getting Into/Out of Interlisp", includes Chapters 3 through 5. Part three, "The Interlisp-D language and Programming Environment", includes Chapters 6 through 15. These chapters discuss primary elements in Interlisp-D, and orient you in relation to those elements. Part four, "Important Other Things to Know to Work Successfully", includes Chapters 16 through 31. Part five, "More Language and Environment and Packages", includes Chapters 32 through 44.

Through out we make reference to the *Interlisp-D Reference Manual* by section and page number. The material in the primer is just an introduction. When you need more depth use the detailed treatment provided in the manual.

While only you can plot your ultimate destination, you will find this primer indispensable for clearly defining and guiding you to the first landmarks on your way.

#### Acknowledgements

The early inspiration and model for this primer came from the Intelligent Tutoring Systems group and the Learning Research and Development Center at the University of Pittsburgh. We gratefully acknowledge their pioneering contribution to more effective artificial intelligence.

This primer was developed by Computer Possibilities, a company committed to making AI technology available. Primary development and writing was done by Cynthia Cosic, with technical writing support provided by Sam Zordich.

At Xerox Artificial Intelligence Systems, John Vittal managed and directed the project. Substantial assistance was provided by many members of the AIS staff who provided both editorial and systems support.

<b>1. A Brief Glossary</b>	1.1
<b>2. The Mouse and the Keyboard</b>	2.1
2.1. The Mouse	2.1
2.1.1. 2 and 3 Button Mice	2.1
2.2. The Keyboard	2.2
2.2.1. The 1186 Keyboard	2.2
2.2.2. The 1108 Keyboard	2.2
<b>3. Turning On Your Lisp Machine</b>	3.1
3.1. Turning on the 1108	3.1
3.2. Turning on the 1186	3.2
3.3. Loading Interlisp-D from the Hard Disk	3.3
3.4. After Booting Lisp	3.5
3.5. Restarting Lisp After Logging Out	3.5
<b>4. If You Have a Fileserver</b>	4.1
4.1. Turning on your 1108	4.1
4.2. Turning on your 1186	4.1
4.3. Location of Files	4.2
4.4. The Timeserver	4.2
<b>5. Logging Out And Turning the Machine Off</b>	5.1
5.1. Logging Out	5.1
5.2. Turning The Machine Off	5.2
<b>6. Typing Shortcuts</b>	6.1
6.1. If you make a Mistake	6.3
<b>7. Using Menus</b>	7.1
7.1. Making a Selection from a Menu	7.2
7.2. Explanations of Menu Items	7.2
7.3. Submenus	7.3
<b>8. How to use Files</b>	8.1
8.1. Types of Files	8.1



---

8.2. Directories	8.1
8.3. Directory Options	8.2
8.4. Subfile Directories	8.3
8.5. To See What Files Are Loaded	8.3
8.6. Simple Commands for Manipulating Files	8.3
8.7. Connecting to a Directory	8.4
8.8. File Version Numbers	8.4
<b>9. FileBrowser</b>	<b>9.1</b>
9.1. Calling the FileBrowser	9.1
9.2. FileBrowser Commands	9.3
<b>10. Those Wonderful Windows!</b>	<b>10.1</b>
10.1. Windows provided by Interlisp-D	10.1
10.2. Creating a window	10.2
10.3. The Right Button Default Window Menu	10.2
10.4. An explanation of each menu item	10.3
10.5. Scrollable Windows	10.3
10.6. Other Window Functions	10.5
10.6.1. PROMTPRINT	10.5
10.6.2. WHICHW	10.6
<b>11. Editing and Saving</b>	<b>11.1</b>
11.1. Defining Functions	11.1
11.2. Simple Editing in the Interlisp-D Executive Window	11.2
11.3. Using The List Structure Editor	11.4
11.3.1. Commenting Functions	11.6
11.4. File Functions and Variables - How to See Them and Save Them	11.7
11.5. File Variables	11.7
11.6. Saving Interlisp-D on Files	11.7
<b>12. Your Init File</b>	<b>12.1</b>
12.1. Making an Init File	12.2
<b>13. Flexibility and Forgiveness: CLISP and DWIM</b>	<b>13.1</b>
13.1. CLISP	13.1
13.2. DWIM	13.2
<b>14. Break Package</b>	<b>14.1</b>
14.1. Break Windows	14.1
14.2. Break Package Example	14.1

---

14.3. Ways to Stop Execution from the Keyboard, called "Breaking Lisp"	14.3
14.4. Programming Breaks and Debugging Code	14.4
14.5. Break Menu	14.4
14.6. Returning to Top Level	14.5
<b>15. On-Line Help with Interlisp-D: HELPSYS and DINFO</b>	<b>15.1</b>
15.1. HelpSys	15.1
15.2. DInfo	15.1
<b>16. Floppy Disks</b>	<b>16.1</b>
16.1. Buying Floppy Disks	16.1
16.2. Basic Floppy Disk Information	16.1
16.3. Care of Floppies	16.2
16.4. Write Enabling and Write Protecting Floppies	16.3
16.4.1. Write Enabling an 1108's Floppy Disk	16.3
16.4.2. Write Protecting an 1186's Floppy Disk	16.3
16.5. Inserting Floppies into the Floppy Drive	16.3
16.6. Functions for Floppy Disks	16.4
16.6.1. Formatting Floppies	16.4
16.6.2. Available Space on a Floppy Disk	16.4
16.6.3. The Name of a Floppy Disk	16.4
16.6.4. FLOPPY.MODE	16.5
<b>17. Duplicating Floppy Disks</b>	<b>17.1</b>
17.1. Supplies	17.1
17.2. Preparation	17.1
17.2.1. Handling Floppy Disks	17.1
17.2.2. Setup	17.1
17.3. Copying Floppy Disks	17.2
<b>18. Sysout Files</b>	<b>18.1</b>
18.1. Loading SYSOUT Files	18.1
18.1.1. Loading a SYSOUT file on the 1108	18.1
18.1.2. Loading a SYSOUT file on the 1186	18.2
18.2. Making Your Own SYSOUT File	18.3
<b>19. Using the Epson FX80 Printer</b>	<b>19.1</b>
19.1. Initializing the RS232 Port	19.1
19.2. Power up the Printer	19.1
19.3. To Align Top of Page	19.1

---

<b>19.4. Functions To Print Files and Bitmaps</b>	<b>19.2</b>
19.4.1. RS232.Print	19.2
19.4.2. FX80STREAM	19.2
19.4.3. Printing a Portion of the Screen	19.3
<b>20. RS232 File Transfer With a VAX</b>	<b>20.1</b>
20.1. Prerequisites	20.1
20.2. Using Chat to Transfer Files	20.1
<b>21. Ethernet File Transfer</b>	<b>21.1</b>
21.1. Prerequisites	21.1
21.2. File Transfer	21.1
<b>22. What To Do If ...</b>	<b>22.1</b>
<b>23. The Text Editor, TEdit</b>	<b>23.1</b>
23.1. Using TEdit	23.1
23.2. Managing the TEdit Window	23.2
23.3. Selecting Text	23.3
23.4. Deleting, Copying, and Moving Text with TEdit	23.4
23.4.1. Deleting Text From a File	23.4
23.4.2. Copying Text	23.4
23.4.3. Moving Text	23.5
23.5. TEdit Menus	23.6
23.5.1. Finding and Substituting Text with TEdit	23.7
23.5.1.1. Finding Text	23.7
23.5.1.2. Substituting Text	23.8
23.5.2. Text Formatting	23.10
23.5.2.1. Choosing Fonts	23.10
23.5.2.2. Paragraph Formatting	23.11
23.5.3. Adding Bitmaps and Sketches to your TEdit File	23.13
23.5.3.1. Adding a Bitmap to your TEdit file	23.13
23.5.3.2. Adding a Sketch to your TEdit file	23.14
23.5.4. Getting and Including Files	23.14
23.5.4.1. Get	23.14
23.5.4.2. Include	23.14
23.5.5. Saving and Printing Files	23.15
<b>24. Records May Be Your Favorite Data Structure!</b>	<b>24.1</b>
24.1. Interlisp Record Primitives	24.1

---

24.2. Example	24.3
24.3. A Few Tips	24.4
<b>25. Local Variables - Using LET and PROG</b>	<b>25.1</b>
25.1. LET	25.1
25.2. PROG	25.3
25.3. Parallel versus Sequential Variable Binding	25.6
25.3.1. LET*	25.6
25.3.2. PROG*	25.7
<b>26. Iterative statements</b>	<b>26.1</b>
26.1. General Structure and Use	26.1
26.2. Local Variables	26.2
26.3. Iteration On Lists	26.3
26.4. Parallel Iteration	26.4
26.5. Conditional Iteration	26.5
26.6. More Iteration	26.6
<b>27. Windows and Regions</b>	<b>27.1</b>
27.1. Windows	27.1
27.1.1. CREATEW	27.1
27.1.2. WINDOWPROP	27.2
27.1.3. Getting windows to do things	27.3
27.1.3.1. BUTTONEVENTFN	27.4
27.1.4. Looking at a window's properties	27.5
27.2. Regions	27.5
<b>28. What Are Menus?</b>	<b>28.1</b>
28.1. Displaying Menus	28.1
28.2. Getting Menus to DO Stuff	28.2
28.2.1. The WHENHELDFN and WHENSELECTEDFN fields of a menu	28.4
28.3. Looking at a menu's fields	28.5
<b>29. Bitmaps</b>	<b>29.1</b>
<b>30. Displaystreams</b>	<b>30.1</b>
30.1. Drawing on a Displaystream	30.1
30.1.1. DRAWLINE	30.1
30.1.2. DRAWTO	30.2
30.1.3. DRAWCIRCLE	30.3

30.1.3.1. FILLCIRCLE	30.3
30.2. Locating and Changing Your Position in a Displaystream	30.4
30.2.1. DSPXPOSITION	30.5
30.2.2. DSPYPOSITION	30.5
30.2.3. MOVETO	30.5
<b>31. Fonts</b>	31.1
31.1. What makes up a FONT?	31.1
31.2. Fontdescriptors, and FONTCREATE	31.2
31.3. Display Fonts - Their files, and how to find them	31.3
31.4. Interpress Fonts - Their files, and how to find them	31.4
31.5. Functions for Using Fonts	31.4
31.5.1. FONTPROP - Looking at Font Properties	31.4
31.5.2. STRINGWIDTH	31.5
31.5.3. DSPFONT - Changing the Font in One Window	31.6
31.5.4. Globally Changing Fonts	31.7
31.5.5. Personalizing Your Font Profile	31.7
<b>32. The Inspector</b>	32.1
32.1. Calling the Inspector	32.1
32.2. Using the Inspector	32.2
32.3. Inspector Example	32.2
<b>33. Masterscope</b>	33.1
33.1. The SHOW DATA command and GRAPHER	33.2
33.2. Databasefns: Automatic Construction and Upkeep of a Masterscope Database	33.3
<b>34. Where Does All the Time Go? SPY</b>	34.1
34.1. How to use Spy with the SPY Window	34.1
34.2. How to use SPY from the Lisp Top Level	34.2
34.3. Interpreting SPY's Results	34.2
<b>35. SKETCH</b>	35.1
35.1. Starting Sketch	35.1
35.2. Selecting Sketch elements	35.1
35.3. Drawing with Sketch	35.2
35.3.1. Simple Shapes: Circles, Ellipses, and Boxes	35.3
35.3.1.1. Drawing Circles	35.3
35.3.1.2. Ellipses	35.3

35.3.1.3. Boxes	35.3
35.3.1.4. Changing a Box's Filling	35.4
35.3.2. Lines, Curves, and Arcs	35.4
35.3.2.1. A Single Line	35.4
35.3.2.2. A Series of Lines	35.4
35.3.2.3. Drawing an Open Curve	35.5
35.3.2.4. An Arc	35.5
35.3.3. Closed Curves and Polygons	35.6
35.4. Adding a Bitmap to a Sketch	35.7
35.5. To Add Text to a Sketch	35.8
35.5.1. Editing Text	35.9
35.6. Editing a Sketch	35.10
35.7. Saving Your Work	35.11
35.8. To Continue a Sketch That Has Been Saved on a File	35.11
<b>36. Free Menus</b>	<b>36.1</b>
36.1. An Example Free Menu	36.1
36.2. Parts of a Free Menu Item	36.2
36.3. Types of Free Menu Items	36.3
<b>37. The Grapher</b>	<b>37.1</b>
37.1. Say it with Graphs	37.1
37.2. Making a Graph from a List	37.4
37.3. Incorporating Grapher into Your Program	37.5
37.4. More of Grapher	37.5
<b>38. Virtual Keyboards, and the Keyboard Editor</b>	<b>38.1</b>
38.1. Using the Virtual Keyboards Package	38.1
38.2. Using the Keyboard Editor	38.2
<b>39. IconW</b>	<b>39.1</b>
<b>40. TELERAID</b>	<b>40.1</b>
<b>41. Resource Management</b>	<b>41.1</b>
41.1. Naming Variables and Records	41.1
41.2. Some Space and Time Considerations	41.3
41.2.1. Global Variables	41.3
41.2.2. Circular Lists	41.4
41.2.3. When You Run Out Of Space	41.4

<b>42. Simple Interactions with the Cursor, a Bitmap, and a Window</b>	42.1
<b>42.1. An Example Function Using GETMOUSESTATE</b>	42.1
<b>42.2. Advising GETMOUSESTATE</b>	42.2
<b>42.3. Changing the Cursor</b>	42.2
<b>42.4. Functions for "Tracing the cursor"</b>	42.3
<b>42.5. Running the Functions</b>	42.6
<b>43. Glossary of Global System Variables</b>	43.1
<b>43.1. Directories</b>	43.1
<b>43.2. Flags</b>	43.2
<b>43.3. History Lists</b>	43.3
<b>43.4. System Menus</b>	43.3
<b>43.5. Windows</b>	43.4
<b>43.6. Miscellaneous</b>	43.4
<b>44. Other References that will be Useful to You</b>	44.1

---

The following definitions will acquaint you with general terms used throughout this primer. You will probably want to read through them now, and use this chapter as a reference while you read through the rest of the primer.

- advising** An Interlisp-D facility for specifying function modifications without necessarily knowing how a particular function works or even what it does. Even system functions can be changed with advising.
- argument** An argument is a piece of information given to an Interlisp-D function so that it can execute successfully. When a function is explained in the primer, the arguments that it requires will also be given. Arguments are also called parameters.
- atom** The smallest structure in Lisp; like a variable in other programming languages, but can also have a property list and a function definition.
- Background Menu** The menu that appears when the mouse is not in any window and the right mouse button is pressed. A typical background menu is shown in Figure 1.1.

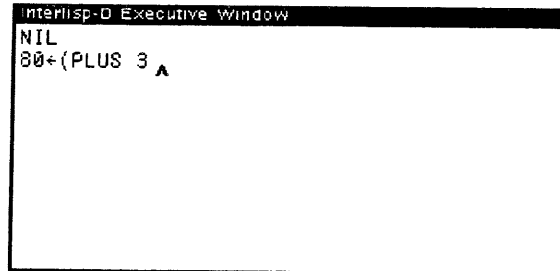


**Figure 1.1.** The Menu that appears when the mouse is not in any window, and the right mouse button is pressed. Your background menu may have some different items in it

- binding** The value of a variable. It could be either a local or a global variable. See unbound.
- bitmap** A rectangular array of "pixels," each of which is on or off representing one point in the bitmap image.
- BREAK** An Interlisp function that causes a function to stop executing, open a Break window, and allow the user to find out what is happening while the function is halted.
- Break Window** A window that opens when an error is encountered while running your program (i.e., when your program has broken). There are tools to help you debug your program from this window. This is explained further in Chapter 14, Page 14.1.
- browse** To examine a data structure by use of a display that allows the user to "move" around within the data structure.
- button**



- (1) (n.) A key on a mouse.
- (2) (v.t.) To depress one of the mouse keys when making a selection.
- CAR** A function that returns the head or first element of a list. See **CDR**.
- caret** The small blinking arrowhead that marks where text will appear when it is typed in from the keyboard. An example of the caret in the Interlisp-D Executive Window is shown in Figure 1.2.



**Figure 1.2.** The caret is to the right of the number 3. When a character is typed at the keyboard, it will appear at the caret

- CDR** A function that returns the tail (that is, everything but the first element) of a list. See **CAR**.
- CLISP** A mechanism for augmenting the standard Lisp syntax. One such augmentation included in Interlisp is the iterative statement. See Section 13.1.
- cr** Please press your carriage return key.
- datatype**
- (1) The kind of a datum. In Interlisp, there are many system-defined datatypes e.g. Floating Point, Integer, Atom, etc.
- (2) A datatype can also be user-defined. In this case it is like a record made up from system types and other user-defined datatypes.
- DWIM** "Do-what-I-mean." Many errors made by Interlisp users could be corrected without any information about the purpose of the program or expression in question (e.g. misspellings, certain kinds of parenthesis errors). The DWIM facility is called automatically whenever an error occurs in the evaluation of an Interlisp expression. If DWIM is able to make a correction, the computation continues as though no error had occurred; otherwise, the standard error mechanism is invoked.
- error** Occasionally, while a program is running, an error may occur which will stop the computation. Interlisp provides extensive facilities for detecting and handling error conditions, to enable the testing, debugging, and revising of imperfect programs.
- evaluate or EVAL** Means to find the value of a form. For example, if the variable X is bound to 5, we get 5 by evaluating X. Evaluation of a Interlisp function involves evaluating the arguments and then applying the function.
- file package** A set of functions and conventions that facilitate the bookkeeping involved with working in a large system consisting of many source code files and their compiled counterparts. Essentially, the file package keeps track of where things are and

	what things have changed. It also keeps track of which files have been modified and need to be updated and recompiled.
form	Another way of saying s-expression. An Interlisp-D expression that can be evaluated.
function	A Lisp function is a piece of lisp code that executes and returns a value.
history	The programmer's assistant is built around a memory structure called the history list. The history functions (e.g. FIX, UNDO, REDO) are part of this assistant. These operations allow you to conveniently re-work previously specified operations.
History List	As you type on the screen, you will notice a number followed by a prompt arrow. Each number, and the information on that line, is sequentially stored as the History List. Using the History List, you can easily reexecute lines typed earlier in a worksession. See Chapter 6.
icon	A pictorial representation, usually of shrunken window.
Interlisp-D Executive Window	This is your main window, where you will run functions and develop your programs. See Figure 1.3. This is the window that the caret is in when you turn on your machine and load Interlisp-D.

```

Interlisp-D Executive Window
NIL
80+(PROMPTPRINT "HELLO" ^

```

Figure 1.3. TTY Window

inspector	An interactive display program for examining and changing the parts of a data structure. Interlisp-D has inspectors for lists and other data types.
iterative statement	(also called i.s.) A statement in Interlisp that repetitively executes a body of code. (E.g. (for x from 1 to 5 do (PRINT x)) is an i.s.)
iterative variable	(also called i.v.) Usually, an iterative statement is controlled by the value that the i.v. takes on. In the iterative statement example above, <p style="text-align: center;">x</p> is the iterative variable because its value is being changed by each cycle through the loop. All iterative variables are local to the iterative statement where they are defined.
LISP	Family of languages invented for "list processing." These languages have in common a set of basic primitives for creating and manipulating symbol structures. Interlisp-D is an implementation of the LISP language together with an environment (set of tools) for programming, an a set of packages that extend the functionality of the system.
list	A collection of atoms and lists; a list is denoted by surrounding its contents with a pair of parentheses.

- Loading LISP** This is the process of bringing Interlisp-D from floppy disks, hard disks, or some other secondary storage into your main, or working, memory. You will need to load (i.e., install, and boot) Interlisp-D if you have not logged off the machine at the end of a session. The process of loading Interlisp-D is explained in Chapter 3.
- Maintenance Panel Codes** Should you have a problem with your equipment, these codes will indicate the status of your processor. On the 1108, these are the red LED numbers under the floppy drive door. There is a cover over these numbers. Pull down the cover located immediately under the floppy door button. The code numbers are defined for the 1108 in the *1108 User's Guide*, in the MP Codes Chapter.
- If there is a problem with the 1186, the mouse cursor will change from its normal arrow to the code number that describes the problem. The code numbers are defined for the 1186 in the *1186 User's Guide* in the Cursor Codes subsection of the Diagnostics Chapter.
- Masterscope** A program analysis tool. When told to analyze a program, Masterscope creates a data base of information about the program. In particular, Masterscope knows which functions call other functions and which functions use which variables. Masterscope can then answer questions about the program and display the information with a browser.
- menu** A way of graphically presenting the user with a set of options. There are two kinds of menus: pop-up menus are created when needed and disappear after an item has been selected; permanent menus remain on the screen after use.
- mouse** The Mouse is the box to the right of your keyboard. It controls the movement of the cursor on your screen. As you become familiar with the mouse, you will find it much quicker to use the mouse than the keyboard. See Figure 1.4. (Note: Some mice have three buttons; the button in the center is known as the middle mouse button. If your mouse has only two buttons, you can simulate a middle button by pressing the left and right buttons simultaneously.)

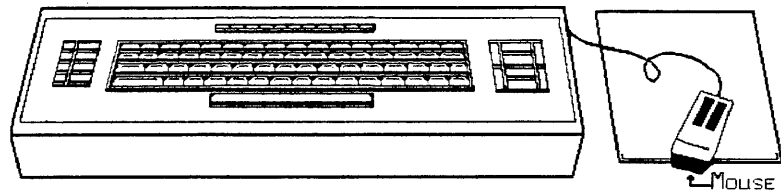


Figure 1.4. Mouse

- Mouse Cursor** The small arrow on the screen that points to the northwest. See Figure 1.5.



Figure 1.5. Mouse Cursor

#### Mouse Cursor Icons



Wait. The processor is busy.



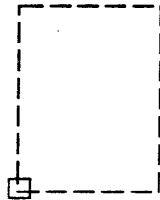
The processor is saving a snapshot of your current system session. This is usually done when the processor has been idle for a while.



The "Mouse Confirm Cursor". It appears when you have to confirm that the choice you just made was correct. If it was, press the left button. If the choice was not right, press the right button to abort.



This means "sweep out" the shape of the window. To do this, move the mouse to a position where you want a corner. Press the left mouse button, and hold it down. Move the mouse diagonally to sketch a rectangle. When the rectangle is the desired size and shape, release the left button.



This is the "move window" prompt. Move the mouse so that the large "ghost" rectangle is in the position where you want the window. When you click the left mouse button, the window will appear at this new location.

**NIL** NIL is the Interlisp-D symbol for the empty list. It can also be represented by a left paren followed by a right paren: (). It is the only expression in Interlisp-D that is both an atom and a list.

**pixel** Pixel stands for Picture Element. The screen of your Lisp Machine is made up of a rectangular array of pixels. Each pixel corresponds to one bit. When a bit is turned on, i.e. set to 1, the pixel on the screen represented by this bit is black.

**pretty printing** Pretty printing refers to the way Interlisp-D functions are printed with special indentation, to make them easier to read. Functions are pretty printed in the structure editor, DEdit (See Section 11.3, Page 11.4). You can pretty print uncompiled functions by calling the function PP with the function you would like to see as an argument, i.e. (PP *function-name*). For an example of this, see Figure 1.6.

```

Top level -- Connected to {DSK}<LISPFILES>PRIMER>
NIL
96←(PP HEAD)
(HEAD
 [LAMBDA (LST)
 (CAR LST)])
(HEAD)
97←A
(* edited: "28-Jun-88 13:35")

```

Figure 1.6. An example use of the pretty printing function, PP

**Programmer's Assistant** The programmer's assistant accesses the History List to allow you to **FIX**, **UNDO**, and/or **REDO** your previous expressions typed to the Interlisp-D executive window. (See Chapter 6.)

**Prompt Window** The skinny black window at the top of the screen. It displays system prompts, or prompts you have developed. (See Figure 1.7.)



Figure 1.7. Prompt Window

**property list** A list of the form ( <property-name1> <property-value1> <property-name2> <property-value2> . . . ) associated with an atom. It accessed by the functions GETPROP and PUTPROP.

**record** A record is a data-structure that consists of named "fields". Accessing elements of a record can be separated from the details of how the data structure is actually stored. This eliminates many programming details. A record definition establishes a record template, describing the form of a record. A record instance is an actual record storing data according to a particular record template. (See datatype, second definition.)

**Right Button Default Window Menu** This is the menu that appears when the mouse is in a window, and the right mouse button is pressed. It looks like the menu in Figure 1.8. If this menu does not appear when you depress the right button of the mouse and the mouse is in the window, move the mouse so that it is pointing to the title bar of the window, and press the right button.

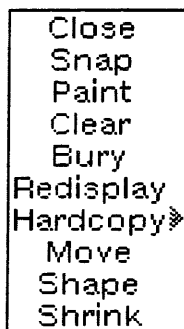


Figure 1.8. The Right Button Default Window Menu

**s-expression** Short for "symbolic expression." In Lisp, this refers to any well-formed collection of left parens, atoms, and right parens.

**stack** A pushdown list. Whenever a function is entered, information about that specific function call is pushed onto (i.e. added to the front of) the stack; this information includes the variable names and their values associated with the function call. When the function is exited, that data is popped off the stack.

**storage devices** Information is stored for your Lisp machine on floppy disks, or on the hard disk. They are referred to as {FLOPPY} and {DSK} respectively.

**sysout** A file containing a whole Lisp environment: namely, Interlisp-D, everything the user defined or loaded into the environment, the

- windows that appeared on the screen, the amount of memory used, and so on. Everything is stored in the `sysout` file exactly as it was when the function `SYSOUT` was called).
- TRACE** A function that creates a trace of the execution of another function. Each time the traced function is called, it prints out the values of the arguments it was called with, and prints out the value it returns upon completion.
- unbound** Without value; an atom is unbound if a value has never been assigned to it.
- window** A rectangular area of the screen that acts as the main display area for some Interlisp process.

[This page intentionally left blank]

## 2. THE MOUSE AND THE KEYBOARD

---

---

### 2.1 The Mouse

---

The mouse is the small box with buttons beside the keyboard. See Figure 1.4 in Chapter 1, Page 1.4. It moves around on a mouse pad, a small piece of plastic for a mechanical mouse, or grey paper for an optical mouse. The pad keeps the mouse from picking up dirt and oils from other surfaces, and should be placed squarely on the desktop. The cord to the mouse, its "tail", should always be directed perpendicular to and towards the back of the mouse pad. With the tail in this direction, you can always be sure that when you move the mouse, the mouse cursor will move in the same direction. The mouse and the mouse cursor should always move together in the same direction.

If the mouse is at the end of the pad, pick up the mouse and move it to the center of the pad. As long as the metal balls of a mechanical mouse do not move, or the sensors of an optical mouse cannot detect the grey and white blocks of the mouse pad, the mouse cursor on the screen will not move.

#### 2.1.1 2 and 3 Button Mice

---

Your mouse may have 2 or 3 buttons. If there are two buttons on the mouse, they are referred to as the left and right mouse buttons. It may, however, have three buttons; the button in the center is known as the middle mouse button. If your mouse has only two buttons, you can simulate a middle button by simultaneously pressing the left and right buttons.

When you press the mouse buttons, hold the buttons down until you are sure of what you want to do. Menus are displayed by pressing the mouse buttons, and choices are made by releasing the mouse buttons. Careful! - Don't click the mouse buttons too fast, or you may make a choice from a menu that you didn't even get a chance to see!



## 2.2 The Keyboard

---

Most of the keys on the keyboard of the 1108 and the 1186 are arranged like those on a typewriter. Some keys, however, are different. Some have special functions for editors or other programs, and will be explained when they are needed for the program. The rest will be explained here, so that you can acquaint yourself with them before going on.

### 2.2.1 The 1186 Keyboard

---

On the 1186 Keyboard, the control key is the one marked "CTL" at the bottom of the leftmost group of keys. This key will may be marked "EDIT."

The "back arrow" (←) is the shifted "-" (hyphen) key (hold down the shift key, and press "-"), and is labelled with a "—".

The function keys are those in the top row of the keyboard. They are numbered F1 through F10, from left to right. When one of these is needed, such as when turning on the machine, and booting Interlisp-D, you will be told which one to press.

### 2.2.2 The 1108 Keyboard

---

On the 1108 Keyboard, the control key is the one marked either "PROPS" or "CTL" at the bottom of the leftmost group of keys.

The backslash key is labelled with a small forward pointing arrow, and is located above the tab key.

Type a vertical bar by holding down the shift key and pressing the key labelled with a small forward pointing arrow, located above the tab key.

The "up arrow" key is the shifted 6 key (hold down the shift key, and press 6); it is labelled with a cent-sign.

The "back arrow" (←) is the shifted "-" (hyphen) key (Hold down the shift key, and press "—").

### 3. TURNING ON YOUR LISP MACHINE

---

The focus of this chapter is the steps that must be done between turning the machine on and actually beginning to use Interlisp-D. In particular, it is necessary to get a copy of the Interlisp-D sysout (complete stored version of the Interlisp-D environment) into your machine. This chapter assumes that you do not have a file server. If your machine is connected to a network with a fileserver, see Chapter 4, Page 4.1

To load Interlisp-D from the hard disk, your hard disk must be larger than 10 megabytes. If you have a 10 megabyte disk you must reload Lisp from floppies. (See Section 3.3, Page 3.3, or Section 18.1.2, Page 18.2.

If your disk is larger than 10 Mb, then Interlisp-D should already be loaded on the hard disk. If it isn't, refer to Section 3.3, Page 3.3, or Section 18.1.2, Page 18.2.

---

#### 3.1 Turning on the 1108

---

- (1) Pull down the cover of the maintenance panel, and expose the four digit LED display.
- (2) Insert the floppy disk labeled "INSTALLATION UTILITY" into the floppy drive. (If you have never handled floppy disks before, be sure to read "Care Of Floppys", Section 16.3, Page 16.2.)
- (3) To turn the machine on, simultaneously press both the B RESET and the ALT B buttons, then push the rocker switch to the 1 position.
- (4) Release only the B RESET button, and watch the red numbers count up 0000, 0001, 0002. As soon as it reads 0002 release the ALT B button. This is called a "2 BOOT".  

If you accidentally kept the ALT B button depressed when 0002 rolled around, just continue to hold it down, and the machine will count around to 0002 again.
- (5) When the question "Time offset from Greenwich?" appears, type "-5" for Eastern Standard Time (subtract one for each time zone westward), and press `CR`. See Figure 3.1.
- (6) For the next 3 questions, simply type `CR` after each prompt.

- (7) You will be requested to enter the date and time. Use the MM/DD/YY HH:MM:SS format. For example, if the date was June 24, 1986 5 PM, enter: 6/24/86 17:00.
- (8) A menu will appear, as in Figure 3.1. Choose number 12, Boot SystemTools Volume.

```

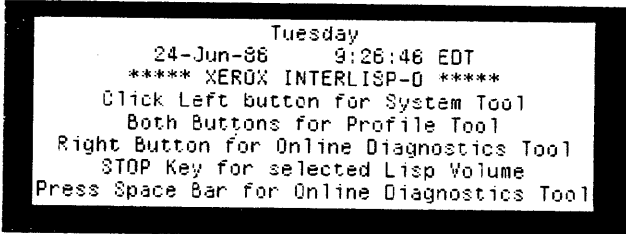
Installation Utility 8.0
Copyright (C) 1981, 1982, 1983, 1984 by Xerox Corporation
All rights reserved.
Time is not set
Time zone offset from Greenwich[-12..12]: -5
Minute offset[0..59]: 0
First day of Daylight Savings Time[0..366]: 121
Last day of Daylight Savings Time[0..366]: 305
Please Enter the date and time: 6/24/86 9:18
Set time to June 24, 86 9:18:00
OK? (Y/N): Y

Utility Options:
1 Partition 80 Mb disk according to USER definition
2 Partition 10 Mb disk for 1 Lisp volume and 1 LispFiles volume(1500 Pages)
3 Partition 29 Mb disk for 2 Lisp volumes(8 Mb) and 1 LispFiles volume(25000 Pages)
4 Partition 29 Mb disk for 2 Lisp volumes(8 Mb) and 1 LispFiles volume(8000 pages)
5 Partition 42 Mb disk for 2 Lisp volumes(10 Mb, 10 Mb) and 1 LispFiles volume(21100 pages)
6 Partition 42 Mb disk for 2 Lisp volumes(15 Mb, 8 Mb) and 1 LispFiles volume(17000 pages)
7 Partition 42 Mb disk for 3 Lisp volumes(8 Mb, 8 Mb, 4 Mb) and 1 LispFiles volume(21000 pages)
8 Partition 42 Mb disk for 3 Lisp volumes(15 Mb, 8 Mb, 4 Mb) and 1 LispFiles volume(25000 pages)
9 Partition 80 Mb disk for 4 Lisp volumes(32 Mb, 8 Mb, 8 Mb, 4Mb) and 2 LispFiles volume(15000, 17000 pages)
10 Partition 80 Mb disk for 4 Lisp volumes(15 Mb, 15 Mb, 15 Mb, 4Mb) and 2 LispFiles volume(15000, 17000 pages)
11 Initialize Workstation SystemTools Volume
12 Boot SystemTools volume
13 Erase SystemTools volume
14 Erase LispFiles volume
15 Erase Lisp volume
16 Install Lisp Microcode File
17 Physical Volume Scavenge
18 Scavenge LispFiles Volume
19 Scavenge SystemTools Volume
Enter choice number, then press RETURN: 12

```

Figure 3.1. The 1108 screen when booting the machine

- (9) After a few minutes, you will get a black screen with a white bouncing rectangle like the one in Figure 3.2.



```

Tuesday
24-Jun-86 9:26:46 EDT
***** XEROX INTERLISP-D *****
Click Left button for System Tool
Both Buttons for Profile Tool
Right Button for Online Diagnostics Tool
STOP Key for selected Lisp Volume
Press Space Bar for Online Diagnostics Tool

```

Figure 3.2. The Bouncing White Rectangle

This is the idle state of the machine. Move ahead to Section 3.3 to boot Interlisp-D from this state.

## 3.2 Turning on the 1186

- (1) Insert the floppy disk labeled "INSTALLATION UTILITY" into the floppy drive. (If you have never handled floppy disks before, be sure to read "Care Of Floppys", Section 16.3.)
- (2) Turn on the machine by pressing the rocker switch to the 1 position, and pressing the red button labelled "B Reset" below the rocker switch.
- (3) A group of icons will appear at the bottom of the screen. Choose the second icon from the left, the one with a picture of a floppy,

by pressing the key labeled F2. This will boot the 1186 from the INSTALLATION UTILITY floppy.

- (4) When the question "Time offset from Greenwich?" appears, type "-5" for Eastern Standard Time (subtract one for each time zone westward), and **CR**. (See Figure 3.3.)
- (5) You will then be given three more questions. Answer them by hitting **CR** after each prompt.
- (6) When the 1186 prompts for the date and time, enter them in the MM/DD/YY HH:MM:SS format. For example, if the date was June 24, 1986 5 PM, you would enter: 6/24/86 17:00. This is illustrated for the 1186 in Figure 3.3.

```

Installer Version 1.0
Copyright (C) 1984, 1985 by Xerox Corporation. All rights reserved.
Product ID = 8AA41244EH = 252002021186 = 2-858-395-854
Memory Size = 3584K bytes

Calculating Time Server...Time is not set
Time zone offset from Greenwich[-12..12]: -5
Minute offset[0..59]: 0
First day of Daylight Savings Time[0..366]: 121
Last day of Daylight Savings Time[0..366]: 305
Please Enter the date and 24 hour time in form
      MM/DD/YY HH:MM:SS
Time: 6/24/86 9:16
Set time to 24-Jun-86 9:16:00
Okay? (Y/N): Y

MAIN MENU:
  1  LispInstallation

Enter choice number, then press RETURN: 1

Choices Available:
  1  Partition 10 Mb disk for 1 Lisp volume(14000 pages) and 1 LispFiles volume(1500 pages)
  2  Partition 20 Mb disk for 1 Lisp volume(30Mb) and 1 LispFiles volume(~10900 pages)
  3  Partition 40 Mb disk for 2 Lisp volumes(10Mb, 10Mb) and 1 LispFiles volume(~17500 pages)
  4  Partition 40 Mb disk for 2 Lisp volumes(16Mb, 8Mb) and 1 LispFiles volume(~13000 pages)
  5  Partition 40 Mb disk for 3 Lisp volumes(16Mb, 8Mb, 4Mb) and 1 LispFiles volume(750 pages)
  6  Partition 80 Mb disk for 4 Lisp volumes(32Mb, 8Mb, 8Mb, 4Mb) and 2 LispFiles volumes(7000, ~7000 pages)
  7  Partition 80 Mb disk for 4 Lisp volumes(16Mb, 16Mb, 16Mb, 8Mb) and 2 LispFiles volumes(7000, ~7000 pages)
  8  Initialize Workstation SystemTools Volume
  9  Boot SystemTools volume
 10  Erase SystemTools volume
 11  Erase LispFiles volume
 12  Physical Volume Scavenge
 13  Scavenge LispFiles Volume
 14  Scavenge SystemTools volume
 15  Boot from Lisp Volume
 16  Install Lisp Microcode Only
 17  Return to MAIN MENU

Enter choice number, then press RETURN: 9

```

**Figure 3.3.** The 1186 screen when booting the machine

- (7) A menu will appear, as in Figure 3.3. Choose number 9, Boot SystemTools Volume.
- (8) After a few minutes, you will see a black screen with a white bouncing rectangle like the one in Figure 3.2. This is the idle state of the machine. Move ahead to Section 3.3 to boot Interlisp-D from this state.

### 3.3 Loading Interlisp-D from the Hard Disk

- (1) Starting from the idle state of the machine, the black screen with the white bouncing rectangle, click the left button to get the Lisp Installation Tool. The screen will look like Figure 3.4

Document: {General, ErrorMessage}	Device: {PUP-F8, N8-F8, Floppy, LocalDisk}																					
File: Lisp.sysout	Volume Size = 32000	Free Pages = 10																				
Volume Menu: Lisp	Max. Vmem Size = 31969	Vmem Size = 13309																				
Volume Password:	User Password:	Domain:																				
Volume Boot file:																						
User:																						
Organization:																						
<table border="1"> <tr> <td>Sysin!</td> <td>Erase!</td> <td>Set Max Vmem!</td> <td>Make Script!</td> <td></td> </tr> <tr> <td>Sysin &amp; Boot!</td> <td>Remote List!</td> <td>Copy Vmem!</td> <td>Scavenge!</td> <td>Floppy!</td> </tr> <tr> <td>Boot!</td> <td>Help!</td> <td>Quit!</td> <td></td> <td></td> </tr> <tr> <td>Fetch Lisp Microcode!</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>			Sysin!	Erase!	Set Max Vmem!	Make Script!		Sysin & Boot!	Remote List!	Copy Vmem!	Scavenge!	Floppy!	Boot!	Help!	Quit!			Fetch Lisp Microcode!				
Sysin!	Erase!	Set Max Vmem!	Make Script!																			
Sysin & Boot!	Remote List!	Copy Vmem!	Scavenge!	Floppy!																		
Boot!	Help!	Quit!																				
Fetch Lisp Microcode!																						


Figure 3.4. The Lisp Installation Tool - With the mouse pointing to Copy Vmem

- (2) Choose CopyVMem by positioning the mouse over the word and clicking the left mouse button. Only click the left mouse once. If the screen clears, and the maintenance panel code reads 0915, you have made a mistake and need to do a 1 BOOT and start again at the beginning of this section.
- (3) Another window will appear over the in the lower left hand corner of the screen, like the one shown in Figure 3.5.

Source Volume: Lisp2	Destination Volume: Lisp
Source Volume Password:	Destination Volume Password:
Start!	
Quit!	

Figure 3.5. The window that will open when Copy Vmem! is chosen, with the mouse pointing to Start!

It may have values already filled in for Source and Destination. If so, skip this step, and the next one. Otherwise, position the mouse over the word Source. Press the middle mouse button (on two button mice, push the left and right buttons simultaneously) and hold it down. Choose LISP2 from the menu that will appear. This selection will display the name of the chosen Interlisp-D Volume after the word "Source".

- (4) Move the mouse cursor over the word "Dest. Volume." Press the middle mouse button and hold it down. Choose the volume where you want to reload lisp. If you are unsure which volume you want, choose LISP.
- (5) Move the mouse over the word "START!" and click the left mouse button. Click the left mouse button again when you get the mouse confirm cursor: .
- (6) When the copy completes, you will be asked if you want to boot the destination volume. Click the left button if you do. Continue with After Booting Interlisp-D, Section 3.4.

---

## 3.4 After Booting Lisp

---

At this point, you will be prompted to "Enter my pup host number in octal:". If there is no pup host number associated with your machine, simply type any number between 0 and 277, and press `CR`. If there already is a number there, simply type `CR`.

---

## 3.5 Restarting Lisp After Logging Out

---

To use this section, your machine should be in the idle state (the black screen with the white bouncing rectangle). If it is off, start at the beginning of the chapter. If it is white with windows, you are already in Lisp. If there is a bouncing Interlisp-D logo, just press the space bar to get back into Lisp.

- (1) Starting from the white bouncing rectangle (the idle state of the machine), click the left mouse button to start the InstallLispTool.
- (2) Choose the volume you want by moving the mouse cursor over the words "Volume Menu:". (See Figure 3.4.) Press the middle mouse button and hold it down. Choose the desired volume from the menu that appears. If you are unsure which volume you want, choose LISP. The menu will disappear, and the you chose will be displayed after the word "Volume:".
- (3) Start the volume by positioning the mouse cursor over the word "Boot" (This is one of the choices in Figure 3.4) and clicking the left button. Confirm this by clicking the left button once more. You have now booted Interlisp-D. Continue with After Booting LISP, Section 3.4, Page 3.5.

[This page intentionally left blank]

## 4. IF YOU HAVE A FILESERVER

---

If your lisp machine is connected to a network and a fileserver, there are some important differences that you need to be aware of. This chapter will point them out.

---

### 4.1 Turning on your 1108

---

- (1) Pull down the cover of the maintenance panel, and expose the four digit LED display.
- (2) To turn the machine on, simultaneously press both the B RESET and the ALT B buttons, then push the rocker switch to the 1 position.
- (3) Release only the B RESET button, and watch the red numbers count up 0000, 0001. As soon as it reads 0001 release the ALT B button. This is called a "1 BOOT".

If you accidentally kept the ALT B button depressed when 0001 rolled around, just continue to hold it down, and the machine will count around to 0001 again. (After counting to 0009 it goes back to 0001.)

- (4) Continue with the instructions for Turning on your 1108 on Section 3.1, Page 3.2, beginning with choosing number 12, "Boot SystemTools Volume" from the menu that appears.

---

### 4.2 Turning on your 1186

---

- (1) Turn on the machine by pressing the rocker switch to the 1 position, and pressing the red button labelled "B Reset" below the rocker switch.
- (2) A group of icons will appear at the bottom of the screen. Choose the third icon from the left, the one with a picture of a network, by pressing the key labeled F3. This will boot the 1186 from the network.
- (3) Continue with the instructions for Turning on the 1186 on Section 3.2, Page 3.3, beginning with choosing number 9, "Boot SystemTools Volume" from the menu that appears.



---

## 4.3 Location of Files

---

Both your files and system files could be located either on the local hard disk, on floppy, or on the file server. You can use files from the file server using the instructions in Section 21.2, Page 21.1.

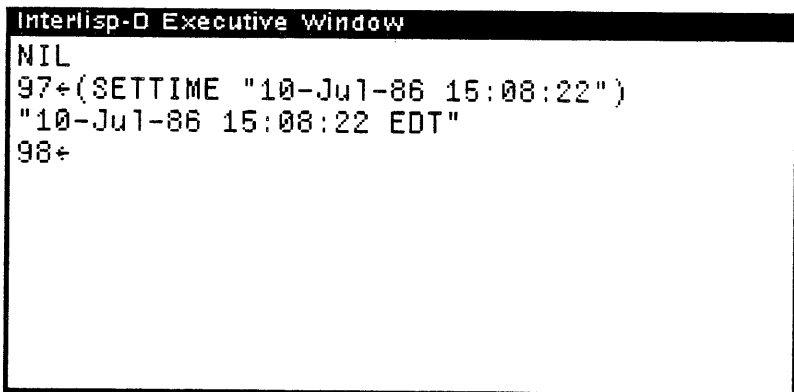
Fileservers cannot give you "random access" to your files. That means that, unlike the files on your local hard disk or on your floppy, a file from the fileserver that is not random access cannot be used by some functions. For example, TEdit (see Chapter 23) cannot use files that are not random access. This also means, for example, that if both your compiled and uncompiled program files are on the fileserver, and you need to make a change to one of the functions that is compiled, the system cannot load only this function for editing. Instead, you will have to load the whole uncompiled file.

---

## 4.4 The Timeserver

---

Because you are connected to a network, you are using a network utility called the timeserver. The timeserver sets the date and time on your lisp machine by getting it from another machine running on the network. This means that you do not have to set the time when booting your machine, but it also means that if the time was set incorrectly by another user, your machine will also have the incorrect time. You can always reset the time on your machine with the **SETTIME** function. To use it, type (**SETTIME** *date*), where *date* is a string such as the one shown in Figure 4.1.



```
Interlisp-D Executive Window
NIL
97+(SETTIME "10-Jul-86 15:08:22")
"10-Jul-86 15:08:22 EDT"
98+
```

Figure 4.1. Using the **SETTIME** function to set the date and time

# 5.LOGGING OUT AND TURNING THE MACHINE OFF

---

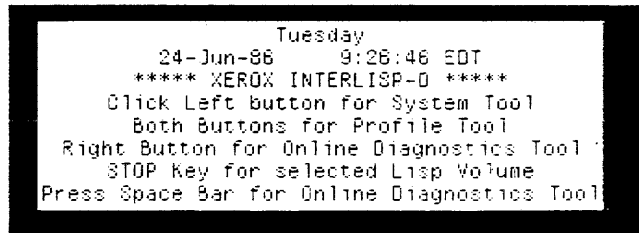
Logging out is the process of cleanly exiting from Interlisp-D. When you logout you greatly simplify starting Interlisp-D for your next session, because the entire current state of the system is saved. If you do not logout, you will need to reload Interlisp-D when you next login. If you logout improperly, you could lose all your work.

---

## 5.1 Logging Out

---

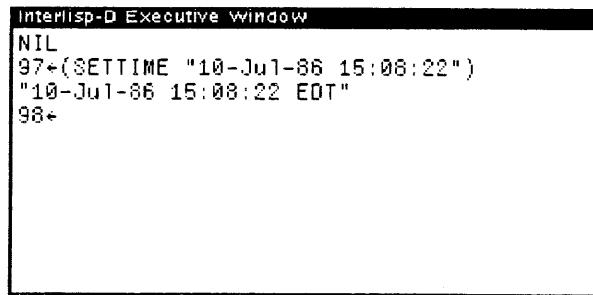
- (1) Make sure you have saved all your work using the **MAKEFILES** command. (See Section 11.6, Page 11.7.)
- (2) Before turning the machine off, remove the floppy from the floppy drive. Be sure to do this even if you need to reload LISP.
- (3) Log out by typing: (**LOGOUT**).
- (4) Wait until you see the bouncing white rectangle. The rectangle will look like the one in Figure 5.1.



**Figure 5.1.** The white rectangle that will bounce around the screen

If you have not set the time, the 1108 will not let you log out. This has happened if the screen is grey and the number on the maintenance panel is 0937. Wait at least 2 minutes to be sure that there is a problem. If nothing has happened, then:

- (1) Reenter LISP (do a 0 Boot by pressing both the B-RESET and the ALT-B buttons, immediately releasing the B-RESET button, and holding the ALT-B button until the maintenance panel reads 0000 and then releasing the ALT-B button.)
- (2) Set the time by typing  
(**SETTIME date**)  
where *date* is a string such as the one shown in Figure 5.2.

A screenshot of a terminal window titled "Interlisp-D Executive window". The window contains the following text:

```
NIL
97+(SETTIME "10-Jul-86 15:08:22")
"10-Jul-86 15:08:22 EDT"
98+
```

Figure 5.2. Using the **SETTIME** function to set the date and time

- (3) Logout again, and wait for the bouncing white rectangle.

---

## 5.2 Turning The Machine Off

---

Do not turn off the machine until you have successfully logged out. To turn the machine off, push the rocker switch to the 0 position. According to both the *1108 User's Guide* and the *1186 User's Guide*, you should wait at least 3 minutes before turning the processor back on.

Once you have logged in as per Chapters 3 or 4, you are in Interlisp-D. The functions you type into the Interlisp-D executive window will now execute, that is, perform the designated task. Please note that Interlisp-D is case-sensitive; often it matters whether text is typed in capital- or lower-case letters. The shiftlock key is above the left shift key; when it is pressed (on the 1186, the red LED will be on; on the 1108, the key will be depressed), everything typed is in capital letters.

You must type all Interlisp-D functions in parentheses. The Interlisp-D interpreter will read from the left parenthesis to the closing right parenthesis to determine both the function you want to execute, and the arguments to that function. Executing this function is called evaluation. When the function is evaluated it returns a value, which is then printed in the Interlisp-D executive window. This entire process is called the read-eval-print loop, and is how most LISP interpreters, including the one for Interlisp-D, run.

The prompt in Interlisp-D is a number followed by a left pointing arrow (see Figure 6.3). This number is the function's position on the History List -- a list that stores your interactions with the Interlisp-D interpreter. Type the function (**PLUS 3 4**), and notice the number the History List assigns to the function (the number immediately to the left of the arrow). Interlisp-D reads in the function and its arguments, evaluates the function, then prints the number 7.

In addition to this read-eval-print loop, there is also a "programmer's assistant". It is the programmer's assistant that prints the number as part of the prompt in the Interlisp-D executive window, and uses these numbers to reference the function calls typed after them.

When you issue commands to the programmer's assistant, you will not use parentheses as you do with ordinary function calls. You simply type the command, and some specification that indicates which item on the history list the command refers to. Some programmer's assistant commands are **FIX**, **REDO**, and **UNDO**. They are explained in detail below.

Programmer's assistant commands are useful only at the Interlisp-D top level, that is, when you are typing into the Interlisp-D executive window. They will not work in user-defined functions.

As an example use of the programmer's assistant, use **REDO** to redo your function call (**PLUS 3 4**). Type **REDO** (Note: programmer's assistant commands can be typed in either upper

or lower case) at the prompt, then specify the previous expression in one of the following ways:

- (1) When you originally typed in the function you now want to refer to, there was a History List number to the left of the arrow in the prompt. Type this number after the programmer's assistant command. This is the method illustrated in the following figure:

```

24←(PLUS 3 4)
7
25←REDO 24
7
26←

```

**Figure 6.1.** Using the programmer's assistant to **REDO** a function, when you know the its number on the history list

- (2) A negative number will specify the function call typed in that number of prompts ago. In this example, you would type in -1, the position immediately before the current position. This is shown in the following figure:

```

26←(PLUS 3 4)
7
27←REDO -1
7
28←

```

**Figure 6.2.** Typing a negative number after the programmer's assistant command will cause it use the function found on the History List that many positions before the current one.

- (3) You can also specify the function for the programmer's assistant with one of the items that was in that function call. The programmer's assistant will search backwards in the History List, and use the first function it finds that includes that item. For example, type **REDO PLUS** to have the function **(PLUS 3 4)** reevaluated.
- (4) If you type a Programmer's Assistant command without specifying a function (i.e., simply typing the command, then a **CR**), the Programmer's Assistant executes the command using the function entered at the previous prompt.

Here are a few more examples of using the programmer's assistant:

```

Interlisp-D Executive Window
NIL
54+ (PLUS 4 5)
9
55+REDO
9
56+?? -2

54. +(PLUS 4 5)
9

56+(SETQ B 'BOY)
BOY
57+B
BOY
58+UNDO SETQ
SETQ undone.
59+B

UNBOUND ATOM
B

60+REDO 56
BOY
61+B
BOY
62+

```

Figure 6.3. Some Applications of the Programmer's Assistant

## 6.1 If you make a Mistake

Editing in the Interlisp-D Executive Window is explained in Section 11.2, Page 11.2. In this section, only a few of the most useful commands will be repeated.

To move the caret to a new place in the command being typed, point the mouse cursor at the appropriate position, and press the left mouse button.

To move the caret back to the end of the command being typed, press CONTROL-X. (Hold the CONTROL key down, and type "X".)

The way you choose to delete an error may depend on the amount you need to remove. To delete:

- |                                |  |
|--------------------------------|--|
| The character behind the caret | simply press the backspace key   |
| The word behind the caret      | press CONTROL-W. (Hold the CONTROL key down, and type "W".)  |
| Any part of the command,       | first move the caret to the appropriate place in the command. Hold the right mouse button down and move the mouse cursor over the text. All of the blackened text between the caret and mouse cursor is deleted when you release the right mouse button. |

The entire command      press CONTROL-U. (Hold the CONTROL key down, and type "U".)  
Deletions can be undone. Just press the **UNDO** key.  
To add more text to the line, move the caret to the appropriate position, and just type. Whatever you type will appear at the caret.

The purpose of this chapter is to show you how to use menus. Many things can be done more easily using menus, and there are many different menus provided in the Interlisp-D environment. Some are "pop-up" menus, that are only available until a selection is made, then disappear until they are needed again. An example of one of these is the "background menu", that appears when the mouse is not in any window and the right mouse button is pressed. A background menu is shown in Figure 7.1. Yours may have different items in it.



Figure 7.1. A background menu.

Another common pop-up menu is the right button default window menu. This menu is explained more in Section 10.4, Page 10.3.

Other menus are more permanent, such as the menu that is always available for use with the Interlisp-D Filebrowser. This menu is shown in figure Figure 7.2, and the specifics of its use with the filebrowser is explained in Chapter 9).

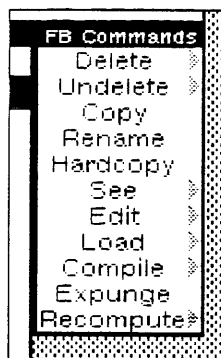


Figure 7.2. The menu that is available when using the Filebrowser



## 7.1 Making a Selection from a Menu

To make a selection from a menu, point with the mouse to the item you would like to select. If one of the mouse buttons is already pressed, the menu item should blacken. If it is a permanent menu, you must press the left mouse button to blacken the item. When you release the button, the item will be chosen. Figure 7.3 shows a menu with the item "Undo" chosen.

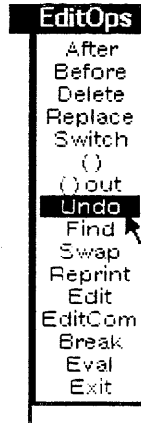


Figure 7.3. A menu with the item "Undo" chosen

## 7.2 Explanations of Menu Items

Many menu items have explanations associated with them. If you are not sure what the consequences of choosing a particular menu item will be, blacken the menu item, and do not release the left button. If the menu item has an explanation associated with it, the explanation will be printed in the prompt window. Figure 7.4 shows the explanation associated with the item "Snap" from the background menu.

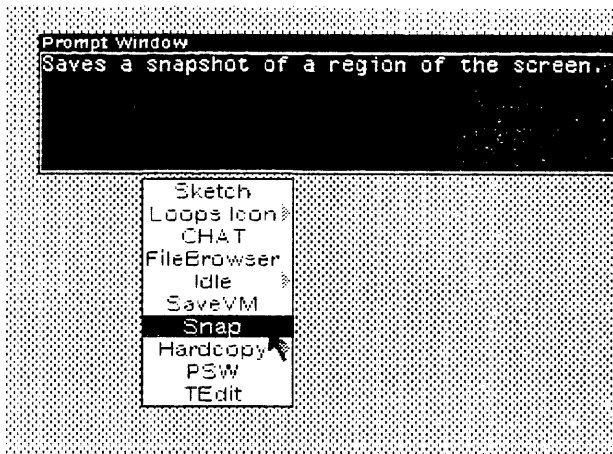
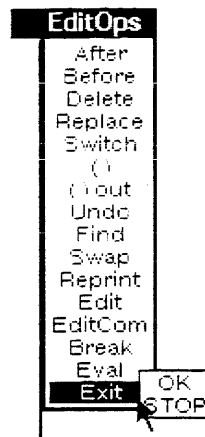


Figure 7.4. The explanation associated with the chosen item, "Snap", is displayed in the prompt window

## 7.3 Submenus

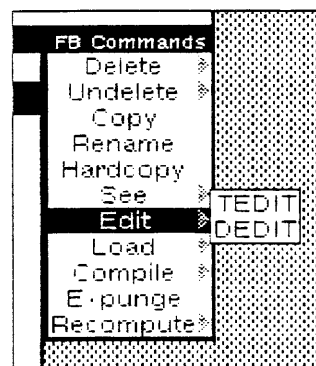
Some menu items have submenus associated with them. This means that, for these items, you can make even more precise choices if you would like to.

A submenu can also be found in one of two ways. One is to point to the item with the mouse cursor, and press the middle mouse button. If there is a submenu associated with that item, it will appear. (See Figure 7.5.)



**Figure 7.5.** The submenu associated with the menu item Exit - It appeared when the mouse cursor pointed to the menu item, and the middle mouse button was pressed.

A submenu can be indicated by a gray arrow to the right of the menu item, like the one to the right of the "Hardcopy" choice in Figure 7.1. To see the submenu, blacken the menu item, and move the mouse to follow the arrow. An example of this is shown in Figure 7.6. Choosing an item from a submenu is done in the same way as choosing an item from the menu. Any submenus that might be associated with the items in the submenu are indicated in the same way as the submenus associated with the items in the menu.



**Figure 7.6.** The submenu associated with the menu item Edit - It appeared when the menu item was blackened, and the mouse was moved to follow the gray arrow.

In summary, here are a few rules of thumb to remember about the interactions of the mouse, and system menus:

- Press the left mouse button to select an item of a menu
- Press the middle mouse button to get more options - one of the ways to find a submenu

- Press the right mouse button to see the default right button window menu, and the background menu

## 8.1 Types of Files

---

A program file, or lisp file, contains a series of expressions that can be read and evaluated by the Interlisp-D interpreter. These expressions can include function or macro definitions, variables and their values, properties of variables, and so on. How to save interlisp-D expressions on these files is explained in Section 11.6, Page 11.7. Loading a file is explained below, in Section 8.6, Page 8.4.

Not all files, however, have Interlisp-D expressions stored on them. For example, TEdit files (see Chapter 23) store text; sketches are stored on files made with the package Sketch (see Chapter 35), or can be incorporated into TEdit files. These files are not loaded directly into the environment, but are accessed with the package used to create them, such as TEdit or Sketch.

When you name a file, there are conventions that you should follow. These conventions allow you to tell the type of a file by the extension to its name. If a file contains:

Interlisp-D expressions,	it should not have an extension. For example, a file called "MYCODE" should contain Interlisp-D expressions;
compiled code,	it should have the extension ".DCOM". For example, a file called "MYCODE.DCOM" should contain compiled code;
a Sketch,	then its extension should be ".SKETCH". For example, a file called "MOUNTAINS.SKETCH" should contain a Sketch;
text,	it should have the extension ".TEDIT". For example, a file called "REPORT.TEDIT" should contain text that can be edited with the editor TEDIT.

## 8.2 Directories

---

This section focuses on how you can find files, and how you can easily manipulate files. To see all the files listed on a device, use the function **DIR**. For example, to see what files are stored on the hard disk, type

```
(DIR {DSK})
```

To see what files are stored on the floppy disk inside of the floppy drive, type

**(DIR {FLOPPY})**

Partial directory listings can be gotten by specifying a file name, rather than just a device name. The wildcard "\*" can be used to match any number of unknown characters. For example, the command

**(DIR {DSK}T\*)**

will list the names of all files stored on the hard disk that begin with the letter T. An example using the wildcard is shown in Figure 8.1

```
33+(DIR {DSK}<LISPPFILES>PRIMER>T*)
      {DSK}<LISPPFILES>PRIMER>
TAGREFS.TEDIT;2
TBLOCNT.TEDIT;1
NIL
34←
```

Figure 8.1. Using the function DIR with a wildcard

### 8.3 Directory Options

Various words can appear as extra arguments to the DIR command. These words give you extra information about the files.

- (1) **SIZE** displays the size of each file in the directory. For example, type

**(DIR {DSK} SIZE)**

- (2) **DATE** displays the creation date of each file in the directory. An example of this is shown in Figure 8.2

```
35+(DIR {DSK}<LISPPFILES>PRIMER>T* DATE)
      CREATIONDATE
      {DSK}<LISPPFILES>PRIMER>
TAGREFS.TEDIT;2      26-Jun-86 19:00:02
TBLOCNT.TEDIT;1     26-Jun-86 19:58:37
NIL
38←
```

Figure 8.2. An example using the directory option DATE

- (3) **DEL** deletes all the files found by the directory command

## 8.4 Subfile Directories

Subfile directories are very helpful for organizing files. A set of files that have a single purpose, for example all the external documentation files for a system, can be grouped together into a subfile directory.

To associate a subfile directory with a filename, simply include the desired subfile directory as part of the name of the file. Subfile directories are specified after the device name and before the simple filename. The first subfile directory should be between less-than and greater-than signs `< >`, with nested subdirectory names only followed by a greater-than sign `>`. For example:

```
{DSK}<Directory>SubDirectory>SubSubDirectory>...>filename
```

## 8.5 To See What Files Are Loaded

If you type `FILELST<CR>`, the names of all the files you loaded will display.

Type `SYSFILES<CR>`, to see what files are loaded to create the `SYSOUT`.

## 8.6 Simple Commands for Manipulating Files

The following commands will work with the `{FLOPPY}` and other devices, but have been shown with `{DSK}` for simplicity.

To have the contents of a file displayed in a window:

```
(SEE '{DSK}filename)
```

To copy a file: `(COPYFILE '{DSK}oldfilename '{DSK}newfilename)`

An example of this is shown in Figure 8.3

```
43+(COPYFILE 'TAGREFS.TEDIT 'PRIMERREFS.TEDIT)
{DSK}<LISPPFILES>PRIMER>PRIMERREFS.TEDIT;1
44+
```

Figure 8.3. An example of the use of the function `COPYFILE`

To delete a file: `(DELFILE '{DSK}filename)`

An example of this is shown in Figure 8.4.

```
48+(DELFILE 'SAMPLE.TEDIT)
{DSK}<LISPPFILES>PRIMER>SAMPLE.TEDIT;1
49+
```

Figure 8.4. The function `DELFILE`

To rename a file: `(RENAMEFILE '{DSK}oldfilename '{DSK}newfilename)`

"LOAD" a file: Files that contain Interlisp-D expressions can be loaded into the environment. That means that the information on them is read, evaluated, and incorporated into the Interlisp-D environment. To load a file, type:

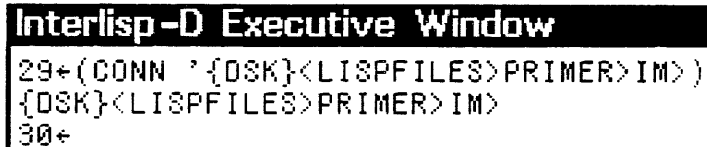
```
(LOAD '{DSK}filename)
```

When using these functions, always be sure to specify the full filename, including subfile directories if appropriate.

## 8.7 Connecting to a Directory

Often, each person or project has a subdirectory where their files are stored. If this is your situation, you will want any files you create to be put into this directory automatically. This means you should "connect" to the directory.

**CONN** is the Interlisp-D form that connects you to a directory. For example, **CONN** in the following figure:



```
Interlisp-D Executive Window
29+(CONN '{DSK}<LISPPFILES>PRIMER>IM)
{DSK}<LISPPFILES>PRIMER>IM)
30+
```

Figure 8.5. **CONN**ecting to the subdirectory "PRIMER"s subsubdirectory "IM"

connects you to the subsubdirectory **iM**, in the subdirectory **PRIMER**, in the directory **LISPPFILES**, on the device **DSK**. This information, the device and the directory names down to the subdirectory you want to be connected to, is called the "path" to that subdirectory. **CONN** expects the path to a directory as an argument.

Once you are connected to a directory, the command **DIR** will assume that you want to see the files in that directory, or any of its subdirectories.

Other commands that require a filename as an argument (e.g., **SEE**, above) will assume, if there is no path specified with the filename, that the file is in the connected directory. This will often save you typing.

## 8.8 File Version Numbers

When stored, each file name is followed by a semicolon and a number.

```
MYFILE.TEDIT;1
```

The number is the version number of the file. This is the system's way of protecting your files from being overwritten. Each time the file is written, a new file is created with a version number one

greater than the last. This new file will have everything from your previous file, plus all of your changes.

In most cases, you can exclude the version number when referencing the file. When the version is not specified, and there is more than one version of the file on that particular directory, the system generally uses your most recent version. An exception is the function **DELFILE**, which deletes the oldest version (the one with the lowest version number) if none is specified.



[This page intentionally left blank]

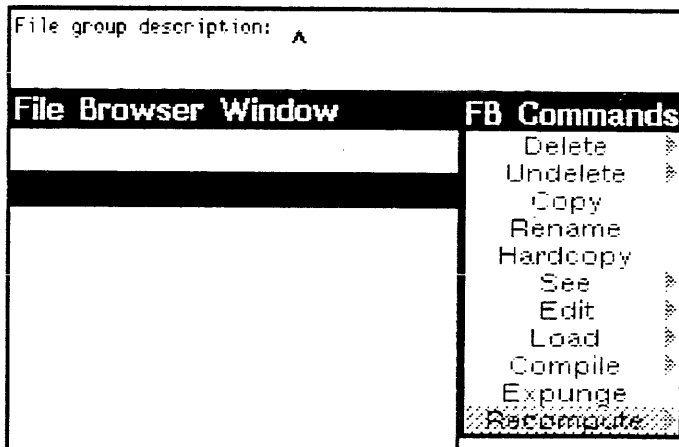
The FileBrowser is a Lisp Library Package that works with files stored on disk and floppy devices, and can be used as a file directory editor.

## 9.1 Calling the FileBrowser

Calling the FileBrowser with the device name, calls up the files stored on the device:

```
(FB '{DSK})
```

Another way to call a FileBrowser is to choose "FileBrowser" from the Background menu. You will be prompted for a description of the files to be included. (See Figure 9.1.) Simply type an asterisk ("\*"), then `CR`, to see all the files in the connected directory.



**Figure 9.1.** Prompt for the files to be included in the FileBrowser. Type an asterisk ("\*") and then `CR` to see all the files in the connected directory

These show a directory of the device in a window you can leave on the screen at all times. The parts of the FileBrowser window are shown below:

The Prompt window. The FileBrowser uses this area to prompt you to enter information.

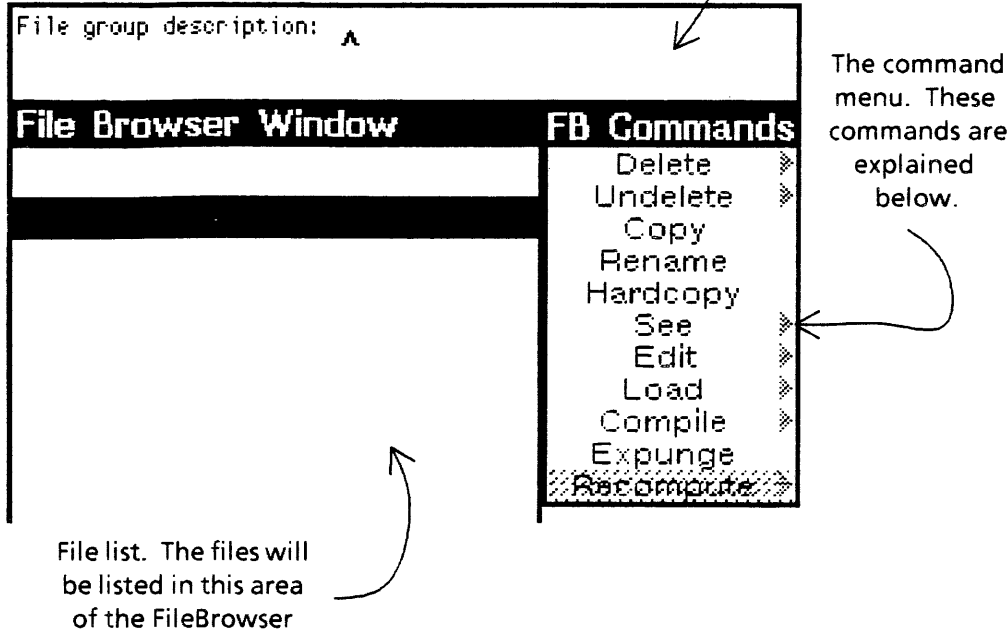


Figure 9.2. The parts of a FileBrowser

Now you do not need to continually type the directory command.

To use the FileBrowser, choose a file by pointing to the file with the mouse, and pressing the left or the middle mouse button. A small dark arrow will appear to the left of the file name. Choose a command from the menu at the right. In Figure 9.3, the files OCH11.MSS;2, OCH12.MSS;2, and OCH13.MSS;2 have been selected.

The left mouse button only allows you to choose one file at a time. Even if you choose other files, only the last file you picked with the left mouse button will remain marked as chosen. When you use the middle mouse button to choose a file, the file is added to those already picked.

To unpick an already chosen file, hold the CONTROL key while pressing the middle mouse button. On the 1186, the control key is the one marked EDIT or CTL to the left of the keyboard. On the 1108, the CONTROL key is also to the left of the keyboard, and is marked PROPS.

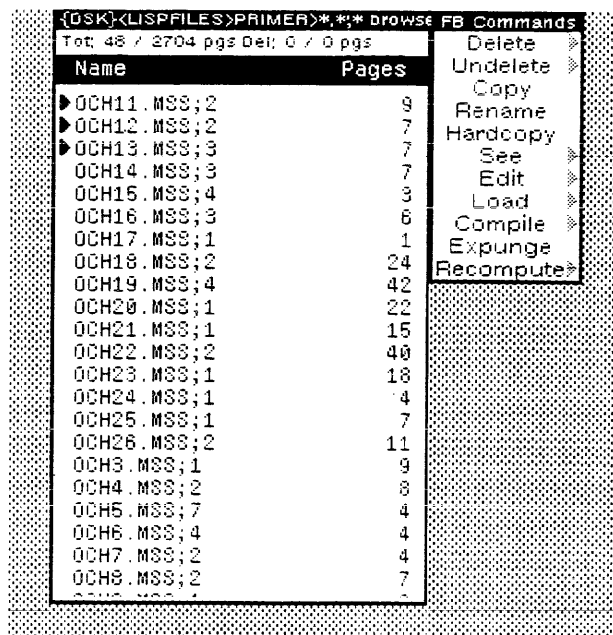


Figure 9.3. Files Chosen

A summary of the FileBrowser commands is shown below.

## 9.2 FileBrowser Commands

**Delete** In the FileBrowser, this command marks a file, or files, for deletion. (See Figure 9.4). These files are marked by a black line crossing through them. You may select and mark any number of files for deletion. **Delete** does not actually remove these files from the device. The **Expunge** command actually wipes out the files previously marked for deletion (see Figure 9.5).

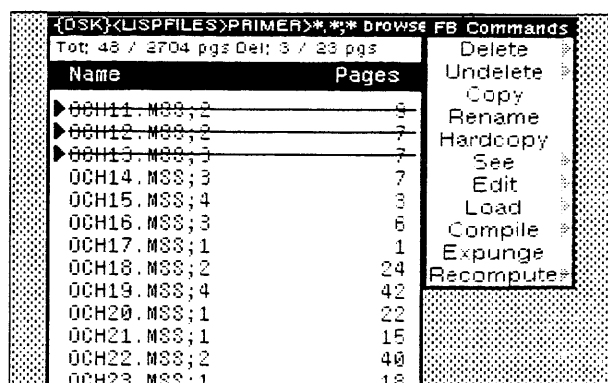


Figure 9.4. Files Marked for Deletion

The screenshot shows a window titled "{DSK}<LISPFILES>PRIMER>\*.\*;\* browse FB Commands". At the top, it displays "Tot: 48 / 2704 pgs Del: 3 / 23 pgs". Below this is a table with two columns: "Name" and "Pages". The table lists files from OCH11.MSS;2 to OCH25.MSS;1. A context menu is open over the file OCH14.MSS;3, showing options: Delete, Undelete, Copy, Rename, Hardcopy, See, Edit, Load, Compile, Expunge, and Recompute.

Name	Pages
OCH11.MSS;2	9
OCH12.MSS;2	7
OCH13.MSS;3	7
OCH14.MSS;3	7
OCH15.MSS;4	3
OCH16.MSS;3	6
OCH17.MSS;1	1
OCH18.MSS;2	24
OCH19.MSS;4	42
OCH20.MSS;1	22
OCH21.MSS;1	15
OCH22.MSS;2	40
OCH23.MSS;1	18
OCH24.MSS;1	4
OCH25.MSS;1	7

Figure 9.5. Files Marked For Deletion And Not Yet Expunged

- Undelete** undoes the delete command for one or more files. **Undelete** erases the black line through a file marked for deletion.
- Copy** This command copies the chosen file. The destination filename should be typed at a prompt that appears in the window above the FileBrowser. Wildcards do not work for this prompt. You must type the whole unquoted filename. If more than one file is chosen to be copied, you will be prompted for a directory name. The files will be copied into the directory you give, but with the same filenames as the ones they have in their original location.
- Rename** This command works much like the **Copy** command, but does not leave the original file. The chosen file will be renamed to the destination filename. You will be prompted, in the prompt window, for the destination filename. Give the complete unquoted filename. If more than one file is chosen to be renamed, you will be prompted for a directory name. The files will be moved into the directory you give.
- Hardcopy** If you do not have a laser hardcopy device, using this command will cause an error. Otherwise, it gives a hardcopy of the file.
- See** Shows you a file in a window. To use this command, choose a single filename, then the **See** command. You will be prompted for a window. Each time the **See** command is chosen, a new window is opened to display the file.
- Edit** Calls the editor with the file as input. If the file is an executable one, (i.e. Lisp code as opposed to a documentation file), only the **FILECOMS** list will be edited. The **FILECOMS** list is the list of variables, lists, and functions that are contained on that file. FileBrowser will first load it and then allow you to edit the **FILECOMS**.
- Load** Choose a file with the left mouse button, or a group of files with the middle mouse button. Once the filenames have been blackened, choose the **Load** command to load them all into Interlisp-D.
- Compile** This command calls the file compiler **TCOMPL**, with the chosen filename(s) as arguments. **TCOMPL** compiles a file found on a storage device (**{FLOPPY}** or **{DSK}**), not the functions defined in the Interlisp-D image. If any functions on a loaded file have been changed, run the function (**MAKEFILE 'filename**) to

write the current version before compiling it. Files do not have to be loaded to use the compile command.

**Expunge** **Expunge** completely deletes all the marked files from the directory. This allows you to remove unwanted files from your floppy.

**Recompute** Choose this command when you know that the directory has been changed and should be reread, for example, after inserting a new floppy disk.

[This page intentionally left blank]

# 10. THOSE WONDERFUL WINDOWS!

A window is a designated area on the screen. Every rectangular box on the screen is a window. While Interlisp-D supplies many of the windows (such as the Interlisp-D executive window), you may also create your own. Among other things, you will type, draw pictures, and save portions of your screen with windows.

## 10.1 Windows provided by Interlisp-D

Two important windows are available as soon as you enter the Interlisp-D environment. One is the Interlisp-D executive window, the main window where you will run your functions. It is the window that the caret is in when you turn on your machine, and load Interlisp-D. It is shown in Figure 10.1.

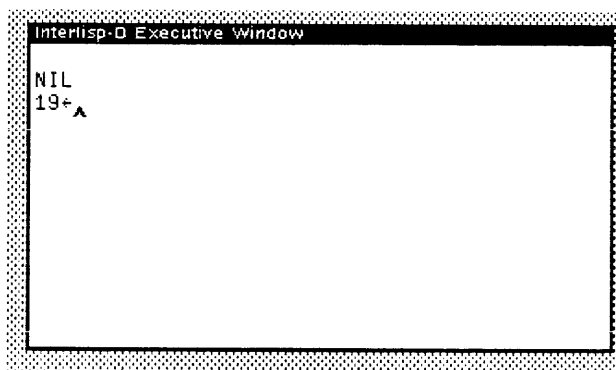


Figure 10.1. Interlisp-D Executive Window

The other window that is open when you enter Interlisp-D is the "Prompt Window". It is the long thin black window at the top of the screen. It displays system prompts, or prompts you have associated with your programs. (See Figure 10.2.)

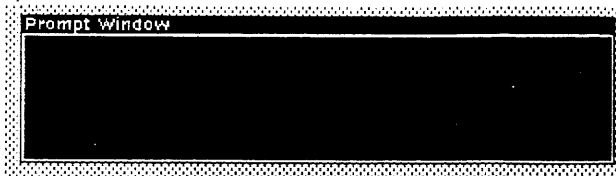


Figure 10.2. Prompt Window

Other programs, such as the editors, also use windows. These windows appear when the program starts to run, and close (no longer appear on the screen) when the program is done running.



## 10.2 Creating a window

To create a new window, type: **(CREATEW)**. The mouse cursor will change, and have a small square attached to it. (See Figure 10.3.)

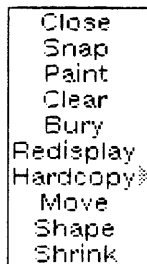


**Figure 10.3.** The mouse cursor asking you to *sweep out* a window.

There may be a prompt in the prompt window to create a window. Press and hold the left mouse button. Move the mouse around, and notice that it sweeps out a rectangle. When the rectangle is the size that you'd like your window to be, release the left mouse button. More specific information about the creation of windows, such as giving them titles and specifying their size and position on the screen when they are created, is given in Section 27.1.2, Page 27.2.

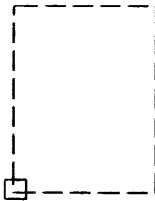
## 10.3 The Right Button Default Window Menu

Position the cursor inside the window you just created, and press and hold the right mouse button. A menu of commands should appear (do not release the right button!), like the one in Figure 10.4. To execute one of the commands on this menu, choose the item. Making a choice from a menu is explained in Section 7.1, Page 7.2.



**Figure 10.4.** The Right Button Default Window Menu

As an example, select "Move" from this menu. The mouse cursor will become a ghost window (just an outline of a window, the same size as the one you are moving), with a square attached to one corner, like the one shown in Figure 10.5.



**Figure 10.5.** The mouse cursor for moving a window

Move the mouse around. The ghost window will follow. Click the left mouse button to place the window in a new location.

Choose "Shape", and notice that you are prompted to sweep out another window. Your original window will have the shape of the window you sketch out.

## 10.4 An explanation of each menu item

The meaning of each right button default window menu item is explained below:

Close	removes the window from the screen;
Snap	copies a portion of the screen into a new window;
Paint	allows drawing in a window;
Clear	clears the window by erasing everything within the window boundaries;
Bury	puts the window beneath all other windows that overlap it;
Redisplay	redisplays the window contents;
Hardcopy	sends the contents of the window to a printer or to a file;
Move	allows the window to be moved to a new spot on the screen;
Shape	repositions and/or reshapes the window;
Shrink	reduces the window to a small black rectangle called an icon. (See Figure 10.6.)



Figure 10.6. An example icon

Expand	changes an icon back to its original window. Position the mouse cursor on the icon, depress the right button, and select Expand. Or, just button the icon with the middle mouse button.
--------	---

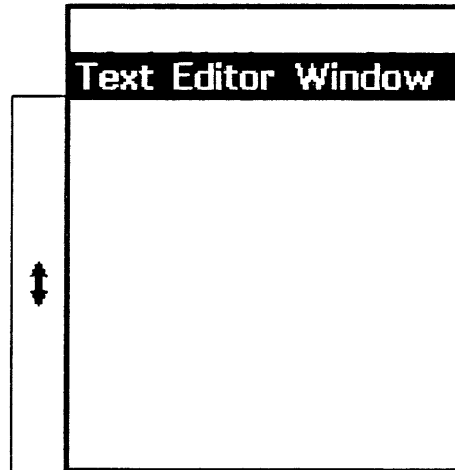
These right-button default window menu selections are available in most windows, including the Interlisp-D Executive window. When the right button has other functions in a window (as in an editor window), the right button default window menu should be accessible by pressing the Right button in the black border at the top of the window.

## 10.5 Scrollable Windows

Some windows in Interlisp-D are "scrollable". This means that you can move the contents of the window up and down, or side to side, to see anything that doesn't fit in the window.

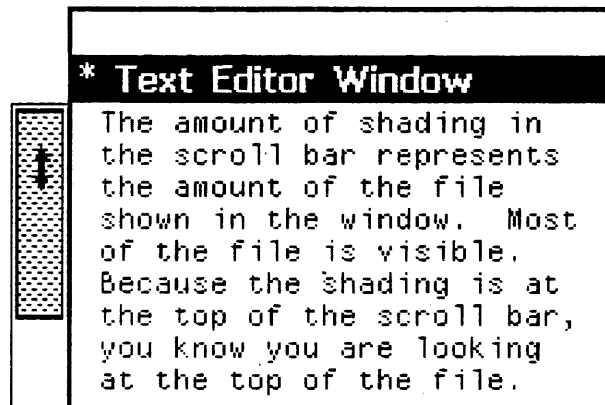
Point the mouse cursor to the left or bottom border of a window. If the window is scrollable, a "scroll bar" will appear.

The mouse cursor will change to a double headed arrow. (See Figure 10.7.)



**Figure 10.7.** The scroll bar of a scrollable window. The mouse cursor changes to a double headed arrow.

The scroll bar represents the full contents of the window. The example scroll bar is completely white because the window has nothing in it. When a part of the scroll bar is shaded, the amount shaded represents the amount of the window's contents currently shown. If everything is showing, the scroll bar will be fully shaded. (See Figure 10.8.) The position of the shading is also important. It represents the relationship of the section currently displayed to the full contents of the window. For example, if the shaded section is at the bottom of the scroll bar, you are looking at the end of the file.



**Figure 10.8.** The amount of shading in the scroll bar represents the amount of the file shown in the window. Most of the file is visible. Because the shading is at the top of the scroll bar, you know you are looking at the top of the file.

When the scroll bar is visible, you can control the section of the window's contents displayed:

- To move the contents higher in the window (scroll the contents "up" in the window), press the left button of the mouse, the mouse cursor changes to look like this:



**Figure 10.9.** Upward scrolling cursor.

The contents of the window will scroll up, making the line that the cursor is beside the topmost line in the window.

- To move the contents lower in the window (scroll the contents "down" in the window), press the right button of the mouse, and the mouse cursor changes to look like this:



**Figure 10.10.** Downward scrolling cursor.

The contents of the window scroll down, moving the line that is the topmost line in the window to beside the cursor.

- To show a specific section of the window's contents, remember that the scroll bar represents the full contents of the window. Move the mouse cursor to the relative position of the section you want to see (e.g., to the top of the scroll bar if you want to see the top of the window's contents.). Press the middle button of the mouse. The mouse cursor will look like this:



**Figure 10.11.** Proportional scrolling cursor.

When you release the middle mouse button, the window's contents at that relative position will be displayed.

## 10.6 Other Window Functions

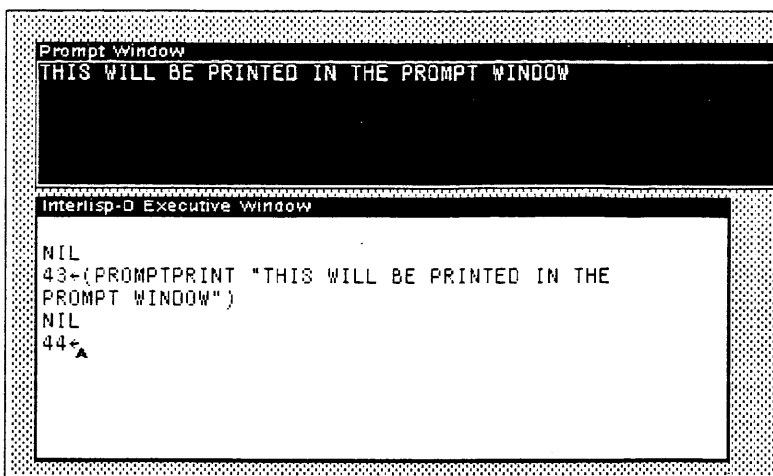
### 10.6.1 PROMPTPRINT

Prints an expression to the black prompt window.

For example, type

**(PROMPTPRINT "THIS WILL BE PRINTED IN THE PROMPT WINDOW")**

The message will appear in the prompt window. (See Figure 10.12.)



**Figure 10.12.** PROMPTPRINTing

## 10.6.2 WHICHW

---

Returns as a value the name of the window that the mouse cursor is in.

**(WHICHW)** can be used as an argument to any function expecting a window, or to reclaim a window that has no name (that is not attached to some particular part of the program.).

This chapter explains how to define functions, how to edit them, and how to save your work.

## 11.1 Defining Functions

DEFINEQ can be used to define new functions. The syntax for it is:

```
(DEFINEQ (<functionname> (<parameter-list>
                          <body-of-function>))
```

New functions can be created with DEFINEQ by typing directly into the Interlisp-D executive window. Once defined, a function is a part of the Interlisp-D environment. For example, the function EXAMPLE-ADDER is defined in Figure 11.1.

```
Interlisp-D Executive
NIL
48+(DEFINEQ (EXAMPLE-ADDER (A B C)
                        (PRINT "THE SUM OF THE
                        THREE NUMBERS IS ")
                        (IPLUS A B C)))
(EXAMPLE-ADDER)
47+
```

Figure 11.1. Defining the function EXAMPLE-ADDER

Now that the function is defined, it can be called from the Interlisp-D executive window:

```
Interlisp-D Executive
NIL
49+(EXAMPLE-ADDER 3 4 5)
"THE SUM OF THE THREE NUMBERS IS "
12
50+
```

Figure 11.2. After EXAMPLE-ADDER is defined, it can be executed. The function returns 12, after printing out the message.

Functions can also be defined using the editor DEdit described above. To do this, simply type

```
(DF function-name)
```

You will be asked whether you would like to edit a Dummy definition. A dummy definition is a standard template for your function definition. Answer by typing Y for Yes, and you will be able to define the function in the editor. (See Figure 11.3. The use of the editor is explained in Section 11.3, Page 11.4.)

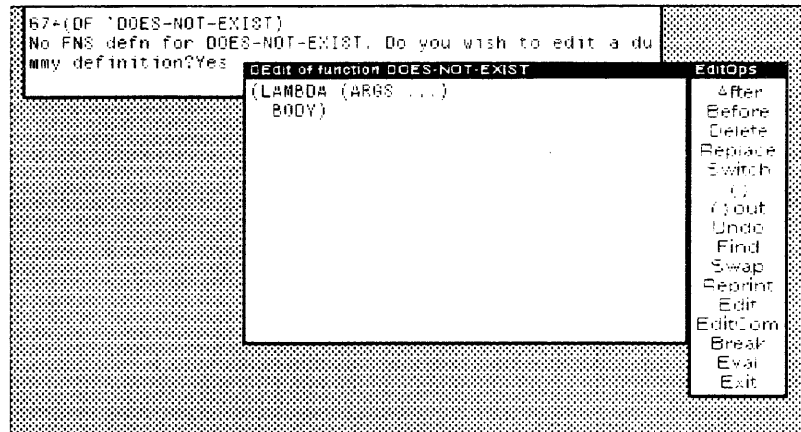


Figure 11.3. Using DEdit to define a function

## 11.2 Simple Editing in the Interlisp-D Executive Window

First, type in an example function to edit:

```
51←(DEFINEQ (YOUR-FIRST-FUNCTION (A B)
  (if (GREATERP A B)
    then '(THE FIRST IS GREATER)
    else '(THE SECOND IS GREATER))))
```

To run the function, type `(YOUR-FIRST-FUNCTION 3 5)`.

```
52←(YOUR-FIRST-FUNCTION 3 5)
(THE SECOND IS GREATER)
```

Now, let's alter this. Type:

```
53←FIX 51 CR
```

Note that your original function is redisplayed, and ready to edit. (See Figure 11.4.)

```

Interlisp-D Executive
NIL
58←FIX 51
*(DEFINED
  [YOUR-FIRST-FUNCTION
    (A B) (* edited:
"31-Dec-00 19:28")
  (IF (GREATERP A B)
      THEN (QUOTE (THE FIRST IS
                   GREATER))
      ELSE (QUOTE (THE SECOND IS
                   GREATER]))

```

Figure 11.4. Using **FIX** to edit a function

- Move** the text cursor to the appropriate place in the function by positioning the mouse cursor and pressing the left mouse button.
- Delete** text by moving the caret to the beginning of the section to be deleted. Hold the right mouse button down and move the mouse cursor over the text. All of the blackened text between the caret and mouse cursor is deleted when you release the right mouse button.
- If you make a mistake** deletions can be undone. On an 1108, press the **OPEN** key to **UNDO** the deletion. On an 1108, press the **UNDO** key on the keypad to the left of the keyboard.

Now change **GREATER** to **BIGGER**:

- (1) Position the mouse cursor on the G of GREATER, and click the left mouse button. The text cursor is now where the mouse cursor is.
- (2) Next, press the right mouse button and hold it down. Notice that if you move the mouse cursor around, it will blacken the characters from the text cursor to the mouse cursor. Move the mouse so that the word "GREATER" is blackened.
- (3) Release the right mouse button and GREATER is deleted.
- (4) Without moving the cursor, type in BIGGER.
- (5) There are two ways to end the editing session and run the function. One is to type CONTROL-X. (Hold the CONTROL key down, and type "X".) Another is to move the text cursor to the end of the line and **CR**. In both cases, the function has been edited!

Try the new version of the function by typing:

```
59←(YOUR-FIRST-FUNCTION 8 9)
(THE SECOND IS BIGGER)
```

and get the new result, or you can type:

```
59←REDO 52CR
(THE SECOND IS BIGGER)
```



### 11.3 Using The List Structure Editor

If the function you want to edit is not readily available (i.e. the function is not in the Interlisp-D Executive window, and you can't remember the history list number, or you simply have a lot of editing), use the List Structure Editor, often called DEdit. This editor is evoked with a call to DF:

81←(DF YOUR-FIRST-FUNCTION)

Your function will be displayed in an edit window, as in Figure 11.5.

If there is no edit window on the screen, you will be prompted to create a window. As before, hold the left mouse button down, move the mouse until it forms a rectangle of an acceptable size and shape, then release the button. Your function definition will automatically appear in this edit window.

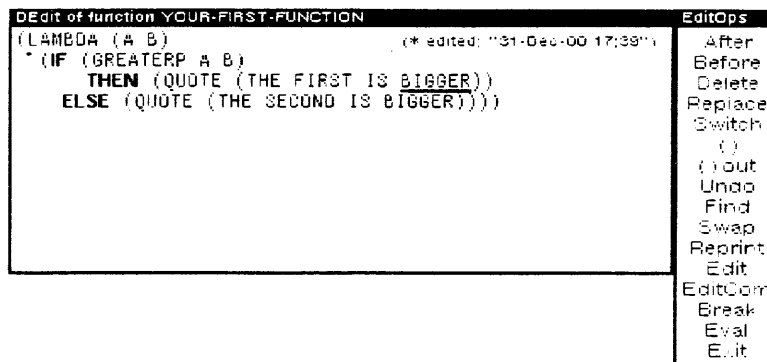


Figure 11.5. An Edit Window

Many changes are easily done with the structure editor. Notice that by pressing the left mouse button, different expressions are underlined. Underline **BIGGER** as in Figure 11.5. Release the left mouse button.

To add an expression that doesn't appear in the edit window, (i.e. it can't simply be underlined), just type it in. Doing this will create an edit buffer below the DEdit window. For example, type **LARGER** and hit **CR**. (Remember to **CR**! You won't be able to do anything in the editor until you **CR** - this can fool you at first, so beware.) A new window opens up at the bottom for the new expression. (See Figure 11.6.)

**LARGER** now has the bold line underneath it, while **BIGGER** has a dotted line.

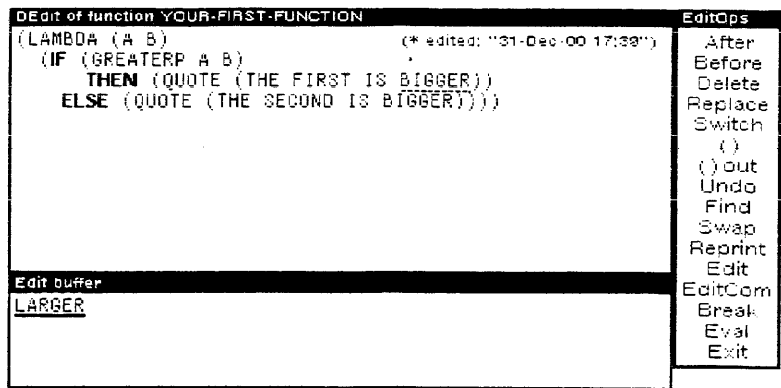


Figure 11.6. Edit Window with Edit Buffer

DEdit keeps track of items you have chosen by using a stack. The underlines tell you the order of the items on the stack. The solid underline indicates the item on the top of the stack; the dotted underline indicates the second to the top. (**BIGGER** was pushed on first. When **LARGER** was pushed on, **BIGGER** became the second element in the "stack", and **LARGER** the first.)

Many commands operate with two items on the stack. Some of them are listed below:

**After** pops the stack, and adds this top item (in this example, **LARGER**) to the edit window after the second item on the stack (in this example, **BIGGER**). The item that was at the top of the stack, **LARGER**, will now appear in both the original and the new position.

**Before** pops the stack, and adds this top item (in this example, **LARGER**) to the edit window before the second item on the stack. (See Figure 11.7.)

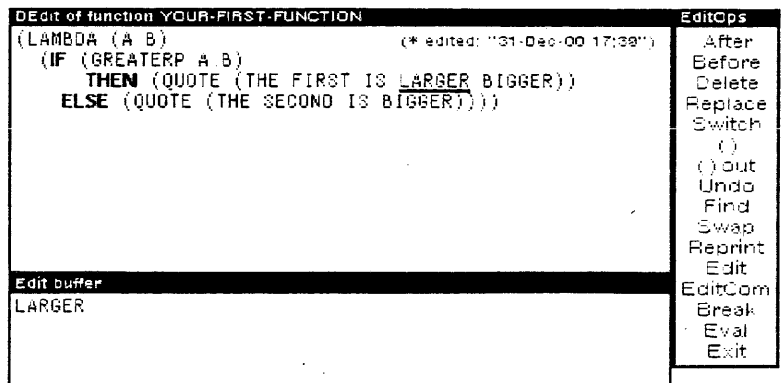


Figure 11.7. The command **Before** is chosen; the word **LARGER** appears before the word **BIGGER**

**Replace** pops the stack, and substitutes this top item for the second item on the stack.

**Switch** changes the position of the first and second items on the stack in the edit window.

**Find** pops the stack, and searches this top expression for an occurrence of the second item on the stack. If the item is found, it is underlined with a solid line, that is, pushed on the stack. To find the next occurrence, simply choose "Find" again. If the expression is not found, the prompt window will blink, and a

message that the item was not found will appear. (See Figure 11.8 for an example of an item, the atom **THIRD**, not appearing in the function, **YOUR-FIRST-FUNCTION**.)

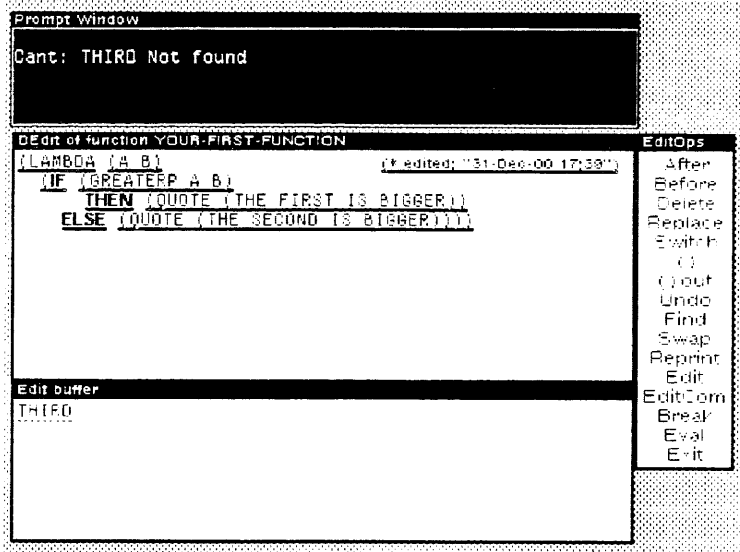


Figure 11.8. The atom **THIRD** is not in the function being edited

**Swap** changes places, on the stack, of the first and second items on the stack. The edit window does not change, except that the expression that had a solid underline now has a dotted underline, and vice versa.

**Delete** works on only the top item of the stack. Delete removes the solid underlined expression from the edit window.

**Undo** undoes the last editor command.

Completing the example begun earlier, here's how to have the word **LARGER** that you typed into the edit buffer appear in place of the **BIGGER** that you selected from the **DEdit** window: select the **SWITCH** command. Notice that the two items are switched, and the stack is popped. Now select **EXIT** and to leave the editor, and your function will again be redefined.

### 11.3.1 Commenting Functions

Text can be marked as a comment by nesting it in a set of parentheses with a star immediately after the left parenthesis.

(\* This is the form of a comment)

Inside an editor window, the comment will be printed in a smaller font and may be moved to the far right of the code. Sometimes, however, centered comments are more appropriate. To center a comment, type **\*\*** after the left parenthesis.

(\* \* This comment will not be moved to the far right of the code, but will be centered)

It is also possible to insert linebreaks within a comment. A dash should be placed in the comment wherever a carriage return is needed. This feature allows several comments to be placed inside one set of parentheses.

(\* This comment will be typed onto two lines. -

especially useful if you want to space your comments)

There are other editor commands which can be very useful. To learn about them, read to the *Interlisp-D Reference Manual*, Volume 2, Section 16, on DEDIT.

---

## 11.4 File Functions and Variables - How to See Them and Save Them

---

With Interlisp-D, all work is done inside the "Lisp Environment". There is no "Operating System" or "Command Level" other than the Interlisp-D Executive Window. All functions and data structures are defined and edited using normal Interlisp-D commands. This section describes tools in the Interlisp-D environment that will keep track of any changes that you make in the environment that you have not yet saved on files, such as defining new functions, changing the values of variables, or adding new variables. And it then has you save the changes in a file you specify.

---

## 11.5 File Variables

---

Certain system-defined global variables are used by the file package to keep track of the environment as it stands. You can get system information by checking the values of these variables. Two important variables follow.

- **FILELST** evaluates to a list, all files that you have loaded into the Interisp-D environment.
- **filenameCOMS** (Each file loaded into the Lisp environment has associated with it a global variable, whose name is formed by appending "COMS" to the end of the filename.) This variable evaluates to a list of all the functions, variables, bitmaps, windows, and so on, that are stored on that particular file.

For example, if you type:

```
MYFILECOMS
```

the system will respond with something like:

```
((FNS YOUR-FIRST-FUNCTION )
 (VARS))
```

---

## 11.6 Saving Interlisp-D on Files

---

The functions (**FILES?**) and (**MAKEFILE 'filename**) are useful when it is time to save function, variables, windows, bitmaps, records and whatever else to files.

**(FILES?)** displays a list of variables that have values and are not already a part of any file, and then the functions that are not already part of any file.

Type:

**(FILES?)**

the system will respond with something like:

```
the variables: MY-VARIABLE CURRENT.TURTLE...to be dumped.
the functions: RIGHT LEFT FORWARD BACKWARD CLEAR-SCREEN...to
be dumped.
want to say where the above go?
```

If you type **Y**, the system will prompt with each item. There are three options:

- (1) To save the item, type the filename (unquoted) of the file where the item should be placed. (This can be a brand new file or an existing file.)
- (2) To skip the item, without removing it from consideration the next time **(FILES?)** is called, type **^C**. This will allow you to postpone the decision about where to save the item.
- (3) If the item should not be saved at all, type **]**. **Nowhere** will appear after the item.

Part of an example interaction is shown in the following figure:

```
Interlisp-D Executive Window
NIL
81+(FILES?)
the variables: MY-VAR...to be dumped.
the functions: MY-SECOND-FUNCTION,
               YOUR-FIRST-FUNCTION
...to be dumped.
want to say where the above go ? Yes
(variables)
MY-VAR Nowhere
(functions)
MY-SECOND-FUNCTION File name: EXAMPLE
```

**Figure 11.9.** Part of an interaction using the function **FILES?**

**(FILES?)** assembles the items by adding them to the appropriate file's COMS variable. (See Section 11.5, Page 11.7.) **(FILES?)** does NOT write the file to secondary storage (disks or floppies). It only updates the global variables discussed in Section 11.5.

**(MAKEFILE 'filename)** actually writes the file to secondary storage. Files should only be written when the time is set. If the time is not set, you will run into problems, such as not being able to copy your file. To check the time, type

**(DATE)**

If the date is correct, you can safely use **MAKEFILE**. If it is not correct, set the time with the function **SETTIME**. To use it, type **(SETTIME date)**, where *date* is a string such as the one shown in Figure 11.10.

```
Interlisp-D Executive Window
NIL
97+(SETTIME "10-Jul-86 15:08:22")
"10-Jul-86 15:08:22 EDT"
98+
```

Figure 11.10. Using the **SETTIME** function to set the date and time

Once the time is set correctly, use the function **MAKEFILE**. Type:

```
(MAKEFILE 'MY.FILE.NAME)
```

and the system will create the file. The function returns the full name of the file created. (i.e. {DSK}MY.FILE.NAME.;1).

Note: Files written to {DSK} are permanent files. They can be removed only by the user deleting them or by reformatting the disk.

Other file manipulation functions can be found in Section 8.6, Page 8.3.

[This page intentionally left blank]

Interlisp-D has a number of global variables that control the environment of your 1108 or 1186. Global variables make it easy to customize the environment to fit your needs. One way to do this is to develop an "INIT" file. This is a file that is loaded when you log on to your machine. You can use it to set variables, load files, define functions, and any other things that you want to do to make the Interlisp-D environment suit you.

Your Init file could be called INIT, INIT.LISP, INIT.USER, or whatever the convention is at your site. There is no default name preferred by the system, it just looks for the files listed in the variable **USERGREETFILES**, (see below). Check to see what the preference is at your site. Put this file in your directory. Your directory name should be the same as your login name.

The INIT file is loaded by the function **GREET**. **GREET** is normally run when Interlisp-D is started. If this is not the case at your site, or you want to use the machine and Interlisp-D has already been started, you can run the function **GREET** yourself. If your user name was, for example, TURING, then you would type:

```
(GREET 'TURING)
```

This does a number of things, including undoing any previous greeting operation, loading the site init file, and loading your init file. Where **GREET** looks for your INIT file depends on the value of the variable **USERGREETFILES**. The value of this variable is set when the system's **SYSOUT** file is made, so check its value at your site! For example, its value could be:

```
Interlisp-D Executive Window
NIL
3+USERGREETFILES
({{DSK}<LISPPFILES> USER >INIT.LISP)
  {{DSK}<LISPPFILES>INIT.LISP)
  {{FLOPPY}INIT.LISP)
  {{DSK}<LISPPFILES> USER >INIT.USER)
  {{DSK}<LISPPFILES>INIT.USER)
  {{FLOPPY}INIT.USER)
  {{DSK}<LISPPFILES> USER >INIT)
  {{DSK}<LISPPFILES>INIT)
  {{FLOPPY}INIT))
4-
```

Figure 12.1. A possible value of **USERGREETFILES**.

In each place you see, "> USER >", the argument passed to **GREET** is substituted into the path. This is your login name if you are just starting Interlisp-D. For example, the first value in the list would have the system check to see whether there was a file, **{DSK}<LISPPFILES>TURING>INIT.LISP**. No error is generated if you do not have an INIT file, and none of the files in **USERGREETFILES** are found.



## 12.1 Making an Init File

As described in Section 11.5, Page 11.7, each Interlisp-D program file has a global variable associated with it, whose name is formed by appending "COMS" to the end of the root filename. For any of the standard INIT file names, the variable INITCOMS is used. To set up an init file, begin by editing this variable. First, type:

```
(SETQ INITCOMS '((VARS)))
```

Now, to edit the variable, type:

```
(DV INITCOMS)
```

A DEdit window will appear. This DEdit window is the same as the one called with the function DF, and described in Section 11.3, Page 11.4. This chapter will assume that you know how to use the structure editor, DEdit.

The COMS variable is a list of lists. The first atom in each internal list specifies for the file package what types of items are in the list, and what it is to do with them. This section will deal with three types of lists: VARS, FILES, and P. Please read about others in the *Interlisp-D Reference Manual*, Volume II, Chapter 17.

The list that begins with "VARS" allows you to set the values of variables. For example, one global variable is called DEditLinger. Its default value is T, and means that the DEdit window won't close after you exit DEdit. If it is set to NIL, then the DEdit window will be closed when you exit DEdit. To set it to NIL in your INIT file, edit the VARS list so that it looks like this:

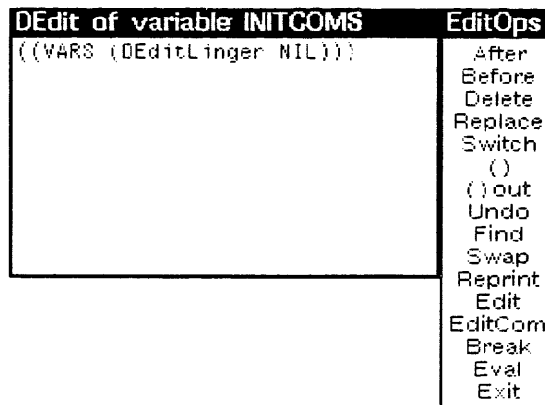


Figure 12.2. Setting the variable DEditLinger in INITCOMS.

Notice that inside the vars list, there is yet another list. The first item in the list is the name of the variable. It is bound to the value of the second item. There are many other variables that you can set by adding them to the VARS list. Some of these variables are described in Chapter 43, and many others can be found in the *Interlisp-D Reference Manual*.

If you want to automatically load files, that can be done in your init file also. For example, if you always want to load the Library file SPY.DCOM, you can load it by editing the INITCOMS variable to list the appropriate file in the list starting with FILES:

DEdit of variable INITGOMS	EditOps
<pre>((VARS (DEditLinger NIL)) (FILESPY))</pre>	After Before Delete Replace Switch () ()out Undo Find Swap Reprint Edit EditCom Break Eval Exit

Figure 12.3. INITGOMS changed to load the file SPY.DCOM

Other files can also be added by simply adding their names to this FILES list.

Another list that can appear in a COMS list begins with "P". This list contains Interlisp-D expressions that are evaluated when the file is loaded. Do not put DEFINEQ expressions in this list. Define the function in the environment, and then save it on the file in the usual way (see Section 11.6, Page 11.7).

One type of expression you might want to see here, however, is a FONTCREATE function (see Section 31.2, Page 31.2). For example, if you want to use a Helvetica 12 BOLD font, and there is not a fontdescriptor for it normally in your environment, the appropriate call to FONTCREATE should be in the "P" list. The INITGOMS would look like this:

DEdit of variable INITGOMS	EditOps
<pre>((VARS (DEditLinger NIL)) (FILESPY) (P (FONTCREATE (QUOTE                 HELVETICA                 12                 (QUOTE BOLD))))))</pre>	After Before Delete Replace Switch () ()out Undo Find Swap Reprint Edit EditCom Break Eval Exit

Figure 12.4. INITGOMS edited to include a call to FONTCREATE. The form will be evaluated when the INIT file is loaded.

To quit, exit from DEdit in the usual way. When you run the function MAKEFILES (See Section 11.6, Page 11.7.), be sure that you are connected to the directory (see Section 8.7, Page 8.4) where the INIT file should appear. Now when GREET is run, your init file will be loaded.

[This page intentionally left blank]

# 13. FLEXIBILITY AND FORGIVENESS: CLISP AND DWIM

---

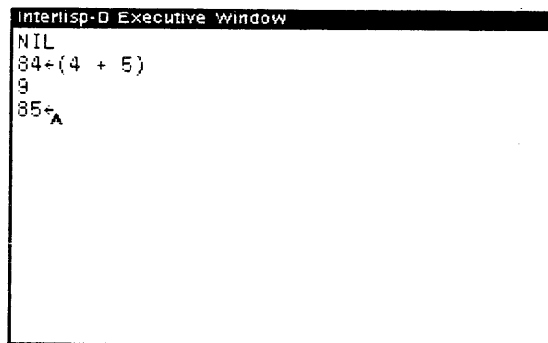
CLISP, (Conversational Lisp), and DWIM, (Do What I Mean), are two Interlisp utilities that make life easier.

---

## 13.1 CLISP

---

CLISP allows the machine to understand and execute commands given in a non-standard way. For example, Figure 13.1 contains an example expression (4 + 5).



```
Interlisp-D Executive window
NIL
84+(4 + 5)
9
85+A
```

Figure 13.1. CLISP allows the use of infix notation

Without CLISP, you would need to type this using the notation (PLUS 4 5). CLISP allows you to use expressions such as (4 + 5) for all arithmetic expressions.

CLISP also allows you to use more readable forms instead of standard Lisp control structures. Expressions like IF-THEN-ELSE statements can replace COND statements. For example, instead of:

```
(COND ((GREATERP A B) (PLUS A 10))
      (T (PLUS B 10)))
```

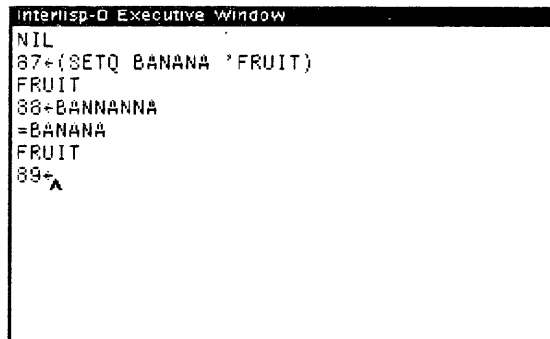
the following can be used:

```
(if (A > B) then (A + 10) else (B + 10))
```

The system translates this CLISP code into Interlisp-D code. Setting flags will allow you to either save the CLISP code, or save the translation. One such flag is CLISPIFTRANFLG; if it is set to NIL, all the IF statements will be replaced with the equivalent COND statements. This means that when you DEdit the function, the IF will be removed and replaced with the COND. Typically, flags such as this one are set in your INIT file. These flags are discussed in the *Interlisp-D Reference Manual* in Volume 2, Section 21.

## 13.2 DWIM

DWIM tries to match unrecognized variable and function names to known ones. This allows Lisp to interpret minor typing errors or misspellings in a function, without causing a break. Line 87 of Figure 13.2 illustrates how the misspelled BANNANNA was replaced by BANANA before the expression was evaluated.



```
Interlisp-D Executive Window
NIL
87=(SETQ BANANA 'FRUIT)
FRUIT
88=BANNANNA
=BANANA
FRUIT
89=
A
```

Figure 13.2. Examples of CLISP and DWIM features

Sometimes DWIM may alter an expression you didn't want it to. This may occur if, for example, a hyphenated function name (eg. **(MY-FUNCTION)**) is misused. If the system doesn't recognize it, it may think you are trying to subtract "FUNCTION" from "MY". DWIM also takes the liberty of updating the function, so it will have to be fixed. However, this is as much a blessing as a curse, since it points out the misused expression!

The Break Package is a part of Interlisp that makes debugging your programs much easier.

---

## 14.1 Break Windows

---

A break is a function either called by the programmer or by the system when an error has occurred. A separate window opens for each break. This window works much like the Interlisp-D Executive Window, except for extra menus unique to a break window. Inside a break window, you can examine variables, look at the call stack at the time of the break, or call the editor. Each successive break opens a new window, where you can execute functions without disturbing the original system stack. These windows disappear when you resolve the break and return to a higher level.

---

## 14.2 Break Package Example

---

This example illustrates the basic break package functions. A more complete explanation of the breaking functions, and the break package will follow.

The correct definition of **FACTORIAL** is:

```
(DEFINEQ (FACTORIAL (X)
  (if (ZEROP X) then 1
      else (ITIMES X (FACTORIAL (SUB1 X))
```

To demonstrate the break package, we have edited in an error: **DUMMY** in the IF statement is an unbound atom, it lacks a value.

```
(DEFINEQ (FACTORIAL (X)
  (if (ZEROP X) then DUMMY
      else (ITIMES X (FACTORIAL (SUB1 X))
```

The evaluated function

```
(FACTORIAL 4)
```

should return 24, but the above function has an error. **DUMMY** is an unbound atom, an atom without an assigned value, so Lisp will "break". A break window appears (Figure 14.1), that has all the functionality of the typing Interlisp-D expressions into the Interlisp-D executive window (The top level), in addition to the break menu functions. Each consecutive break will move to another level "down".

```

51+(PP FACTORIAL)
(FACTORIAL
 [LAMBDA (X) **COMMENT**
  (if (ZEROP X)
    then DUMMY
    else (ITIMES X (FACTORIAL (SUB1 X)))
  )
(FACTORIAL)
52+(FACTORIAL 4)
DUMMY - UNBOUND ATOM break: 1
UNBOUND ATOM
DUMMY {in FACTORIAL} in ((ZEROP X) DUMMY)
(DUMMY broken)
53:

```

Figure 14.1. Break Window

Move the mouse cursor into the break window and hold down the middle mouse button. The Break Menu will appear. Choose BT. Another menu, called the stack menu, will appear beside the break window. Choosing stack items from this menu will display another window. This window displays the function's local variable bindings, or values. (See Figure 14.2) This new window, titled FACTORIAL Frame, is an inspector window. (See Inspector Chapter 32).

```

56 FACTORIAL Frame
   FACTORIAL
57 X      0
(F
(F
58
DUMMY - UNBOUND ATOM break: 1
UNBOUND ATOM
DUMMY {in FACTORIAL} in ((ZEROP X) DUMMY)
(DUMMY broken)
59:
ERRORSET
BREAK1
COND
FACTORIAL
COND
FACTORIAL
COND
FACTORIAL
COND
FACTORIAL
COND

```

Figure 14.2. Back Trace of the System Stack

From the break window, you can call the editor for the function **FACTORIAL** by typing

**(DF FACTORIAL)**

Underline **X**. Choose **EVAL** from the editor menu. The value of **X** at the time of the break will appear in the edit buffer below the editor window. Any list or atom can be evaluated in this way (See Figure 14.3.)

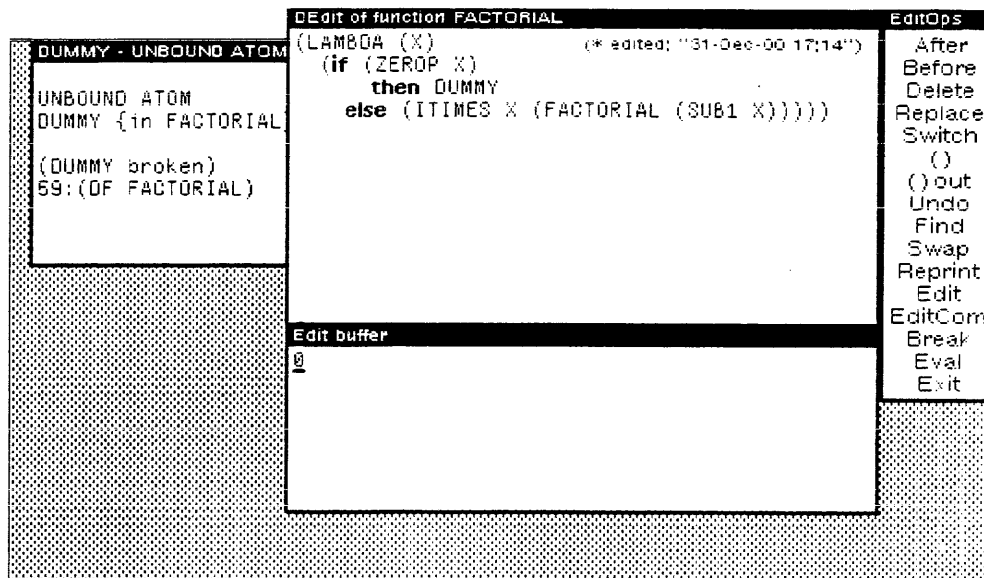


Figure 14.3. Editing from the Break Window

Replace the unbound atom **DUMMY** with 1. Exit the editor with the **EXIT** command on the editor menu.

The function is fixed, and you can restart it from the last call on the stack (It does not have to be started again from the Top Level) To begin again from the last call on the stack, choose the last (top) **FACTORIAL** call in the **BT** menu. Select **REVERT** from the middle button break window, or type it into the window. The break window will close, and a new one will appear with the message: **FACTORIAL** broken.

To start execution with this last call to **FACTORIAL**, choose **OK** from the middle button break menu. The break window will disappear, and the correct answer, 24, will be returned to the top level.

## 14.3 Ways to Stop Execution from the Keyboard, called "Breaking Lisp"

There are ways you can stop execution from the keyboard. They differ in terms of how much of the current operating state is saved:

- Control-G provides you with a menu of processes to interrupt. Your process will usually be "EXEC". Choose it to break your process. A break window will then appear.
- Control-B causes your function to break, saves the stack, then displays a break window with all the usual break functions.

For information on other interrupt characters, see the *Interlisp Reference Manual*, volume III, page 30.1.



## 14.4 Programming Breaks and Debugging Code

Programming breaks are put into code to cause a break when that section of code is executed. This is very useful for debugging code. There are 2 basic ways to set programming breaks:

**(BREAK functionname)**

This function call made at the top level will cause a break at the start of the execution of "functionname". This is helpful in checking the values of parameters given to the function.

Setting a break in the editor

Take the function that you want to break into the editor. Underline the expression that should break before it is evaluated. Choose **BREAK** on the editor command menu. Exit the editor. The function will break at this spot when it is executed.

Once the function is broken, an effective way to use the break window for debugging is to put it into the editor window. (See Section 14.2, Page 14.2.) All the local bindings still exist, so you can use the editor's **EVAL** command to evaluate lists, variables, and expressions individually. Just underline the item in the usual way (move the mouse to the word or parenthesis and press the left mouse button), then choose **EVAL** from the command menu. (See Section 14.2 for more detail.)

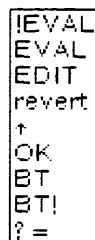
Both kinds of programmed breaks can be undone using the **(UNBREAK)** function. Type

**(UNBREAK functionname)**

Calling **(UNBREAK)** without specifying a function name will unbreak all broken functions.

## 14.5 Break Menu

Move the mouse cursor into the break window. Hold the middle button down, and a new menu will pop up, like the one in Figure 14.4.



```

EVAL
EVAL
EDIT
revert
↑
OK
BT
BT!
? =

```

**Figure 14.4.** The middle button menu in the Break Window

Five of the selections are particularly important when just starting to use Interlisp-D:

**BT** Back Trace displays the stack in a menu beside the break window. Back Trace is a very powerful debugging tool. Each function call is placed on the stack and removed when the execution of that function is complete. Choosing an item on the stack will open another window displaying that item's local

variables and their bindings. This is an inspector window that offers all the power of the inspector. (For details, see the section on the Inspector, Chapter 32).

- ?= Before you use this menu option, display the stack by choosing **BT** from this menu, and choose a function from it. Now, choose **?=**. It will display the current values of the arguments to the function that has been chosen from the stack.
- ↑ Move back to the previous break window, or if there is no other break window, back to the top level, the Interlisp-D Executive Window.
- REVERT Move the point of execution back to a specified function call before the error. The function to revert back to is, by default, the last function call before the break. If, however, a different function call is chosen on the **BT** menu, revert will go back to the start of this function and open a new break window. The items on the stack above the new starting place will no longer exist. This is used in the tutorial example. (See Section 14.2, Page 14.1.)
- OK Continue execution from the point of the break. This is useful if you have a simple error, i.e. an unbound variable or a nonnumeric argument to an arithmetic function. Reset the variable in the break window, then select **OK**. (See Section 14.2.)

(Note: In addition to being available on the middle button menu of the break window, all of these functions can be typed directly into the window. Only **BT** behaves differently when typed. It types the stack into the trace window instead of opening a new window.)

---

## 14.6 Returning to Top Level

---

Typing Control-D will immediately take you to the top level from any break window. The functions called before the break will stop, but any side effects of the function that occurred before the break remain. For example, if a function set a global variable before it broke, the variable will still be set after typing Control-D.

[This page intentionally left blank]

## 15. ON-LINE HELP WITH INTERLISP-D: HELPSYS AND DINFO

---

HELPSYS and DINFO access the on-line *Interlisp-D Reference Manual* for answers to your questions. The *Interlisp-D Reference Manual* must be on the hard disk ({DSK}) or on a file server. The manual is contained in the files Chap\*.IRM. In addition, the file IRM.HASHFILE is required. They can all be found on the Library floppies, and should be stored together in a single directory.

Set the value of the variable IRM.HOST&DIR to this directory. Load the file HELPSYS.DCOM (type (FILESLOAD HELPSYS.DCOM)) to run Helpsys, and DINFO.DCOM (type (FILESLOAD DINFO.DCOM)) to run DInfo.

---

### 15.1 HelpSys

---

Helpsys gives ?<sup>CT</sup> a meaning. When you type it before finishing the function call (i.e., before you type the arguments to the function, or before typing the closing right parenthesis), the manual entry for that function will be displayed.

Another way to see the manual entry for a function is to type **HELP <keyword>**.

If you do not know the name of a function, you can use the function **IRM.SMART.LOOKUP** to see manual entries. Type

(IRM.SMART.LOOKUP keyword)

The character \* can be used as a wildcard. For example, type:

(IRM.SMART.LOOKUP 'PRIN\*)

to see the manual entries for the functions that begin with the letters "PRIN".

---

### 15.2 DInfo

---

DInfo supplies the same information as HELPSYS, but in a different form. It represents the *Interlisp-D Reference Manual* as a tree structure. DInfo will appear on the background menu. Choose it to use the package.

A menu will appear (see Figure 15.1) that has the items Graph!, Node:, Top!, Parent!, Previous!, Next!, display:, Lookup!, and

Find!. The selections on this menu allow you to traverse the Dinfo graph.

```

Graph!   IRM
Node: Saving Virtual Memory State
Top!   IRM Top
Parent! Miscellaneous
Previous! Idle Mode
Next!   Date and Time Functions
Display: Graph Menu Text History
Lookup! *SYS*
Find!  FILE
    
```

Figure 15.1. A Segment of the DINFO Menu

The help text appears in a window below the menu, as in Figure 15.2.

```

Dinfo
17. FILE PACKAGE

Warning: The subsystem within the Interlisp-D environment
used for managing collections of definitions (of functions,
variables, etc.) is known as the "File Package." This
terminology is confusing, because the word "file" is also used
in the more conventional sense as meaning a collection of data
stored some physical media. Unfortunately, it is not possible
to change this terminology at this time, because many
functions and variables (MAKEFILE, FILEPKGTYPES, etc.)
incorporate the word "file" in their names. Eventually, the
system and the documentation will be revamped to
consistently use the term "module" or "definition group" or
"defgroup."

Most implementations of Lisp treat symbolic files as
    
```

Figure 15.2. Part of the text associated with the File Package Dinfo node.

The graph itself also appears in a separate window, as shown in Figure 15.3.

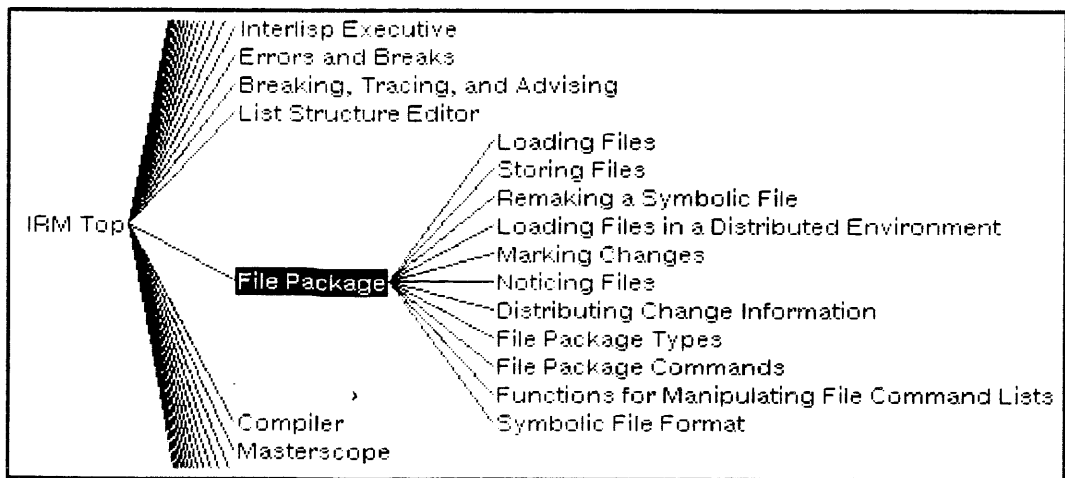


Figure 15.3. A Portion of the Dinfo Graph.

You can select what part of the manual to display by buttoning a node in the graph and scrolling through the text that is displayed

- 
- for the topic that was just buttoned, or by using the menu. The meaning of the commands in the menu is as follows:
- Graph! the only Graph! available is IRM (*Interlisp-D Reference Manual*;) ;
- Node: the node currently being visited;
- Top! The IRM Top;
- Parent! the current node's parent;
- Previous! is the node visited prior to the current node;
- Next! the node below this one in the graph
- Choosing either Parent!, Previous!, Next! or Top! will visit that node.
- Display! The display command determines how the information will be presented. The items to the right of display: are Graph, Menu, Text, and History.
- Graph will display a graph local to the current node, and if one of the nodes of the graph are chosen, that node will be visited.
  - Menu shows a menu of the subnodes of the current node. If one of these items are selected, that node will be visited.
  - Text displays the text of the current node. If you are searching for a particular node, do not turn this feature on during the search. It will slow down your progress through the tree. Turn it on when you have found the node you are looking for.
  - History records and displays a history of the nodes visited. Revisiting a node is done by choosing one of the items in this menu
- Lookup! has DInfo look up a term in the index of the IRM, and display the node that contains it.
- Find! will try to find a term in the *Interlisp-D Reference Manual* entry for the current node

[This page intentionally left blank]

---

## 16.1 Buying Floppy Disks

---

The 1108 uses double density, double sided, unformatted, 8 inch floppy disks.

The 1186 uses double density, double sided, unformatted, 5 1/4 inch floppy disks.

---

## 16.2 Basic Floppy Disk Information

---

- The terms "floppy disk", "diskette", and "floppy" are synonymous.
- The black plastic square is called the jacket. It permanently protects the disk inside from oils, scratches and dust. (See Figure 16.1 and Figure 16.2)

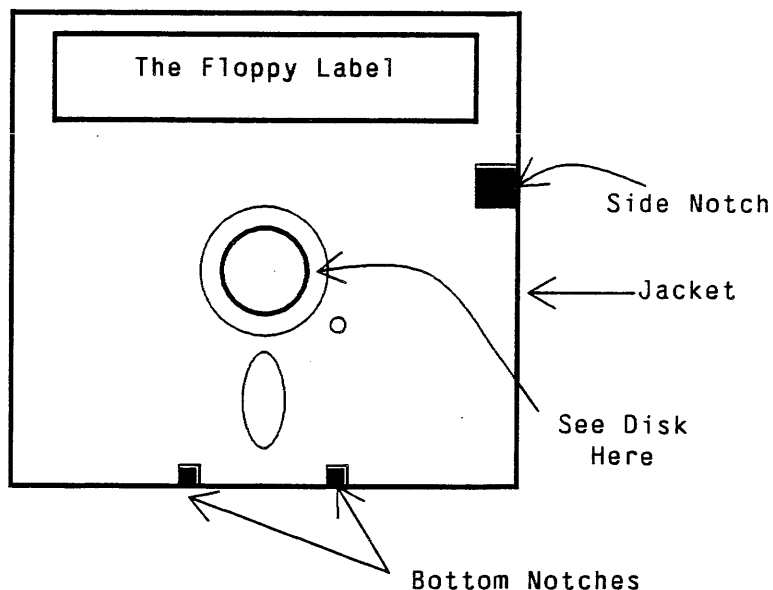


Figure 16.1. A 5 1/4 Inch Floppy



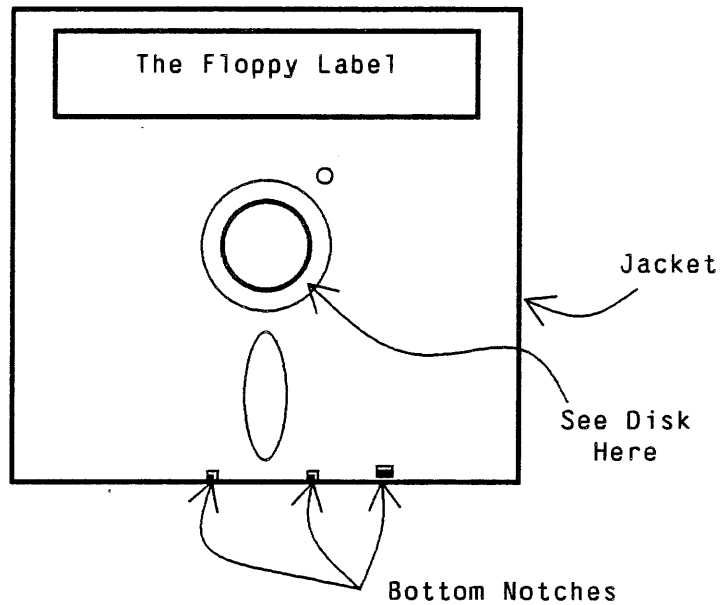


Figure 16.2. An 8 Inch Floppy

- Information is magnetically stored on the inner portion protected by the jacket. You can see a bit of the disk through the ring in the middle.
- The paper cover that comes with a floppy disk is called the sleeve.
- When you look at a floppy, the side with the label is the front, and the edge with the notches the bottom. (On a 5 1/4 inch floppy disk, the bottom is the side with 2 small notches; the right side has one large notch.)

## 16.3 Care of Floppies

Here are some suggestions for how to take care of floppy disks:

- Floppies are very delicate and will become useless if bent, folded, scratched, left sitting in the sun or on your floppy drive, or if things are stacked on them.
- Avoid touching anything but the jacket of the floppy, and be sensitive even at that.
- Keep the floppy in its sleeve when not in use.
- Store the floppy in an upright position, preferably in a rigid box with a lid.
- When labeling floppies, use a felt pen, and press VERY SOFTLY, or write on the floppy label first, then attach it to the floppy.
- Never take a floppy out of the floppy drive when the drive's red light is on.

---

## 16.4 Write Enabling and Write Protecting Floppies

---

### 16.4.1 Write Enabling an 1108's Floppy Disk

---

You can choose to allow the 1108 to write new information to, or alter stored information on, the floppy; or you can choose to protect the floppy's stored information.

The 1108 floppies have three notches on the bottom. When the right notch is UNcovered, the floppy is WRITE PROTECTED, and the 1108 cannot write to the floppy. If this notch is covered, the floppy is WRITE ENABLED, and the 1108 can write to the floppy. (Note that the 1186 and the 1108 are opposite!)

To Write Enable a Floppy

Cover the rightmost notch on the bottom of the disk with half of a write-enable adhesive tab. Fold the tab around to cover the back of the notch with the other half of the tab.

To Write Protect a Floppy Disk

Remove the write enable adhesive tab carefully from the disk. If there is not a tab over the rightmost notch of the floppy, the floppy is already write-protected.

### 16.4.2 Write Protecting an 1186's Floppy Disk

---

You can choose to allow the 1186 to write new information to, or alter stored information on, the floppy; or you can choose to protect the floppy's stored information.

The 1186 floppies have a notch on the right. When the right notch is UNcovered, the floppy is WRITE ENABLED, and the 1186 can write to the floppy. If this notch is covered, the floppy is WRITE PROTECTED, and the 1186 cannot write to the floppy. (Notice that the 1186 and the 1108 are opposite!)

To Write Enable a Floppy Disk

Carefully remove the write protect adhesive tab from the disk. If there is not a tab over the notch on the right side of the floppy, it is already write enabled.

To Write Protect a Floppy

Cover the notch on the right of the disk with half of a write-enable adhesive tab. Fold the tab around to cover the back of the notch with the other half of the tab.

---

## 16.5 Inserting Floppies into the Floppy Drive

---

Open the floppy door and slide the floppy disk in - label side up, the bottom in first. If it is a new floppy, you cannot read or write to it until you format it. (Read the next section for formatting instructions.)

## 16.6 Functions for Floppy Disks

### 16.6.1 Formatting Floppies

Before use, a new floppy must be formatted. Formatting a floppy is also a quick way to erase all of its files. To format a floppy, insert the floppy into the floppy drive and type:

```
(FLOPPY.FORMAT)
```

For a new floppy, this will take 2 to 5 minutes.

**WARNING!** When you format a floppy, you lose anything that was on it previously. Reformat used floppies only if you want to completely change everything on the floppy. (Reformatting is a quick way to recycle floppies.)

### 16.6.2 Available Space on a Floppy Disk

You can check to see how much space is left of your floppy disk with the function `FLOPPY.FREE.PAGES`.

Just type:

```
(FLOPPY.FREE.PAGES)
```

By checking to see how much space is left on your floppy, you will know when it is filling up, and it is time to format another one for the rest of your files.

### 16.6.3 The Name of a Floppy Disk

Another way to keep track of your floppies is to give them names. This can be done when you format the floppy, by giving the name of the floppy as an argument to `FLOPPY.FORMAT`, or by using the function `FLOPPY.NAME`. The syntax of `FLOPPY.NAME` is

```
(FLOPPY.NAME name)
```

If `FLOPPY.NAME` is not given an argument, the current name of the floppy is returned, as in the figure below:

```
Top level -- Connected to {DSK}<LISPFILE
NIL
40←(FLOPPY.NAME)
NEWPRIMER1
41←
```

**Figure 16.3.** If `FLOPPY.NAME` is not given an argument, the current name of the floppy is returned.

## 16.6.4 FLOPPY.MODE

---

The function **FLOPPY.MODE** sets the way the system reads and writes on a floppy. A floppy has one of four modes, either **PILOT**, **HUGEPILOT**, **SYSOUT**, or **CPM**. This primer will cover **PILOT**, **HUGEPILOT**, and **SYSOUT**. For more information, see the *Interisp-D Reference Manual*, Volume III, pages 24.24 to 24.26.

The usual mode of a floppy is **PILOT**. You do not need to run the function **FLOPPY.MODE** before the function **FLOPPY.FORMAT** if you want to format a floppy that should have the **PILOT** mode.

**HUGEPILOT** floppies hold a file (other than a **SYSOUT** file) that is too large to fit on one floppy. If you have a file that is this large, set the mode of the floppies to **HUGEPILOT** by typing:

```
(FLOPPY.MODE 'HUGEPILOT)
```

When an output file is created, you will be prompted to insert a new floppy as needed. Each time you will be asked whether the system can erase and format the new floppy. **REMEMBER** to change the **FLOPPY.MODE** back to **PILOT** when you are done!

**SYSOUT** mode is used for storing **SYSOUT** files on multiple floppy disks. It is set automatically when the function **SYSOUT** is called. As with **HUGEPILOT** floppies, you will be asked to insert new floppies as needed.

[This page intentionally left blank]

---

## 17.1 Supplies

---

Have on hand:

- Source Floppy Disk (original)
- Destination Floppy Disks (the ones you want to copy onto.)
- Labels (Copy the information from the source floppy label to new labels. Don't attach them to the floppy disk yet.)
- Small blank white or metallic colored adhesive tabs to write enable the 1108 floppy or write protect the 1186 floppy.)

---

## 17.2 Preparation

---

### 17.2.1 Handling Floppy Disks

---

If you have never used a floppy disk before, please read Section 16.3, Page 16.2 "Care of Floppy Disks". When speaking of a floppy disk, we call the edge with the notches the bottom, and the side with the label the front.

### 17.2.2 Setup

---

**Source** For the 1108: remove the small tab on the bottom of the source disk.

For the 1186, attach a small tab to the right side notch of the source disk.

Now your original diskette cannot be erased or inadvertently overwritten.

**Destination** For the 1108: attach a write enable tab over the notch on the bottom (See Section 16.4.1, Page 16.3, or Section 16.4.2, Page 16.3.) Be certain that the entire notch, both front and back, is completely covered by the tab.

For the 1186: There must not be a tab covering the notch. Remove the tab if it is there.

Now the destination diskettes can take information.

**Labels** Duplicate the source floppy label for each destination copy you intend to make. Include the date of the copy, and the word "COPY", to indicate that it is a copy and not the original. Don't attach them to the floppies until the copy is finished. This will help to keep track of the ones that are finished and the ones still to be copied.

---

### 17.3 Copying Floppy Disks

---

- (1) Follow the supplies and preparation information from Section 17.1 and Section 17.2, on Page 17.1.
- (2) Insert the source floppy into the floppy disk drive. Close the disk drive door.
- (3) Type the function (**FLOPPY.TO.FILE 'filename**) to read all the information from the floppy found in the floppy drive to the hard disk.
- (4) Once this function has completed, insert the destination floppy disk into the floppy drive, then close the door.
- (5) Type the function (**FLOPPY.FROM.FILE 'filename**) using the same filename as before to write the information onto the destination floppy found in the floppy disk drive.
- (6) The function **FLOPPY.FROM.FILE** formats the floppy, then copies the information.
- (7) To make more than one copy of your original diskette, insert another destination floppy into the floppy drive, close the door, and call the function (**FLOPPY.FROM.FILE 'filename**). (The source floppy does not need to be read onto the hard disk for additional copies.)

A SYSOUT is a file of the whole Interlisp-D environment and everything that you have defined or loaded into it. The file is very large, and takes many floppies to store. When you load the file, the exact environment at the point of the sysout is restored. To make SYSOUT's of your own environment, see Section 18.2.

---

## 18.1 Loading SYSOUT Files

---

### 18.1.1 Loading a SYSOUT file on the 1108

---

Sysouts must be loaded from the Installation Menu. This is the menu that appears when you do a 2-BOOT with the Installation Utility floppy disk in the drive. (refer to Section 3.1, Page 3.2 of this primer.) Any sysout from floppy, new software releases, or sysouts made by you must be loaded with the following instructions.

- (1) Do a 2-BOOT with the floppy labeled INSTALLATION UTILITY in the floppy drive. (A 2-BOOT is done by pressing both the B RESET and the ALT B buttons on the front panel and immediately releasing the B RESET button. Release the ALT B button when the panel reads 0002. Then wait about two minutes.)
- (2) When the question **Time offset from Greenwich?** appears, type **-5** for Eastern Standard Time (subtract one for each time zone westward), and **CR**.
- (3) For the next three more questions, simply type **CR**.
- (4) At this point, the machine will do one of two things. It will either prompt you to enter the date and time, (See Figure 3.1 in Section 3.1.), or it will ask you if you want to change the time (to which you can just respond **NO CR**).
- (5) When the Installation Menu appears, choose **Install Interlisp-D on LISP Volume**.
- (6) Insert the floppy "SYSOUT #1" into the drive and answer **YES** to the ready question. (If you have the wrong floppy in the drive, the machine will tell you - simply put in the correct one.)
- (7) The processor will take a few minutes to read the floppy. Just wait. When it is done, a high pitched tone will signal you to insert "SYSOUT #2" into the floppy drive. When the red light is out, insert "SYSOUT #2".



- (8) Do the same for "SYSOUT #3".
- (9) After "SYSOUT #3" is finished loading, you will be asked if there are any more floppies to load. If you have another, insert **SYSOUT #4** into the drive and answer **Y** <sup>CR</sup>. When the last sysout is loaded, answer **N** <sup>CR</sup>.
- (10) At this point you should be back at the Installation Menu. Do a 1-BOOT. This gives you the screen with the bouncing white recotangle.
- (11) Press the left button to start the **InstallLispTool**.
- (12) Click the left mouse button over Volume Size. You will get a blinking black caret. Move the caret (by pressing the left mouse button over it and holding it down) to the end of the number. Press backspace to erase the number, then type 15100 in its place.
- (13) Now choose **SetVMem** with the left mouse button. Confirm with the left button when you get the confirm mouse cursor. This takes a minute or two. As long as the words **SetVMem** are black, it is not finished.
- (14) When DONE is printed on the screen, choose QUIT with the left mouse button. Now go to Chapter 3 and choose one of the methods for "Getting into LISP."

### 18.1.2 Loading a SYSOUT file on the 1186

---

Any sysout from floppy, new software releases, or sysouts made by you must be loaded with the following instructions.

- (1) Insert the Installation Utility floppy disk in the drive.
- (2) Press the B Reset button on the front of the processor.
- (3) When the Boot Icons appear at the bottom of the screen, press the F2 key, to choose the icon with a picture of the floppy on it.
- (4) When the question **Time offset from Greenwich?** appears, type -5 for Eastern Standard Time (subtract one for each time zone westward), and <sup>CR</sup>.
- (5) For the next three more questions, simply press <sup>CR</sup> after each prompt.
- (6) At this point, the machine will do one of two things. It will either prompt you to enter the date and time, (See Figure 3.3 in Section 3.2, Page 3.3.), or it will ask you if you want to change the time (to which you can just respond **NO** <sup>CR</sup>).
- (7) When the next menu appears, choose **1 InterLisp-D**.
- (8) When the next menu appears, type **3 "System Utilities (Installation etc.)"**
- (9) When the system prompts you to insert the "Installation Files" floppy, do so and press <sup>CR</sup>.
- (10) At this next menu, choose item **1 "Lisp Installation"**.

- (11) Next, type 12 "Install Interlisp-D on Lisp Volume"
- (12) Insert the floppy SYSOUT #1 into the drive and answer YES to the ready question. (If you have the wrong floppy in the drive, the machine will tell you - simply put in the correct one.)
- (13) The processor will take a few minutes to read the floppy. Just wait. When it is done, you will be prompted to insert SYSOUT #2 into the floppy drive. Before opening the floppy drive door, wait until the red light on the floppy door goes out. When the red light is out, insert SYSOUT #2.
- (14) Do the same for SYSOUT #3.
- (15) After SYSOUT #3 is finished loading, you will be asked if there are any more floppies to load. If you have another, insert SYSOUT #4 into the drive and answer Y<sup>CR</sup>. When the last sysout is loaded, answer N<sup>CR</sup>.
- (16) This process will take approximately 30 minutes
- (17) After the floppies have been loaded, type 16 "Copy from Lisp Volume to Lisp2 Volume" to save a back up copy of the sysout.
- (18) Type 13 "Expand Lisp Vmem" before you boot the sysout.
- (19) Type 17 "Boot from Lisp volume" to boot the volume
- (20) The icons should reappear at the bottom of the screen. Press the F1 key to choose the icon with the picture of the computer on it. This will boot Interlisp-D, and the Interlisp-D windows will appear on your screen.

---

## 18.2 Making Your Own SYSOUT File

---

- (1) For the 1108: have about 5 floppy disks available to store your SYSOUT file.  
For the 1186: have about 10 floppy disks available to to store your SYSOUT file.  
The exact number you will need depends on how much you have loaded or defined in the Lisp environment.
- (2) Make sure the floppy disks are write enabled (See Section 16.4.1, Page 16.3, or Section 16.4.2, Page 16.3).
- (3) The floppy disks do not need to be formatted.
- (4) Type (SYSOUT ' {FLOPPY} )
- (5) The machine will prompt you by ringing bells and typing the message **Insert the next floppy disk** in the Interlisp-D Executive Window at the appropriate times.
- (6) The machine will type **MIL** then the number prompt followed by a left pointing arrow when the entire SYSOUT has been written onto your floppies.

- (7) Storing your SYSOUT file was only an interruption, like going to answer the phone, and you may continue to work where you left off.

# 19. USING THE EPSON FX80 PRINTER

---

The FX80 printer is only connected to the machine you are using. You cannot access this printer from any other machine that may be in the room. The software to use with the printer comes from Xerox, and can be found on the file named FXPRINTER.DCOM. This file should be loaded into Lisp. To do this, type (FILESLOAD FXPRINTER).

---

## 19.1 Initializing the RS232 Port

---

Initialize the RS232port by typing into the Interlisp-D Executive Window:

```
(RS232C.INIT 9600 8 NIL 1)
```

The printer is now ready to receive information from your lisp machine.

---

## 19.2 Power up the Printer

---

- (1) Insert the paper so that you just see the perforated edge by the metal bar with numbers. This sets the top of the page in the correct position.
- (2) Turn the printer on. (There is probably a rocker switch on the left side of the machine)
- (3) A light labeled "ON LINE" should be lit on your printer panel. If this light is not on, push the "ON LINE" switch, and it should turn on.
- (4) To use the Form Feed, push "ON LINE" so that the "ON LINE" light will go out, then push form feed. Remember to push "ON LINE" again (to turn the "ON LINE" light back on) after the form feed.

---

## 19.3 To Align Top of Page

---

- (1) Push the "ON LINE" button, to turn the "ON LINE" light out.

- (2) Turn the knob on the right side of the printer so that the dividing line between the sheets of paper is just visible at the silver bar on the paper feed.
- (3) Push the "ON LINE" button again to turn the green lights back on.

---

## 19.4 Functions To Print Files and Bitmaps

---

### 19.4.1 RS232.Print

---

One function to print files, bitmaps, and windows is `RS232.PRINT`. Here are some examples of its use:

- `(RS232.PRINT filename)` to print an entire file. The file does not have to be loaded; printing directly from a floppy works well.
- `(RS232.PRINT bitmapname)` to print an existing bitmap.
- `(RS232.PRINT windowname)` to print what is displayed on a window.

### 19.4.2 FX80STREAM

---

Another set of functions that can be used to print files, bitmaps, and windows are found in the library package `FX80STREAM`. Here are some examples of using this package to print a TEdit file, and do the other operations shown above for `RS232.PRINT`:

In TEdit, select (blacken the item, but do not release the mouse button) **Hardcopy** from the default right button menu. Move the mouse cursor to the point of the arrow at the side of that selection. Another menu will appear. Choose **To a file**, and release the button. At the prompt, type `{RS232}.FX80` followed by a `CR`.

To print what is displayed on a window, or print a bitmap, first type

```
(SETQ FX80 (OPENIMAGESTREAM '{RS232}FOO.FX80))
```

Then:

- `(BITBLT (WHICHW) NIL NIL FX80 0 0)` to print the contents of the window that the mouse is in, or
- `(BITBLT bitmapname NIL NIL FX80 0 0)` to print an existing bitmap, bound to `bitmapname`.

### 19.4.3 Printing a Portion of the Screen

To send a portion of what is on the screen to the printer, you will need a function that copies that part of the screen bitmap onto a smaller bitmap that you can send to the printer. The function is **FX.SNAP**. Here are instructions for how to define and use this function:

- (1) To define it, type exactly what is printed below.

```
(DEFINEQ (FX.SNAP
  [LAMBDA NIL
    (PROG (REG BTEMP)
      (SETQ REG (GETREGION))
      (SETQ BTEMP
        (BITMAPCREATE
          (fetch (REGION WIDTH) of REG)
          (fetch (REGION HEIGHT) of REG)))
      (BITBLT (SCREENBITMAP)
        (fetch (REGION LEFT) of REG)
        (fetch (REGION BOTTOM) of REG)
        BTEMP 0 0)
      (RS232.PRINT BTEMP NIL T)
    ])
```

- (2) Once you have typed this, the function is in the environment, and can be used as often as you like. If you save this function on a file, you can load it each time you need it, and will not have to retype it.
- (3) Type **(FX.SNAP)** to send a picture of the screen to the printer. You will be prompted to sweep out the section of the screen with the shape window prompt. Once you have used the mouse to sweep out the section of the screen to be printed, the printer will automatically start printing.

[This page intentionally left blank]

## 20. RS232 FILE TRANSFER WITH A VAX

---

---

### 20.1 Prerequisites

---

This file transfer chapter is for VAXes not connected to the 1108's and 1186's with an Ethernet. To do file transfers with a VAX, your 1108 or 1186 must have a connection from the RS232 Port to the VAX, and the VAX must have the MODEM or KERMIT transfer protocol available (installed as a system utility). The following files also need to be loaded: RS232CHAT, KERMIT.DCOM, and KERMITMENU.DCOM. The file KERMIT.DCOM contains both the MODEM and KERMIT protocols. Refer to Section 8.6, Page 8.4, for loading instructions.)

---

### 20.2 Using Chat to Transfer Files

---

- (1) Use a 1108 or a 1186 with a connection to the VAX
- (2) To begin, type:  
(RS232C.INIT 4800)  
(RS232CHAT)
- (3) You will be prompted to sweep out a window. This is the new "CHAT" window. Type `CF` after the blinking caret appears in it. The VAX will respond by printing a login prompt into the window.
- (4) Log onto the VAX in the usual way.
- (5) You are responsible for starting the MODEM or KERMIT program on the remote machine. If both MODEM and KERMIT are available, use KERMIT, since it is more flexible. Most KERMIT programs have a "server" mode so that you do not have to request the host to send or receive each file individually.
- (6) To transfer a file to or from the VAX, press the middle button of the mouse with the mouse in the CHAT window. A menu will appear; choose KERMIT. Another menu will pop up, as in Figure 20.1.



```

Send!  Receive!  Exit!
Transfer mode:  Kermit  Modem
Local file:  {DSK}<KLISPPFILES>FILE.TXT
Remote file:  file.txt
File type:  Text      End-of-line Convention:  CRLF

```

Figure 20.1. The Menu for File Transfer to the Vax

- (7) Before the file is transferred, you must give the variable items in the window the correct values. To set up this menu, follow the instructions below, given for each item:
- |                        |   |
|------------------------|---|
| Transfer Mode          | with the mouse, choose either KERMIT or MODEM, to match the file transfer package on your VAX.  |
| Localfile              | The name of the file that is being sent from or received by your 1108 or 1186. Use the mouse to position the caret, then type the filename into the window.   |
| Remotefile             | The name of the file that is being sent from or received by the VAX. Use the mouse to position the caret, then type the filename into the window.             |
| Filetype               | To set this parameter, point to <i>Filetype</i> and press a mouse button. A menu will appear. Choose <i>text</i> to transfer an ASCII file.                   |
| End-of-line Convention | To set this parameter, point to <i>Filetype</i> and press a mouse button. A menu will appear. Choose the item that is appropriate for your VAX, usually CRLF. |
- (8) The commands for using KERMIT or MODEM are those at the top of the menu. Choose the one that is appropriate for your job:
- |          |   |
|----------|---|
| Send!    | This function will move a file from the 1108, or 1186, to the VAX. The remote file transfer program must be prepared to receive the file. |
| Receive! | Moves a file from the VAX to the 1108 or 1186. The remote file transfer program must be prepared to send the file.                        |
- (9) As long as bells are not continuously ringing, the transfer is running normally.
- (10) Choose *Exit!* to close the file transfer menu.
- (11) Move the mouse cursor to the "CHAT" window and press the left button. Type `CR`. LOGOUT when you get the VAX system prompt.
- (12) Press the middle mouse button in the "CHAT" window and choose "BYE" from the menu.
- (13) SHRINK or CLOSE the "CHAT" window.

## 21.1 Prerequisites

---

Both the sending and receiving machines must be connected to an Ethernet.

If the communication is between two lisp machines,

- (1) They must both be running Interlisp-D.
- (2) The file FTPSERVER.DCOM must be loaded; type  
(FILESLOAD FTPSERVER)
- (3) The receiving lisp machine must be running an FTPSERVER process. This process allows the receiving machine to give the sending machine access to the files on the its disk. To make sure that the receiving machine is running this process, call the function

(ADD.PROCESS '(\FTPSERVER))

on that machine.

If you want to communicate with either a fileserver or a VAX, don't worry about the file FTPSERVER.DCOM or the FTPSERVER process. The functions will still work as described below.

## 21.2 File Transfer

---

The File Transfer process allows you to call the **DIR**, **LOAD**, and **COPYFILE** functions.

To address another machine (lisp machine, VAX, Fileserver) on the Ethernet, replace the device name {DSK} or {FLOPPY} with a number that uses the other machine's host number. For example, {1}filename. If you have a Xerox file server with a clearinghouse on it, which validates users and resolves names of servers to their host numbers, you can use the name of the machine instead of its host number.

Some examples of functions addressing other machines.

(DIR {3}DSK:<LISPPFILES>)

(COPYFILE '{3}oldfilename '{DSK}newfilename)

(LOAD '{2}filename)

**(DIR {RoseBowl}<Primers>Interlisp>\*.ip SIZE CREATIONDATE)**

You can copy files to or from other machines. If you do not call a specific directory of the other machine, you will access the directory that the other user is connected to.

The Executive window turns black

An example is shown in Figure 22.1.

Press any key to unfreeze the window and continue. This pause happens when the command you just typed causes enough information to be printed to fill the window — it gives you a chance to read that one window of text before moving on.

```

Interlisp-D Executive
{DSK}<LISPFILERS>PRIMER>
CH110BK.MSS;2
FIG18.BTM;1
FIG19.BTM;1
FIG19B.BTM;1
FIG19C.BTM;1
FIG2.BTM;2
FIG22.BTM;1
FIG22B.BTM;1
FIG6.BTM;1
FIGGLOSS.BTM;1
FIGNS.BTM;2
FIGNS.BTM;1
FRGLOSS.IM;2
FRGLOSS.MSS;3
JDOCTOC.TEDIT;2
NCH2.MSS;3
NCH3.MSS;3
NCH8.MSS;2
NCH9.MSS;8
QCH10.MSS;2

```

Figure 22.1. The Interlisp-D Executive Window, filled, and waiting for a character to be typed to continue

You closed the Executive Window

Just type any character on the keyboard, it will reopen automatically.

The mouse disappears

Type (CURSOR T) in the Interlisp-D Executive window. The cursor will reappear.

A second window appears

This probably happens because you made a typing mistake, as in Figure 22.2.

```

UNDEFINED-FUNCTION - UNDEFINED CAR OF FORM. Break; 1
37+(UNDEFINED-FUNCTION 'ARG1 'ARG2)
=UNDEFINED-(FUNCTION (QUOTE ARG1) (QUOTE ARG2))
? No
UNDEFINED-FUNCTION
(UNDEFINED-FUNCTION broken)
38:

```

Figure 22.2. The Break Window has appeared after a typing error

- Type a Control-D By simultaneously pressing the control key (see Section 2.2, Page 2.2 for the proper key on your machine) and the "D". This aborts the error condition, returning control to the executive window.
- Now retype the previous command.
- If the problem is not simply a typing error, please see Section 14.2.
- A break while writing to a floppy** If the break window has a title **NIL**, check to make sure that the floppy is not write protected. The "WRITEPROTECTED" message will be printed in the promptwindow.
- You keep getting beeped at** Usually the beeping means that Interlisp-D wants input from you. Look for the flashing caret. It will usually be preceded by some kind of prompt, indicating what you should type.
- You can't delete the first letter** of the filename you are typing to (**FILES?**). Type Control-E (error). You will get a linefeed and ←←← printed to the window. Now type the correct filename.
- Your function is just sitting there** not returning a value, and you think that your program may be in an infinite loop or is having some other major problem. You can see what process is currently running by typing Control-T, or you could interrupt the process by typing Control-E.
- The mouse cursor won't move** and there is no response from the keyboard. If you have an 1108 check the Maintenance Panel code. On the 1186, the mouse cursor itself should be a number. If the number is in the 9000s, Interlisp-D has crashed.
- For the 1108: Press the **UNDO** key on the top right of the right keypad. The mouse cursor should now say "Teleraid". If not, try typing Control-Shift-Delete (Yes, press all three keys at once!), and the mouse cursor should now say "Teleraid". Now type Control-D.
- For the 1186: Press the **UNDO** key on the leftmost keypad. The mouse cursor should now say "Teleraid". If not, try typing Control-Shift-Delete (Yes, press all three keys at once!), and the mouse cursor should now say "Teleraid". Now type Control-D.
- If this doesn't work, you must reload Interlisp-D. Remember, save early and save often. (See Section 3.3).
- A break window appears** If the break window looks something like this:

```
{DSK}<LISPPFILES>DAYBREAK>IM>RECORDS.IM;4 - FILE S*
FILE SYSTEM RESOURCES EXCEEDED
{DSK}<LISPPFILES>DAYBREAK>IM>RECORDS.IM;4
(DOPMAP broken)
37: A
```

**Figure 22.3.** The Break Window when there is not enough space to save your file. You are trying to save a file, but there is not enough space on the hard disk.

Exit from the break window by typing an "up arrow" (↑) followed by a **CR**. Delete old versions of files, and any other files you don't need, then try again to save the file

You have run out of space Generally, a **BREAK** window has appeared. The **GAINSPACE** function allows you to delete non-essential data structures. To use it, type:

**(GAINSPACE)**

into the Interlisp-D Executive Window. Answer "N" to all questions except the following.

- Delete edit history
- Delete history list.
- Delete values of old variables.
- Delete your **MASTERSCOPE** database
- Delete information for undoing your greeting.

Save your work and reload Lisp as soon as possible.

A redefined message appears

The message (*Some.Crucial.Function.Or.Variable redefined*), as in Figure 22.4. The function, variable, or property has been "smashed" (i.e. its original definition has been changed). If this is not what you wanted, type **UNDO** immediately!

```

Interlisp-D Executive
NIL
92+(DEFINEQ (CAR (A) (SomeOtherFn A)))
(CAR redefined)
(CAR)
93+UNDO
DEFINEQ undone.
94+A

```

Figure 22.4. CAR redefined! Type **UNDO** immediately

**UNBOUND ATOM**

If this occurs, you probably just typed something wrong, or you passed an argument that should have been quoted to a function.

**UNDEFINED CAR OF FORM**

First, look at what caused the error. If the **CAR** of the form is a list, then you typed something wrong. If it is an atom, then perhaps that atom does not have a function associated with it. If it is a CLISP word like **if**, **for**, or the like, then **DWIM** (see Section 13.2, Page 13.2) may have been turned off. Type **(DWIM 'C)** to reenable **DWIM**.

You have traced **APPLY**

and your screen is spewing out information about everything going on in the environment. Type Control-E, and type **(UNBREAK 'APPLY)** before returning to the Interlisp-D Executive.

[This page intentionally left blank]

TEdit is the Interlisp-D text editor. It is a "what you see is what you get" editor; what you see on the screen closely simulates what will be printed on paper. Besides normal text editing, using this editor allows you to move text, insert bitmaps, sketches, or snapshots of the screen into your text, format the document, and more. This chapter will only cover editing text, however. For more information, see the *Library Packages Manual*.

---

## 23.1 Using TEdit

---

The file, TEDIT.DCOM, must be loaded before you can use TEdit. (See Section 8.6, Page 8.4.) You can start the editor in one of three ways:

- (1) Choose TEdit from the Background menu. (If you are not familiar with the background menu, See Chapter 1 and Chapter 7.)
- (2) Type **(TEDIT)**
- (3) Type **(TEDIT 'filename)** to edit a specific file.

Open a TEdit window in one of the three ways. You will be prompted to sweep out an area of the screen for the text editor window. When the window appears, note the extra white area above the window's title bar. This area is used for prompts from the editor.

You can now type - what you type will appear in the window at the caret. (Shown in Figure 23.1.)

After you type, notice the "\*" that appears before the title in the title bar of the TEdit window. That means that the file contains information that has not been saved. Remember to save your files often!



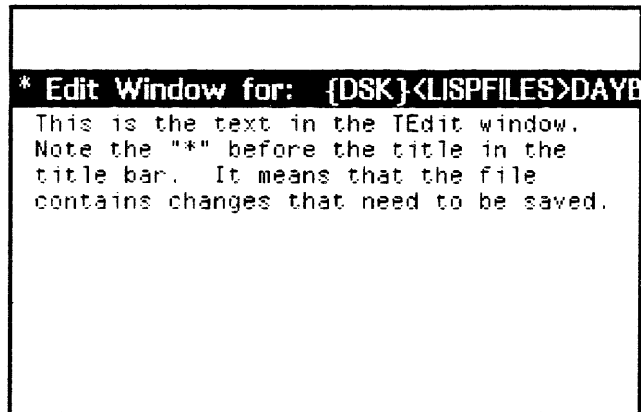


Figure 23.1. Text typed in the TEdit window

To position the caret, hold down the left mouse, and while holding the mouse button down move the mouse until the caret is flashing where you would like the caret to be. Then, release the mouse button.

Carriage returns are used in TEdit to delimit paragraphs. Within a paragraph, TEdit will automatically break the text into lines with a ragged right margin. (If you prefer justified margins, see Section 23.5.2.2, Page 23.12.) In order to insert a carriage return without starting a new paragraph (for the 1108) hold down the OPEN key while pressing `CR`, or (for the 1186) hold down the META key while pressing `CR`.

## 23.2 Managing the TEdit Window

While you are in the TEdit window, the mouse buttons have special meanings. However, you can still access the right button default window menu by pointing the mouse cursor at the TEdit window's title bar and pressing the right mouse button. Be aware that even some of the right mouse button default menu choices have special meaning in TEdit. For instance:

- |           |   |
|-----------|---|
| Close     | Stops the editing session. If there are changes that have not been saved, you will be asked to confirm this choice by pressing the left mouse.  |
| Clear     | Clears the window, but does <i>not</i> change the contents of the file. You will not delete any text by choosing this item from the menu. To have the contents reappear, choose <code>Redisplay</code> .  |
| Redisplay | Will redraw the contents of the window. Remember this if you choose <code>Clear</code> by mistake.  |
| Hardcopy  | Will send a copy of the TEdit file to the printer. Before printing the file, however, it will be correctly formatted for the printer (i.e. length of lines within paragraphs will be made the correct length for the paper, etc.). This process can take some time. |

**Shrink** Shrinks the window into the TEdit icon without stopping the editing session. This icon is a closed book with TEdit on the spine, and the name of the file on the front. (See Figure 23.2.)

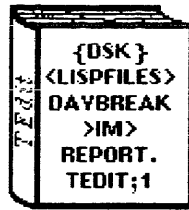


Figure 23.2. The TEdit Icon

The other choices of the default right button menu, snap, bury, move, and shape, work as described in Section 10.4, Page 10.3.

## 23.3 Selecting Text

"Selecting" a block of text for some modification is a common TEdit task. You will need to select text for a variety of commands, including copying and moving text. To select:

- A character** Point to the character and press the left mouse button. The character selected will be underlined.
- A word**
- (1) Point to the word with the mouse cursor.
  - (2) Press the middle mouse button and the word is selected (underlined).
- A line**
- (1) Point to the far left of the line with the mouse cursor. The mouse cursor will change from pointing to the northwest to pointing to the northeast.
  - (2) Press the left mouse button and the line is selected (underlined).
- A paragraph**
- (1) Point to the far left of a line in the paragraph with the mouse cursor. The mouse cursor will change from pointing to the northwest to pointing to the northeast.
  - (2) Press the middle mouse button and the paragraph is selected (underlined).
- Any block of text**
- (1) Move the caret to the beginning of the block of text to be chosen.
  - (2) Select the starting place using one of the methods from above.
  - (3) Press and hold the right mouse button.
  - (4) Move the mouse until the entire block of text is selected. The text will be marked by inverse video (see Figure 23.3.)
  - (5) release the mouse button and the indicated text is selected.

---

## 23.4 Deleting, Copying, and Moving Text with TEdit

---

### 23.4.1 Deleting Text From a File

---

Text can be deleted in a number of ways. The one you choose may depend on the amount to be deleted. If it is a very small amount, backspace to delete the character just behind the caret, or Control-W to delete the word just behind the caret.

For deleting a larger block of text,

- (1) Choose the section to be deleted. Select the last two sentences of the example file, as shown in Figure 23.3.
- (2) Press the **De l e t e** key to remove the highlighted area.

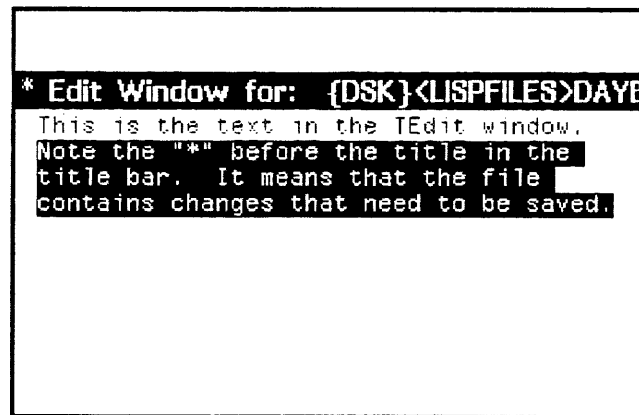


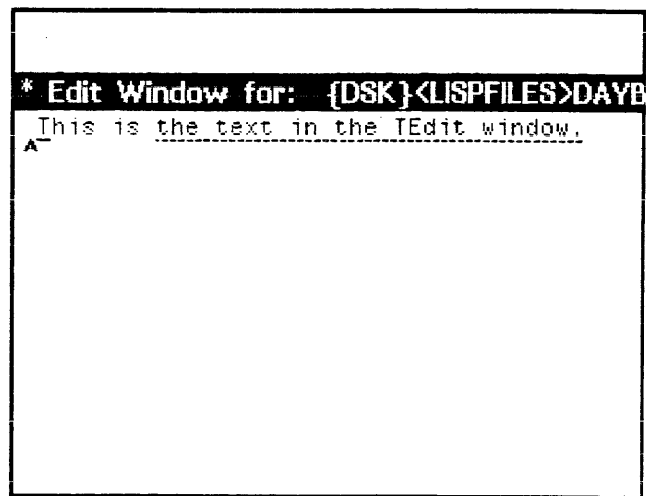
Figure 23.3. The text in reverse video is chosen, and will be deleted.

### 23.4.2 Copying Text

---

To copy a block of text from one place in the file to another,

- (1) Position the caret where you want the copied block of text to appear. For the example file, move it to the beginning of the file.
- (2) press and hold the **COPY** key, or the shift key;
- (3) choose the text to be copied. For the example file, choose the words "the text in the TEdit window". (See Figure 23.4.)



**Figure 23.4.** The underlined text has been chosen. It will be copied to appear before the cursor.

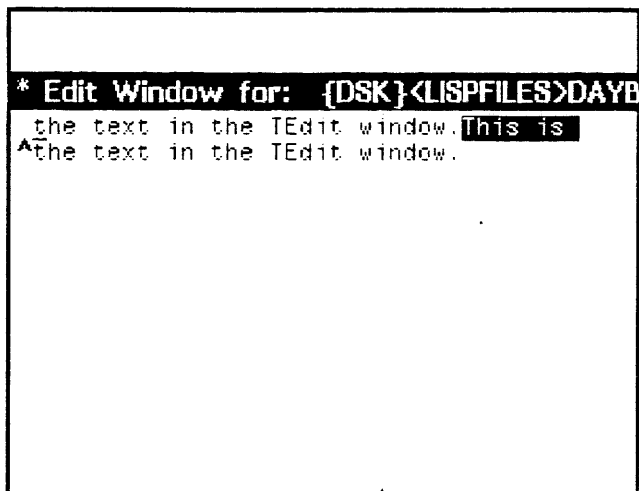
- (4) Release the **COPY** key. The underlined text will now appear both in its original position, and at the caret.

Note: To abort the copy procedure, release the **COPY** key before completing the text selection (i.e. before releasing the right mouse button).

### 23.4.3 Moving Text

To move a block of text from one place in the file to another,

- (1) Position the caret where you want the moved block of text to appear. In the example text, position it at the beginning of the file.
- (2) press and hold the **MOVE** key.
- (3) choose the block of text to be moved. In this example, choose the words "This is ". (See Figure 23.5.)



**Figure 23.5.** The highlighted text will be moved to the caret.

- (4) Release the **MOVE** key. The highlighted text will appear in its new position at the caret.

Note: To abort the move procedure, release the **MOVE** key before completing the text selection (i.e. before releasing the right mouse button).

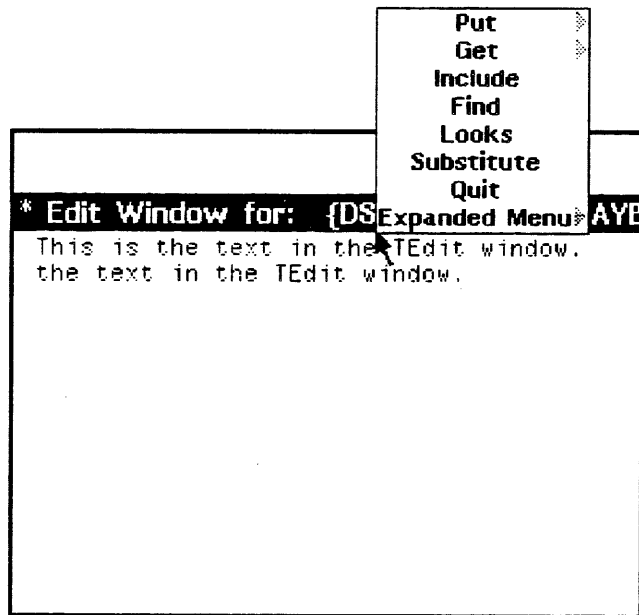
Text can be moved and copied not only within a single TEdit window, but also between them. The same instructions apply in either case.

## 23.5 TEdit Menus

The sections that follow explain commands you can choose from menus specific to TEdit.

### Basic Commands menu

The Basic Commands menu appears when you point the mouse at the title bar of the TEdit Window, and press the middle mouse button. (See Figure 23.6.)



**Figure 23.6.** The TEdit window, with the Basic Commands Menu

Choosing the final item on the Basic Commands menu permanently positions the **Expanded Menu** above the TEdit Prompt Window. Do that and you will see something like Figure 23.7. The expanded menu provides many standard commands.

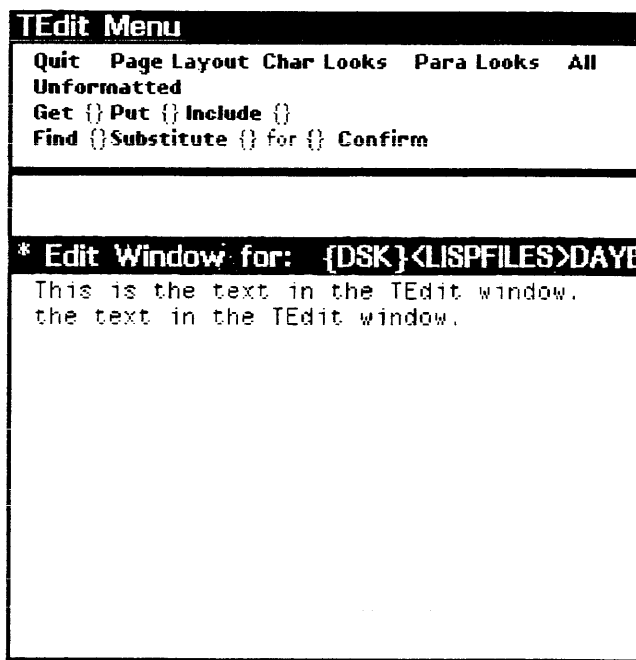


Figure 23.7. The TEdit window, with the Expanded Menu above it.

Note that some of the menu items are followed by "{}". These commands require you to type information between the the brackets before choosing the command. To do this:

- (1) Point with the mouse to the space between the curly brackets;
- (2) press and release the left mouse button. The caret will appear between the brackets;
- (3) type the necessary information. The same edit commands are used to change the text between the curly brackets as the text in the TEdit window.
- (4) execute the command by choosing the command with the mouse.

Both menus are very useful; some of their functions are described below, but please refer to the *Library Packages Manual* for more details.

## 23.5.1 Finding and Substituting Text with TEdit

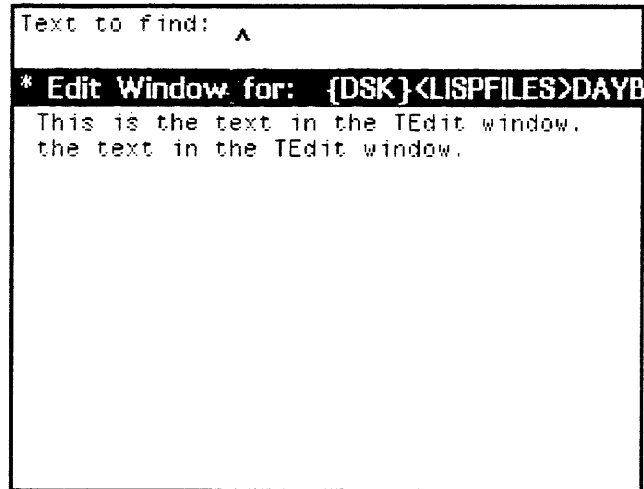
### 23.5.1.1 Finding Text

There are times when you will want to find a certain word, or words, in your TEdit file. You do this with the **Find** command.

To use the **Find** command with the Basic commands menu, or the **Find** key on the left keypad:

- (1) Position the caret in the file where the search should begin;
- (2) Press the **Find** key on the keypad to the left of the keyboard, or choose **Find** from the Basic Commands menu described above;

- (3) You will be prompted to enter the search string. (See Figure 23.8.) If there is a special character in the string, such as a carriage return, type a Control-V before entering it
- (4) Type `CR` to begin the search



**Figure 23.8.** The prompt for the search string from the **F i n d** command

To use the **F i n d** command from the Expanded Menu:

- (1) Type the word to find between the curly brackets beside the **F i n d** command. You do not need to type Control-V before special characters here;
- (2) Position the caret in the file where the search should begin
- (3) Choose **F i n d** from the Expanded Menu.

Note: To abort the search, type either a Control-E or press the **STOP** key.

### 23.5.1.2 Substituting Text

Sometimes you will want to delete a piece of text, and put something new in its place. You can do this with the **S u b s t i t u t e** command.

To replace a large body of text, select the text that needs to be changed. The highlighted text is deleted when you begin to type the replacement text.

To find and replace one or more occurrence of a smaller text string within a selected block of text, use the **S u b s t i t u t e** command from either the expanded menu, or the basic commands menu.

To use the **S u b s t i t u t e** command from the Expanded Menu,

- (1) Choose the text that contains the string(s) to be replaced. For the example, choose the second sentence.
- (2) Type the new text string between the curly brackets following the word **S u b s t i t u t e**. For the example, type "text editor's".

- (3) Type the old text string, the one that is to be replaced, between the curly brackets following the word **for**. For this example, type "TEdit".
- (4) Choose **Confirm** if you would like to verify each substitution. Confirm will then appear in reverse video. (See Figure 23.9.)
- (5) Choose **Substitute**. If you have chosen **Confirm**, you will have to approve each substitution. (See Figure 23.10.) Otherwise, every instance of the old string in the chosen text will be replaced by the new string automatically.

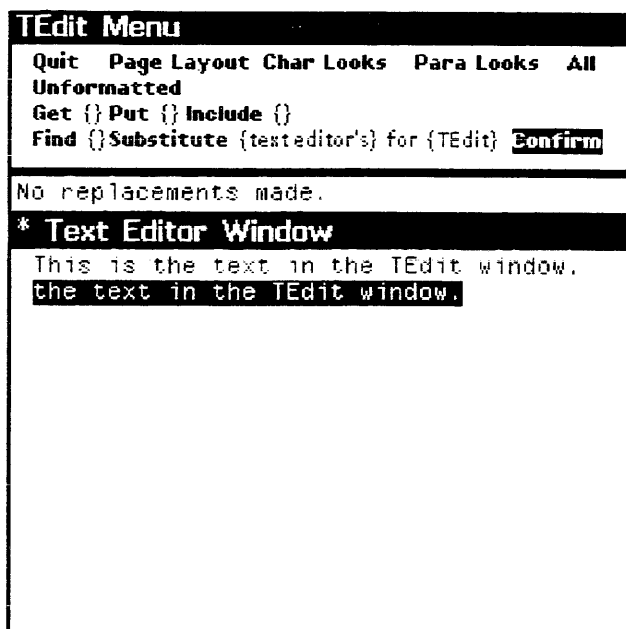


Figure 23.9. Using the **Substitute** command from the Expanded Menu

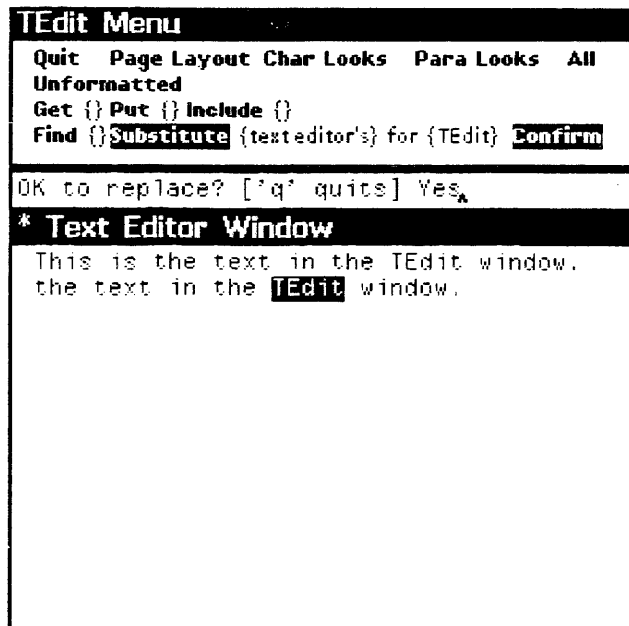


Figure 23.10. Asked to confirm a substitution in the TEdit window

To use the **Substitute** command from the Basic Commands menu,

- (1) Choose the text that contains the instance(s) of the string to be replaced



- (2) Choose the **Substitute** command from the Basic Commands menu
- (3) You will be prompted for the search string and the replace string, and asked whether you would like to confirm each substitution

Note: To stop the Substitute command, type either Control-E, STOP, or Q.

## 23.5.2 Text Formatting

TEdit offers a wide range of possibilities for document formatting. This section explains only the most basic ones. Refer to the *Library Packages Manual* for information on others.

### 23.5.2.1 Choosing Fonts

You can choose the fonts used in a TEdit file. To do this from the Basic Commands Menu:

- (1) Choose the text that you would like to see in a different font;
- (2) Choose **Looks** from the Basic Commands menu;
- (3) A series of three menus will appear, one after the other. The first will offer a choice of fonts (see Figure 23.11.), the second a choice of the properties of the font (e.g. italics, or bold), and the third, the font size. Either choose one of the items, or click the left mouse button outside of the menu to leave the default setting unchanged

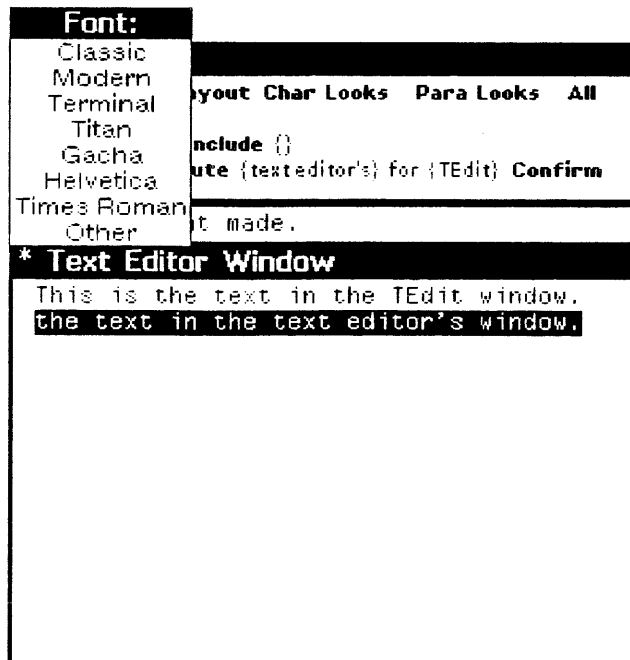


Figure 23.11. The font choices from the **Looks** command

The fonts can also be changed by choosing **Char Looks** from the expanded menu. This will cause another menu to appear. (See Figure 23.12.)

To see the font currently being used in a selected block of text, choose **SHOW** from the menu.

To use this menu to change the fonts in the selected text,

- (1) Choose the text you would like to see in a different font. For this example, choose the second sentence.
- (2) Choose from the **Character Looks Menu** the type of font, and the type of font from the "props" choices. If you do not choose a font, or a font property, the current one is used. For the example, choose "Italic". (See Figure 23.12.)
- (3) Choose **APPLY** and the font will be changed.

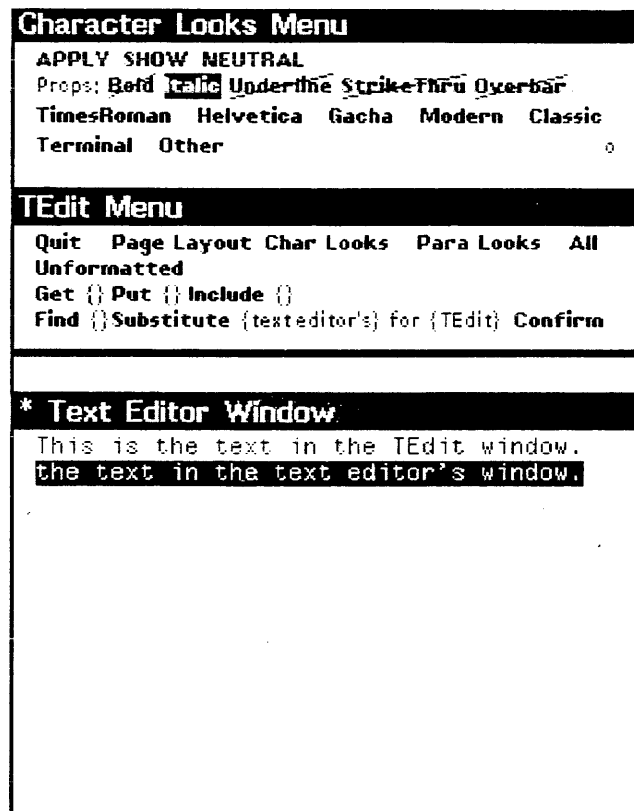


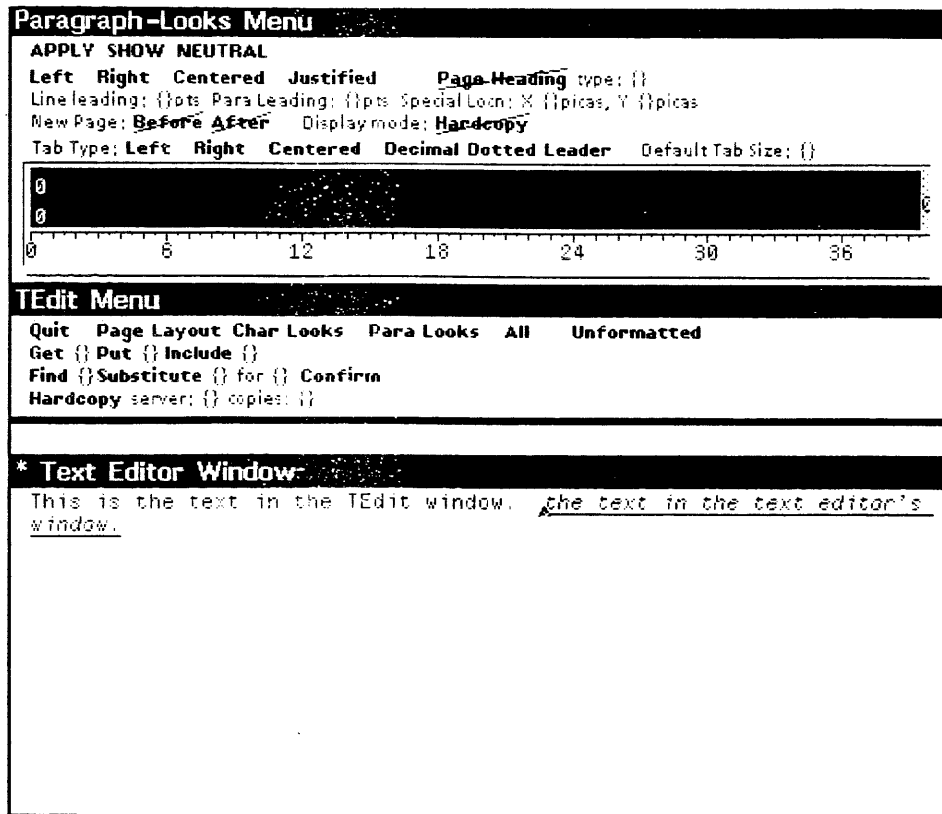
Figure 23.12. Ready to change the font of the second sentence to "Italic"

### 23.5.2.2 Paragraph Formatting

You can format paragraphs with the **Paragraph-Looks Menu**, shown in Figure 23.13. To bring up this menu, choose **Para Looks** from the Expanded Menu. Although the new menu looks complicated, the basics are easy to learn.

To see the current settings for some paragraph, first select the paragraph. Choose **SHOW** from the top line of the menu.

To change the settings, begin by choosing a piece of text to work with.



**Figure 23.13.** The Paragraph Looks menu is above the TEdit Expanded Menu

#### Line Justification

To determine how each line of the chosen text is placed on the page, choose one of the following items from the second line of the **Paragraph-Looks Menu**. **Left** gives a ragged right margin, **Right** gives a ragged left margin, **Centered** centers each line, and **Justified** gives both left and right justification so neither margin is ragged. Choose one with the mouse. When your choice is in reverse video, choose **APPLY** to see how the chosen text is affected.

#### Page Breaks

You can break your text into pages both before and after a paragraph. First, move the caret to the appropriate paragraph. Choose either **Before** or **After** from the fourth line of the **Paragraph-Looks Menu**. When you choose **APPLY**, a grey box at the front of the paragraph marks a page break. (See Figure 23.14.)

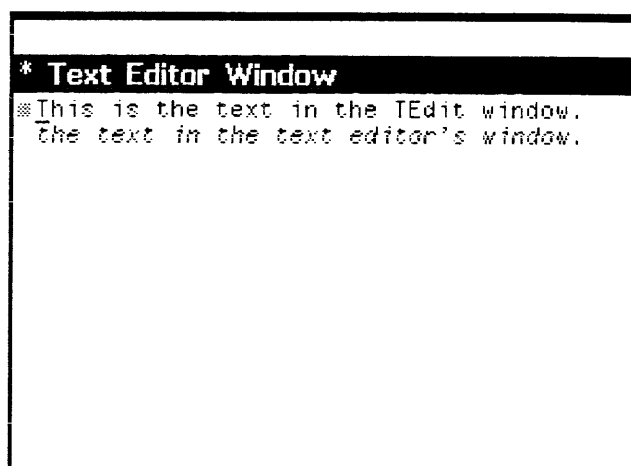


Figure 23.14. The grey box on the far left side of the TEdit window marks a page break.

### 23.5.3 Adding Bitmaps and Sketches to your TEdit File

TEdit allows you to easily add snapshots of the screen, and Sketches to your files. Sketch is a Xerox package that was developed for constructing pictures. Unlike bitmaps, (See Chapter 29.) you do not need to draw every pixel for the shape you want. For instructions on how to use Sketch, see Chapter 35.

#### 23.5.3.1 Adding a Bitmap to your TEdit file

To add a snapshot of the screen to your file,

- (1) Position the caret in the TEdit file where you want the snapshot to appear.
- (2) Press and hold the shift key. Move the mouse cursor into the grey background of the screen, and press the right mouse button. A menu with a single choice, SNAP, will appear. Release the shift key. Choose SNAP from the menu;
- (3) The mouse cursor will change to the prompt to sweep out a window:



Figure 23.15.

Position the mouse cursor at a corner of the region you would like to snap;

- (4) Press and hold the left mouse button. Sweep out the snapshot window. When you are satisfied with the snapshot, release the left mouse button, and it will appear in your TEdit file.

Once the bitmap is in your file, it is treated like a single character. That means that you can move, copy, or delete it just like you do any other character. There are also various operations you can perform on bitmaps in your TEdit file (e.g. trimming and editing). For more information about these, please see the *Library Packages Manual*.

### 23.5.3.2 Adding a Sketch to your TEdit file

---

As mentioned above, Sketch is explained in this primer in Chapter 35. For even more information, see the *A User's Guide to Sketch*. To add a Sketch to a TEdit file,

- (1) Position the caret in the TEdit file where you want the snapshot to appear.
- (2) Press and hold the shift key. Move the mouse cursor into the Sketch window. The control boxes of the Sketch will appear. Click the left mouse button twice in any control box to copy the entire Sketch into your TEdit file.

Like bitmaps, once the Sketch is in your file, it is treated like a single character. That means that you can move, copy, or delete it just like you do any other character.

Once the Sketch is in your file you can still edit it. Simply point to the Sketch with the mouse cursor, and press the right mouse button. A menu with the single item, Edit sketch, will appear. When you choose this item, a Sketch window will open for you to make any changes. For more information, see *A User's Guide to Sketch*.

## 23.5.4 Getting and Including Files

---

The **Get** and **Include** commands address files of text.

### 23.5.4.1 Get

---

**Get** opens a file you want to edit, and brings it into the TEdit window. A file brought into the TEdit window with the command **Get** replaces any file that was previously being edited. **Get** appears in both the Expanded and Basic Commands menus.

When you choose **Get** from the Basic Commands menu, you will be prompted for the file name.

To use **Get** from the Expanded Menu, type the filename between the curly brackets following the word **Get**. Choose **Get**.

In either case, if you have not saved the file you are currently working on, you will have to confirm the **Get** by clicking the left button. Click the left button only if you want to begin work on the new file without saving the changes made to the file that is currently in the window.

### 23.5.4.2 Include

---

**Include** also appears in both menus. It adds a file to the TEdit window, but unlike **Get** it does not affect the file that is being **Included** in any other way. **Include** simply reads in the specified file, and adds it to the current document at the caret.

If you chosen it from the Basic Commands menu, you will be prompted for the file name.

If it is chosen from the Expanded Menu, the filename must be typed between the curly brackets following the word **Include** before choosing it.

### **23.5.5 Saving and Printing Files**

---

Save your files often to decrease the chances of accidentally losing pieces of your work.

To save the file, choose **Put** from either the Basic Commands menu, or the Expanded Menu. Once again, if the command is chosen from the Basic Commands menu you will be prompted for the file name. If you choose it from the Expanded Menu, first type the file name between the curly brackets to the right of the command.

To print your file, choose **Hardcopy** from either the default right button menu, or from the Expanded Menu. If it is chosen from the right button menu, there is the further choice of sending the formatted output to a file, or to a printer. If the file is to be printed often, you may want to send the output to a file. It takes time to format the document for printing, and if the formatted file is saved, you will not have to wait each time.

Alternatively, using the **Hardcopy** command in the Expanded Menu gives you some other options. You can send the output to any printer available, not just the default printer, and you can specify the number of copies to print. As usual, these specifications must be typed between the curly brackets before choosing this command.

Sending a file to the Epson FX80 printer is covered in Chapter 19.

[This page intentionally left blank]

## 24. RECORDS MAY BE YOUR FAVORITE DATA STRUCTURE!

---

A record is a data structure that consists of numerous fields. Each field, can have a simple value such as a list or an atom, or a more complex value, such as a window or a menu (which is also a record), or a function.

There are two parts to working with records. The first part involves giving the record a name, and deciding the field names contained in the record. This is referred to as creating a record definition. The second part involves using the record definition as a template, and giving each of the record's fields a specific value. The second part is referred to as creating a record instance.

Consider this example: you want to buy a car. Your record would be named car, and you would define a record using this list of fields: make; model; year; mileage; sticker price.

Now you go out and do your research. The first car you find is a Plymouth, Champ, 1979, 87,500 miles, \$1700. When you input these values into a copy of the record fields, you have created a record instance.

Next you find an Oldsmobile, Cutlass, 1986, 0 miles, \$16,000. When you input these values into a copy of the record definition, you have created a second record instance.

---

### 24.1 Interlisp Record Primitives

---

The function **RECORD** creates a record definition. This record definition can be considered a template for creating, accessing and storing into record instances.

**(RECORD record-name (list of fields))**

To declare a record definition for an employee, you might need his name and social security number. Declaring a record is done with the function **RECORD**. (See Figure 24.1.)



```

Interlisp-D Executive Window
NIL
33+(RECORD EMPLOYEE (NAME SSN))
EMPLOYEE
34+

```

Figure 24.1. Record Definition of Record Named EMPLOYEE

**RECORD** does not create any record instances, however. It only makes the interpreter aware that you will be using a record definition called **EMPLOYEE** that contains two fields.

**create** is the function that actually creates record instances.

```
(create record-name <field value
                    field value...>)
```

**create** takes the name you gave to the record definition and returns an instance of that record. After the record definition name, you will specify the values for each field. (See Figure 24.2.)

```

Interlisp-D Executive Window
NIL
32+(SETQ WORKER (create EMPLOYEE
                       NAME + "Bill Smith"
                       SSN + 123456789))
("Bill Smith" 123456789)
33+

```

Figure 24.2. Creating and Initializing a Record

To initialize a field, type the field name, followed by left arrow (←) (the left arrow should be surrounded by spaces), followed by the given value. Note that standard evaluation rules for Interlisp-D are not followed by the function **create**. The values supplied for the record's fields are evaluated, but the record name and the field names are not evaluated. (See Figure 24.5.) **create** was written this way to be convenient to use. The functions or atoms supplied as values for the fields are immediately evaluated, and the field contains only the result of that evaluation.

The code in Figure 24.2 returned a new instance of the record definition **EMPLOYEE** for Bill Smith. Note that in the example, the atoms **EMPLOYEE**, **NAME**, and **SSN** were not evaluated. However, "Bill Smith" and 123456789 were evaluated. Any fields which are not given values in the **create** expression are assigned the value **NIL**.

In general **create** should appear inside an assignment expression such as **SET**, **SETQ**, or **PUTPROP**. It may also appear in a **PROG**'s temporary variable lists or when binding argument lists to functions.

**fetch** is an Interlisp-D command that allows you to access the contents of a field in a record. The syntax of the **fetch** command is:

*(fetch fieldname of record-instance)*

The *fieldname* argument will not be evaluated; the *record-instance* argument will be evaluated. Figure 24.3 demonstrates the use of **fetch** with the **WORKER** example used earlier. The field names **NAME** and **SSN** are never evaluated and therefore not quoted. In fact, using **QUOTE** will cause an error! **WORKER** was evaluated, which is what we want. **WORKER** is bound to the desired record instance.

```

Interlisp-D Executive Window
NIL
34+(fetch NAME of WORKER)
"Bill Smith"
35+(fetch SSN of WORKER)
123456789
36+

```

Figure 24.3. **fetch** from Records

**replace** is an Interlisp command that allows you to modify the contents of the record-instance's fields. **replace** works similar to **fetch**, except that it requires a new value in order to overwrite the old value. This extra argument is placed last:

*(replace fieldname of record instance with newvalue).*

An example is shown in Figure Figure 24.4.

```

Interlisp-D Executive Window
NIL
37+(replace SSN of WORKER with 987654321)
987654321
38+WORKER
("Bill Smith" 987654321)
39+(fetch SSN of WORKER)
987654321
40+

```

Figure 24.4. Using **replace** and **fetch** with Records

## 24.2 Example

Following is a line by line explanation of the example in Figure 24.5, which demonstrates the record package facilities:

```

Interlisp-D Executive Window
NIL
43+(RECORD ISOTOPE
      (SYMBOL ISOTOPENUMBER ISOTOPEWEIGHT))
ISOTOPE
44+(SETQ CARBON (create ISOTOPE
                      SYMBOL + 'C
                      ISOTOPENUMBER + 6
                      ISOTOPEWEIGHT + 12])
(C 6 12)
45+(fetch SYMBOL of CARBON)
C
46+(fetch ISOTOPENUMBER of CARBON)
6
47+(fetch ISOTOPEWEIGHT of CARBON)
12
48+(replace ISOTOPEWEIGHT of CARBON
           with 14)
14
49+(fetch ISOTOPEWEIGHT of CARBON)
14
50+(fetch SYMBOL of CARBON)
C
51+(fetch ISOTOPENUMBER of CARBON)
6
52-CARBON
(C 6 14)
53+

```

Figure 24.5. Use of **RECORD**, **create**, **fetch**, and **replace**.

- Line 43 declares a record definition called **ISOTOPE**. Each record instance will contain three fields representing the symbol, isotope number, and isotope weight of the isotope. The system responds by saying that **ISOTOPE** is now a recognized record definition.
- Line 44 assigns **CARBON** a value: a record instance with the properties of Carbon 12. Since **SETQ** returns its second argument, the resulting list shows how a record is stored in list form.
- Lines 45 - 47 demonstrate **fetch**. The contents of the field being **fetch**ed are returned.
- Line 48 replaces the **ISOTOPEWEIGHT** with 14, so now **CARBON** represents a new isotope.
- Line 49 verifies that the change in **CARBON**'s **ISOTOPEWEIGHT** was made
- Lines 50 and 51 show that the change to one field of a record did not change any other fields of that record.
- Line 52 shows the value of the atom **CARBON**. This result is the record instance that **fetch** and **replace** actually act on. The instance is stored as the value of the **CARBON**.

## 24.3 A Few Tips

If you are using several record definitions, avoid using the same field name in two record definitions until you become familiar with the manual. otherwise the system will try to access things in a way that can cause a break. If it is necessary to use the same

field name, refer to your record definition name before you reference the field. So, for example, `(fetch (ISOTOPE SYMBOL) OF CARBON)` says "I want the SYMBOL field from the record instance CARBON of the record definition ISOTOPE."

Although you may create lists which can be properly handled by the record package or vice versa, it is a good idea to only use `create`, `fetch` and `replace` when working with records. This improves readability of your code, and ensures that changes in the Interlisp-D record package or changes you may make to your record definition will not affect your code.

A record is really just a special list. There is no implicit type-checking in the record package, so a list you give to `fetch` and `replace` will be destructively modified as if it were a record instance, regardless of whether or not it has anything to do with records. Figure 24.6 shows poor programming even though no fatal errors result. The example assumes the EMPLOYEE record that was defined earlier in the chapter is a part of the Interlisp-D environment.

```

Interlisp-D Executive Window
NIL
60+(fetch NAME of '(1 2 3))
1
61+(SETQ TEMP '(1 2 3))
(1 2 3)
62+(fetch NAME TEMP)
1
63+(replace SSN TEMP 5)
5
64+TEMP
(1 5 3)
65+

```

Figure 24.6. Sloppy Use of Records and Lists

You can type-check your records by using the command `TYPERECORD` instead of `RECORD`. When record definitions are declared with the command `TYPERECORD`, the `create`, `fetch`, and `replace` functions are the same as for regular records. There is also a function `TYPE?` which determines if its argument is of the right type. If `EMPLOYEE` had been created as a `TYPERECORD`, you could find out if `WORKER` is an instance of the `EMPLOYEE` record definition by asking:

```
(TYPE? EMPLOYEE WORKER)
```

which returns a non-`NIL` value if `WORKER` is an `EMPLOYEE` record instance. Typically it will be used this way:

```
(AND (TYPE? EMPLOYEE WORKER) (fetch SSN WORKER))
```

If the type check fails, the `fetch` is not evaluated; if it succeeds, the `fetch` is evaluated and a value is returned. As in `create`, the record definition name (`EMPLOYEE` in this example) is not evaluated. (See Figure 24.7.)

```
Interlisp-D Executive Window
NIL
67←(TYPE-RECORD PLAYER
      (PIECE-COLOR BOARD-POSITION))
PLAYER
68←(SETQ PLYR1
      (create PLAYER
              (PIECE-COLOR + 'RED
              BOARD-POSITION + 'SQUARE1)
              (PLAYER RED SQUARE1)
              (TYPE? PLAYER PLYR1)
              T
              (TYPE? PLAYER WORKER)
              NIL
              (AND (TYPE? PLAYER PLYR1)
                    (replace BOARD-POSITION of PLYR1
                              with 'SQUARE2)))
      SQUARE2
      72←PLYR1
      (PLAYER RED SQUARE2)
      73←
```

Figure 24.7. **TYPE-RECORD** and Explicit Type Checking

Other variations on record definitions are in the *Interlisp Reference Manual*, Volume 1, Section 8.

# 25. LOCAL VARIABLES - USING LET AND PROG

Local variables are variables created within a function, and accessed and changed only within that function. Contrast a local variable and a global variable:

- Local variables are handled more efficiently by the interpreter and the compiler;
- local variables, especially when well named, can make your code easier to understand, and to change;
- it is better style to use local variables, and to minimize the use of global variables.

This chapter will explain the use of Interlisp-D forms for creating local variables.

## 25.1 LET

Interlisp-D provides several ways to create local variables. One is **LET**. Its use will be illustrated with an example, the function **MY.ADDPROP**. This function maintains a list of values for each property of an atom. When a new value is added with **MY.ADDPROP**, the function first checks to see whether the new value is a member of the list of values. If it is, the current list of values is returned. If it is not already a member of the list, it is added to the list of values for that property.

Without using **LET** the function **MY.ADDPROP** can be written like this:

DEdit of function MY.ADDPROP	EditOps
<pre>(LAMBDA (ATOM PROPERTY VALUE)   (IF (MEMBER VALUE (GETPROP ATOM PROPERTY))       THEN (GETPROP ATOM PROPERTY)       ELSE (PUTPROP ATOM PROPERTY                   (CONS VALUE                         (GETPROP ATOM                           PROPERTY))))))</pre>	<ul style="list-style-type: none"><li>After</li><li>Before</li><li>Delete</li><li>Replace</li><li>Switch</li><li>()</li><li>()out</li><li>Undo</li><li>Find</li><li>Swap</li><li>Reprint</li><li>Edit</li><li>EditCom</li><li>Break</li><li>Eval</li><li>Exit</li></ul>

**Figure 25.1.** The function **MY.ADDPROP** can be written like this, without using the function **LET**

Notice that the function call (**GETPROP ATOM PROPERTY**) appears several times. Use the function **LET** to bind a local variable to the value returned from this function call, so that it only needs to be done once. It will be easiest to define this function in the Interlisp-D Structure Editor, **DEdit**. To do this,

- (1) Type:

```
(DF MY.ADDPROP)
```

in the Interlisp-D Executive window. When you are asked whether you want to edit a dummy definition, type **Y**. A standard template for writing functions will appear. (See Figure 25.2.)

This chapter will assume that you know how to use **DEdit**, covered in Section 11.3, Page 11.4.

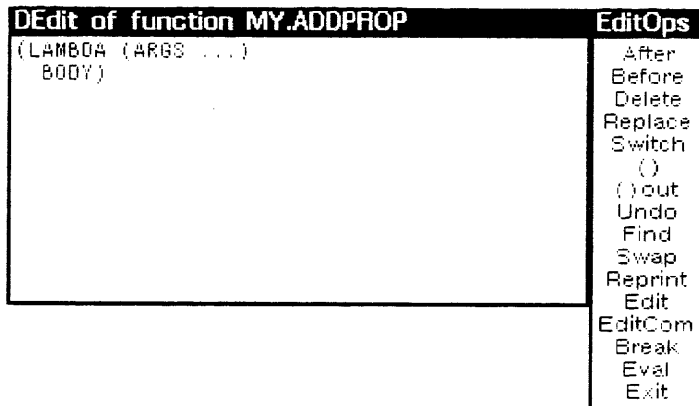


Figure 25.2. The standard template supplied by **DEdit** for defining functions.

- (2) Change the template's formal parameter list so that the function expect three arguments: **ATOM**, **PROPERTY**, and **VALUE**.
- (3) The body of the function will be the **LET** expression. The first argument that **LET** expects is a list of lists. Each list is a list containing a local variable and its initial value.

In this example, there is only one local variable, **OLDVALUES**. This variable's value will be the value returned from the function call, (**GETPROP ATOM PROPERTY**). Add this to your function definition, so that it looks like this:

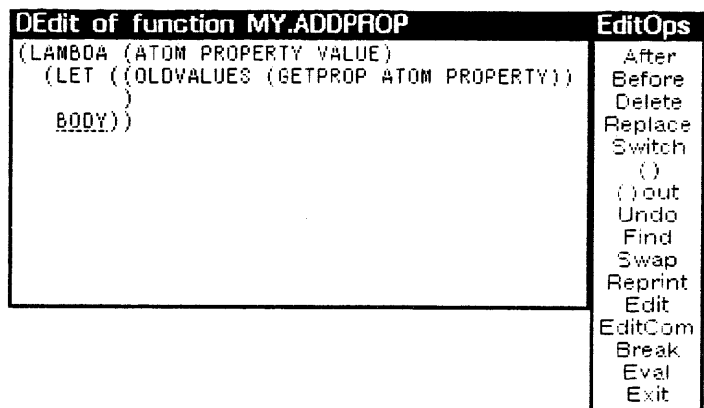


Figure 25.3. **MY.ADDPROP**'s **LET** with its local variable. The body of the **LET** has not yet been added.

- (4) All forms after the list of local variables and their values make up the body of the **LET**. In this example, the body of the **LET** is the

same as the body of the function without LET, except that the local variable, **OLDVALUES**, is used instead of the calls to **GETPROP**.

Add the body to your function, so that it looks like this:

DEdit of function MY.ADDPROP	EditOps
(LAMBDA (ATOM PROPERTY VALUE)	After
(LET ((OLDVALUES (GETPROP ATOM PROPERTY))	Before
)	Delete
(if (MEMBER VALUE OLDVALUES)	Replace
then OLDVALUES	Switch
else (PUTPROP ATOM PROPERTY	( )
(CONS VALUE OLDVALUES))	( )out
)))	Undo
	Find
	Swap
	Reprint
	Edit
	EditCom
	Break
	Eval
	Exit

Figure 25.4. **MY.ADDPROP**, written with **LET**.

Each expression in the body of the **LET** is evaluated, and like a **LAMBDA**, the value of the last expression in the body is returned as the value of the **LET**.

- (5) Exit from the editor, and try your function. Type:

```
(MY.ADDPROP 'APPLE 'COLORS 'RED)
```

and

```
(MY.ADDPROP 'APPLE 'COLORS 'YELLOW)
```

Check the value of the property **COLORS** of the atom **APPLE** by typing:

```
(GETPROP 'APPLE 'COLORS)
```

Both colors will be in the list. Type:

```
(MY.ADDPROP 'APPLE 'COLORS 'RED)
```

again. **RED** will not be added to the list for the second time. When you check the value of the property **COLORS** of the atom **APPLE**, the list (**YELLOW RED**) should be returned, as in the following figure:

```

Interlisp-D Executive Window
NIL
46+(GETPROP 'APPLE 'COLORS)
(YELLOW RED)
47+

```

Figure 25.5. The value of the property **COLORS** of the atom **APPLE**

For more information about **LET**, see the *Library Packages Manual*.

## 25.2 PROG

The same example can be used to illustrate the use of the function **PROG**. There are differences between **PROG** and **LET**,



which you should note as you go through the example. Although LET is the preferred form, there are times when the extra flexibility provided by PROG is needed.

Once again, it will be easiest to define the example function in the Interlisp-D Structure Editor, DEdit. To do this,

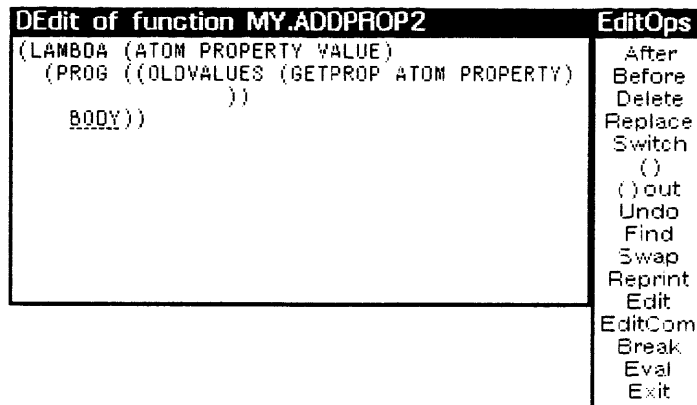
- (1) Type:

**(DF MY.ADDPROP2)**

in the Interlisp-D Executive window. When you are asked whether you want to edit a dummy definition, type Y. A standard template for writing functions will appear. (See Figure 25.2.)

- (2) Change the template's formal parameter list so that the function expect three arguments, **ATOM**, **PROPERTY**, and **VALUE**.
- (3) The body of the function will be the **PROG** expression. Like **LET**, the first argument that **PROG** expects is a list that will allow it to set up local variables.

In this example, there is only one local variable, **OLDVALUES**. This variable's initial value is the value returned from the function call, **(GETPROP ATOM PROPERTY)**. Add this to your function definition, so that it looks like this:



**Figure 25.6.** MY.ADDPROP's LET with its local variable. The body of the LET has not yet been added.

**NOTE:** Unlike LET, PROG's local variable list can contain lists or atoms. Lists contain two items, the first is the local variable's name, the second is its initial value. Atoms are the local variables' names, and each one is bound to **NIL**.

- (4) All forms after the list of local variables and their values, make up the body of the **PROG**. In this example, the body of the **PROG** is almost the same as the body of the function with **LET**, except that the **if** is inside the function **RETURN**. Add the body to your function, so that it looks like this:

Edit of function MY.ADDPROP2	EditOps
<pre>(LAMBDA (ATOM PROPERTY VALUE)   (PROG ((OLDVALUES (GETPROP ATOM PROPERTY)     ))     (RETURN       (if (MEMBER VALUE OLDVALUES)         then OLDVALUES         else (PUTPROP ATOM PROPERTY           (CONS VALUE OLDVALUES             ))))))</pre>	<pre>After Before Delete Replace Switch () ()out Undo Find Swap Reprint Edit EditCom Break Eval Exit</pre>

Figure 25.7. MY.ADDPROP, written with PROG.

NOTE: Unlike LET, the PROG does not return the value of the last expression in its body. The expressions in the body of the PROG are evaluated in order until the function RETURN is reached.

The function RETURN takes one argument. The value of this argument is returned as the value of the PROG. RETURN is needed for the PROG to return a value. Without RETURN, the PROG returns NIL. No other expressions in the body of the PROG are evaluated after the RETURN function is evaluated.

- (5) Exit from the editor, and try your function. Type:

```
(MY.ADDPROP2 'GRAPE 'COLORS 'GREEN)
```

and

```
(MY.ADDPROP2 'GRAPE 'COLORS 'PURPLE)
```

Check the value of the property COLORS of the atom GRAPE by typing:

```
(GETPROP 'GRAPE 'COLORS)
```

Both colors will be in the list. Type:

```
(MY.ADDPROP2 'GRAPE 'COLORS 'PURPLE)
```

again. PURPLE will not be added to the list for the second time. When you check the value of the property COLORS of the atom GRAPE, the list (PURPLE GREEN) should be returned, as in the following figure:

```
Interlisp-D Executive Window
NIL
62>(GETPROP 'GRAPE 'COLORS)
(PURPLE GREEN)
63>
```

Figure 25.8. The value of the property COLORS of the atom APPLE

PROG can also be used for looping, but it is better style to use the appropriate Interlisp-D interactive statement (See Chapter 26.) For more information about PROG, see the *Interlisp-D Reference Manual*, Volume I, Page 9.8.

## 25.3 Parallel versus Sequential Variable Binding

Both **PROG** and **LET** bind their local variables in parallel. This means that all the variable values are set at once, not one after the other. When the local variables are initialized, you cannot compute the value of one local variable using the value of another local variable.

As an example, see Figure 25.9. It shows a function, **RANDOM-SQRT**, that expects one argument, a list of positive integers. First, a random number between one and the length of the list of integers is selected. The positive integer at that position in the list of integers is selected, and its square root returned.

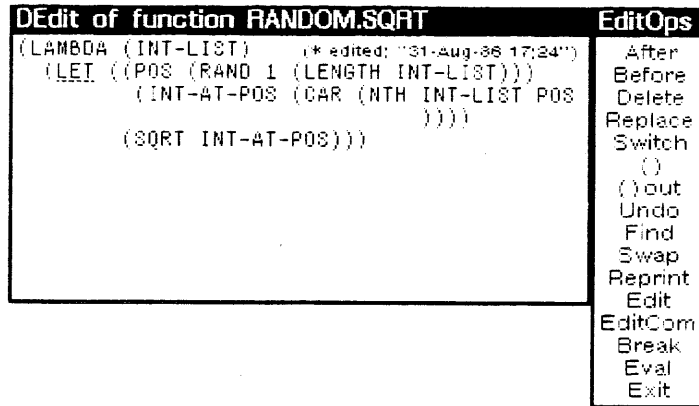


Figure 25.9. The incorrect definition of the function **RANDOM.SQRT**

Unfortunately, the function is written incorrectly. When the variables are initialized, the value of **INT-AT-POS** depends on the value of **POS**. When the function is run, an error is generated, as shown below:

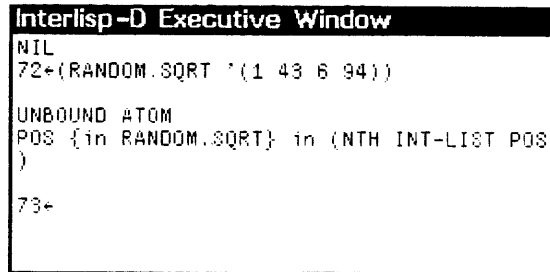


Figure 25.10. The error generated by the incorrect function definition

The same thing would happen if a **PROG** was used instead of **LET**. What is needed for this function is sequential binding. Interlisp-D provides **LET\*** and **PROG\*** for this purpose. They work like **LET** and **PROG**, respectively, except that they bind their local variables sequentially, one after the other, instead of in parallel.

### 25.3.1 LET\*

Write the example function, **RANDOM.SQRT** using the function **LET\***. As usual, it is easiest to define the function in the Interlisp-D Structure Editor, **DEdit**. To do this, type:

(DF RANDOM.SQRT)

in the Interlisp-D Executive window. When you are asked whether you want to edit a dummy definition, type Y. A standard template for writing functions will appear. (See Figure 25.2.)

Change the function template so that your function looks like this:

DEdit of function RANDOM.SQRT	EditOps
<pre>(LAMBDA (INT-LIST) (* edited: '31-Aug-88 17:34")   (LET* ((POS (RAND 1 (LENGTH INT-LIST)))         (INT-AT-POS (CAR (NTH INT-LIST POS)                           )))     (SQRT INT-AT-POS)))</pre>	After Before Delete Replace Switch () ()out Undo Find Swap Reprint Edit EditCom Break Eval Exit

Figure 25.11. The example function, **RANDOM.SQRT**, written with a **LET\***, so that the variables are bound one after the other.

Now run the function. Type:

```
(RANDOM.SQRT '(4 36 81 9))
```

No error is generated!

### 25.3.2 PROG\*

Now write the example function again, but this time use the function **PROG\***, and call it **RANDOM.SQRT2**. As usual, it is easiest to define the function in the Interlisp-D Structure Editor, DEdit. To do this, type:

(DF RANDOM.SQRT2)

in the Interlisp-D Executive window. When you are asked whether you want to edit a dummy definition, type Y. A standard template for writing functions will appear. (See Figure 25.2.)

Change the function template so that your function looks like this:

DEdit of function RANDOM.SQRT2	EditOps
<pre>(LAMBDA (INT-LIST)   (PROG* ((POS (RAND 1 (LENGTH INT-LIST)))          (INT-AT-POS (CAR (NTH INT-LIST                               POS))))     (RETURN (SQRT INT-AT-POS))))</pre>	After Before Delete Replace Switch () ()out Undo Find Swap Reprint Edit EditCom Break Eval Exit

**Figure 25.12.** The example function, **RANDOM.SQRT**, written with a **PROG\***, so that the variables are bound one after the other.

Now run the function. Type:

```
(RANDOM.SQRT '(4 36 81 9))
```

No error is generated! For more information about **PROG\*** see the *Interlisp-D Reference Manual*, Volume I, Page 9.9.

Iterative statements non-recursively repeat a set of steps. This section provides different iterative statement examples for you to follow. Iterative statements are implemented with the CLISP facility discussed in Section 13.1. For now you don't need to know anything about CLISP to use iterative statements.

## 26.1 General Structure and Use

Iterative Statements provide a simple method of looping in Interlisp-D. Using the statements improves readability of code and provides standard paradigms for loop organization.

There are three parts to an iterative statement. The first part specifies the basic structure of the loop. For example, looping through integers from 1 to 10, or through the elements of a list. The second part specifies how the values returned by each iteration are treated. The third part specifies the body of the loop.

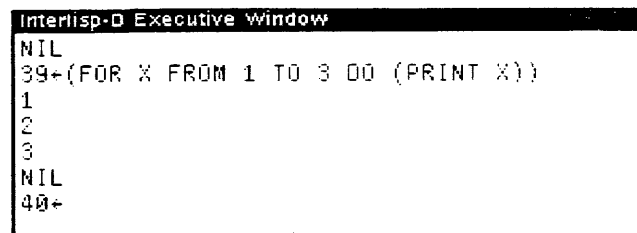
Consider the following example of a fairly generic loop:

```
(FOR X FROM 1 TO 10 looping-keyword
  body-of-the-loop)
```

In this example, FOR x from 1 to 10 is a loop structure. The variable x is a local variable, local to the iterative statement. In the next few paragraphs we discuss ways to use the *looping-keyword* and the *body-of-the-loop*.

The keywords tell the loop how to return the values:

**DO** Evaluates the expressions in the body of the loop. Returns NIL.



```
Interlisp-D Executive Window
NIL
39+(FOR X FROM 1 TO 3 DO (PRINT X))
1
2
3
NIL
40+
```

Figure 26.1. A FOR loop using the keyword **DO**

**COLLECT** Like **DO**, except that the value of each iteration is gathered into a list and the list is returned.

```

Interlisp-D Executive Window
NIL
41+(FOR X FROM 1 TO 3
      COLLECT (TIMES X 3))
(3 6 9)
42+

```

**Figure 26.2.** A **FOR** loop with the keyword **COLLECT**

**THEREIS** Stops iteration when the body of the loop evaluates to a non-NIL value. Returns the value of the local iteration variable when the loop stops.

```

Interlisp-D Executive Window
NIL
43+(FOR X FROM 1 TO 3
      THEREIS (GREATERP (TIMES X 5) 9))
2
44+

```

**Figure 26.3.** The iterative statement **FOR** with the keyword **THEREIS**

One more useful feature is having the **FOR** loop able to count by steps other than one. This is done with **BY**.

**BY** Specifies increments for iteration. Increments can be positive or negative. **BY** determines only the increment of the loop counter. It is used with other keywords that determine the value the loop returns.

```

Interlisp-D Executive Window
NIL
88+(FOR X FROM 1 TO 10 BY 5 DO (PRINT X))
1
6
NIL
89+(FOR X FROM 10 TO 1 BY -5 DO
      (PRINT X))
10
5
NIL
90+

```

**Figure 26.4.** The keyword **BY** used with both a positive and a negative increment. Note it is used with the keyword **DO**

## 26.2 Local Variables

Local variables, unlike global ones, are variables known only in one expression. The local variable used for iteration, *X* in the above examples, need not be declared or initialized. Additional local variables may be specified within an iterative statement by using the CLISP word **BIND**. Initialization of local variables is accomplished by using the format

*Var* ← *Expression*

with **BIND**.

For example, to expand on the example in Figure 26.4 by substituting a local constant for the multiplier 5 and the threshold 1, see Figure 26.5.

```

Interlisp-D Executive Window
NIL
54+(FOR X FROM 1 TO 3
      BIND (MULTIPLIER + 5)
          (THRESHHOLD + 9)
      THEREIS (GREATERP
              (TIMES X MULTIPLIER)
              THRESHHOLD))
2
55+

```

Figure 26.5. Using **BIND** to initialize local variables.

See additional examples in the section on conditional looping at the end of this chapter, Page 26.8.

## 26.3 Iteration On Lists

The easiest and most widely used iteration is on successive CARs of a list. (For an example of this type of iteration, see Figure 26.6.)

**IN** is the CLISP word that sets the iterative variable **X** to the CAR of the current tail for each iteration.

```

Interlisp-D Executive
NIL
100+(SETQ MYLIST '(This is a test.))
(This is a test.)
1+(FOR X IN MYLIST DO (PRINT X))
This
is
a
test.
NIL
2+
X

```

Figure 26.6. Iteration on successive **CARs** of a list with **IN**.

**ON** is used to set the iterative variable **X** to the successive CDRs of a list. (See Figure 26.7.)



```

Interlisp-D Executive
NIL
4+(FOR X ON MYLIST DO (PRINT X))
(This is a test.)
(is a test.)
(a test.)
(test.)
NIL
5+

```

Figure 26.7. Iteration on successive CDRs of a list with **ON**.

## 26.4 Parallel Iteration

**AS** is used between clauses when lists and/or numerical iteration should happen in parallel. In other words, X has many values, and N has many values. With **AS** between the two clauses, when X changes, so must N, and the two do so independent of the values of the other.

All iteration will stop when any of the clauses complete. Because MYLIST in Figure 26.8 has only 4 values, the 1 to 5 iteration is not finished, and the loop completes after 4 cycles.

```

Interlisp-D Executive
NIL
10+(FOR X IN MYLIST AS Y FROM 1 TO 5 DO
      (PRINT X) (PRINT Y))
This
1
is
2
a
3
test.
4
NIL
11+

```

Figure 26.8. The loop only iterates through 4 cycles, because there are only 4 items in MYLIST.

The iteration will stop at the end of the shortest clause, no matter what its position in the iterative statement. Note, for example, that exchanging the position of X and Y in the iterative statement does not affect how it runs, or when it stops running:

```

Interlisp-D Executive Window
NIL
77+(FOR Y FROM 1 TO 5 AS X IN MYLIST DO
      (PRINT X) (PRINT Y))
This
1
is
2
a
3
test.
4
NIL
78+

```

**Figure 26.9.** The loop only iterates through 4 cycles.

Note in the following example, that the clauses are really independent. As is shown in Figure 26.10, you cannot say, for instance,

```
(FOR X ON MYLIST AS Y ON (CDR X) DO ...)
```

```

Interlisp-D Executive
NIL
7+MYLIST
(This is a test.)
8+(FOR X ON MYLIST AS Y ON (CDR X) DO
      (PRINT X) (PRINT Y))

UNBOUND ATOM
X

9+

```

**Figure 26.10.** The values of the iterative variables are set in parallel. If they depend on each other, an error can result.

This is described by saying that the local variables are set in parallel.

---

## 26.5 Conditional Iteration

---

**WHEN** can be used inside of an iterative expression when you wish only certain values to be used in the body of the loop. In the example (See Figure 26.11.), you are asking that you receive the values of X when Y is a list.

```

Interlisp-D Executive Window
NIL
59+(SETQ MYLIST '(A (B) C (D) E (F) G))
(A (B)
  C
  (D)
  E
  (F)
  G)
60+(FOR X FROM 1 TO 5 AS Y IN MYLIST
      WHEN (LISTP Y) COLLECT (X * X))
(4 16)
61+

```

**Figure 26.11.** Conditional Looping. The body of the loop is evaluated only when the value of Y is a list.

## 26.6 More Iteration

There are many other variations on iterative statements. They can all be found in the *Interlisp-D Reference Manual*, Volume 1, Chapter 9. An iterative keyword such as **FOR**, **BIND**, **WHILE**, or **REPEATUNTIL** must be used as the car of the iterative statement. Other than this restriction, iterative statements are flexible, other iterative keywords can be put in many different places in the statement. It is bad style, however, to take too much advantage of this flexibility. Here are some guidelines for good looping style: When using the keyword **WHEN** or **UNTIL**, place it before the **DO** or **COLLECT** keyword; when using the keyword **REPEATWHILE** or **REPEATUNTIL**, place it after the **DO** or **COLLECT** keyword.

The easiest way to learn to write iterative statements is to practice with the examples in this chapter.

**DO**

```

Interlisp-D Executive
NIL
18+(FOR CNT FROM 1 TO 3 DO
      (PRINT CNT)
      (PRINT " squared equals ")
      (PRINT (TIMES CNT CNT)))
1 squared equals 1
2 squared equals 4
3 squared equals 9
NIL
19+

```

**Figure 26.12.** Numerically controlled **FOR** loop. Prints squares of the number range

**Interlisp-D Executive**

```

NIL
23+(FOR ITEM IN
      '(1 (and in the end) 2 but 3)
      DO (IF (NUMBERP ITEM)
              THEN (PRIN1 ITEM)
                   (PRIN1 " is a number!")
                   (TERPRI))
          (IF (LISTP ITEM)
              THEN (PRINT ITEM)])
1 is a number!
(and in the end)
2 is a number!
3 is a number!
NIL
24+

```

Figure 26.13. List controlled **FOR** loop.**COLLECT****Interlisp-D Executive**

```

NIL
25+(FOR ITEM IN
      '(1 (and in the end) 2 but 3)
      COLLECT (IF (NUMBERP ITEM)
                  THEN
                    (PRIN1 ITEM)
                    (PRIN1 " is a number!")
                    (TERPRI))
              (IF (LISTP ITEM)
                  THEN
                    (PRINT ITEM)])
1 is a number!
(and in the end)
2 is a number!
3 is a number!
(NIL (and in the end)
  NIL NIL NIL)
26+

```

Figure 26.14. The same as last example except uses **COLLECT** instead of **DO**.**THEREIS****Interlisp-D Executive**

```

NIL
27+(FOR ITEM IN
      '(There is a 3 in this list.)
      THEREIS (IF (NUMBERP ITEM)
                  THEN
                    (PRINT "I found it!")))
"I found it"
3
28+

```

Figure 26.15. This loop will stop when the body evaluates to a *non-NIL* value.**AS**

```

Interlisp-D Executive
NIL
29+(FOR ITEM1 IN '(1 2 3)
      AS ITEM2 in
        '(is as was shall-be)
      AS ITEMS in '(A B C D E)
      COLLECT (LIST ITEM1 ITEM2 ITEMS))
((1 is A)
 (2 as B)
 (3 was C))
30+

```

**Figure 26.16.** Collects groupings of elements at the same position within each list.

### WHEN

```

Interlisp-D Executive
NIL
34+(FOR ITEM IN '(1 and 2 and 3)
      COLLECT (PRINT ITEM)
      WHEN (NUMBERP ITEM))
1
2
3
(1 2 3)
35+

```

**Figure 26.17.** Conditional in conjunction with a **FOR** loop. Prints and collects the numbers in a list.

### UNTIL

```

Interlisp-D Executive
NIL
38+(FOR ITEM IN
      '(There is a 3 in this list.)
      UNTIL (NUMBERP ITEM)
      COLLECT (PRINT ITEM))
There
is
a
(There is a)
39+

```

**Figure 26.18.** Conditional in conjunction with a **FOR** loop. Prints and collects the numbers in a list.

### BIND

```
Interlisp-D Executive Window
NIL
66-(FOR ITEM IN '(A B C D E)
      BIND (CNT + 0)
      DO (PRIN1 (LIST "Item"
                     (SETQ CNT (ADD1 CNT))
                     "is" (CAR ITEM)))
          (TERPRI))
(Item 1 is NIL)
(Item 2 is NIL)
(Item 3 is NIL)
(Item 4 is NIL)
(Item 5 is NIL)
NIL
67+
```

Figure 26.19. Independent use of conditional. Also illustrates the use of **BIND**. Prints the numerical positions and elements of a list.

[This page intentionally left blank]

---

## 27.1 Windows

---

Windows have two basic parts: an area on the screen containing a collection of pixels, and a property list. The window properties determine how the window looks, the menus that can be accessed from it, what should happen when the mouse is inside the window and a mouse button is pressed, and so on.

### 27.1.1 CREATEW

---

Some of the window's properties can be specified when a window is created with the function `CREATEW`. In particular, it is easy to specify the size and position of the window; its title; and the width of its borders.

*(CREATEW region title borderwidth)*

*Region* is a record, named `REGION`, with the fields `left`, `bottom`, `width`, and `height`. A region describes a rectangular area on the screen, the window's dimensions and position. The fields `left` and `bottom` refer to the position of the bottom left corner of the region on the screen. `Width` and `height` refer to the width and height of the region. The usable space inside the window will be smaller than the `width` and `height`, because some of the window's region is consumed by the title bar, and some is taken by the borders.

*Title* is a string that will be placed in the title bar of the window.

*Borderwidth* is the width of the border around the exterior of the window, in number of pixels.

For example, typing:

```
(SETQ MY.WINDOW (CREATEW  
  (CREATEREGION 100 150 300 200)  
  "THIS IS MY OWN WINDOW" )
```

produces a window with a default borderwidth. Note that you did not need to specify all the window's properties. (See Figure 27.1.)



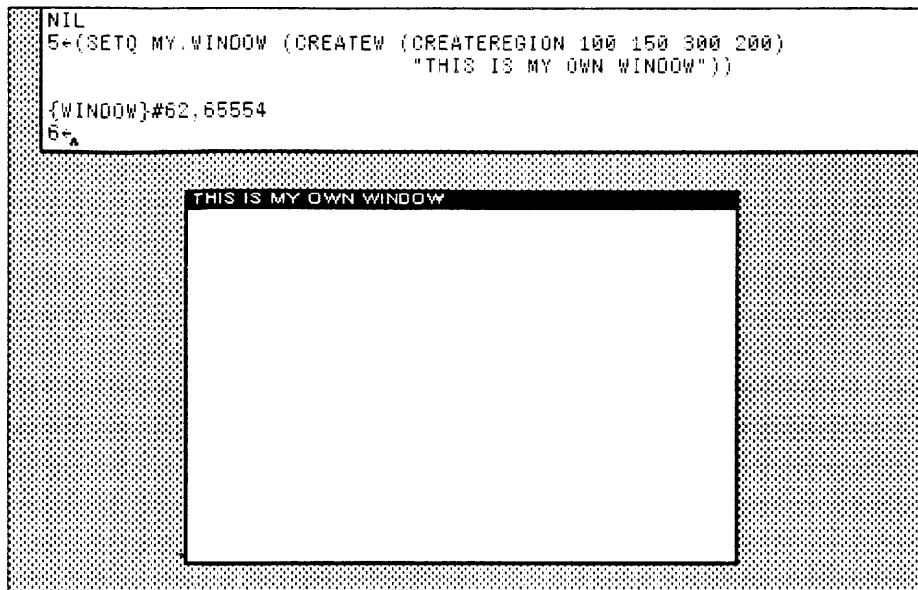


Figure 27.1. Creating a Window

In fact, if **(CREATEW)** is called without specifying a region, you will be prompted to sweep out a region for the window. (See Section 10.2, Page 10.2.)

## 27.1.2 WINDOWPROP

The function to access or add to any property of a window's property list is **WINDOWPROP**.

**(WINDOWPROP window property <value>)**

When you use **WINDOWPROP** with only two arguments - *window* and *property* - it returns the value of the window's property. When you use **WINDOWPROP** with all three arguments - *window*, *property* and *value* - it sets the value the window's property to the value you inserted for the third argument.

For example, consider the window, **MY.WINDOW**, created using **(CREATEW)**. **TITLE** and **REGION** are both properties. Type

**(WINDOWPROP MY.WINDOW 'TITLE)**

and the value of **MY.WINDOW**'s **TITLE** property is returned, "THIS IS MY OWN WINDOW". To change the title, use the **WINDOWPROP** function, and give it the window, the property title, and the new title of the window.

**(WINDOWPROP MY.WINDOW 'TITLE "MY FIRST WINDOW")**

automatically changes the title and automatically updates the window. Now the window looks like Figure 27.2.

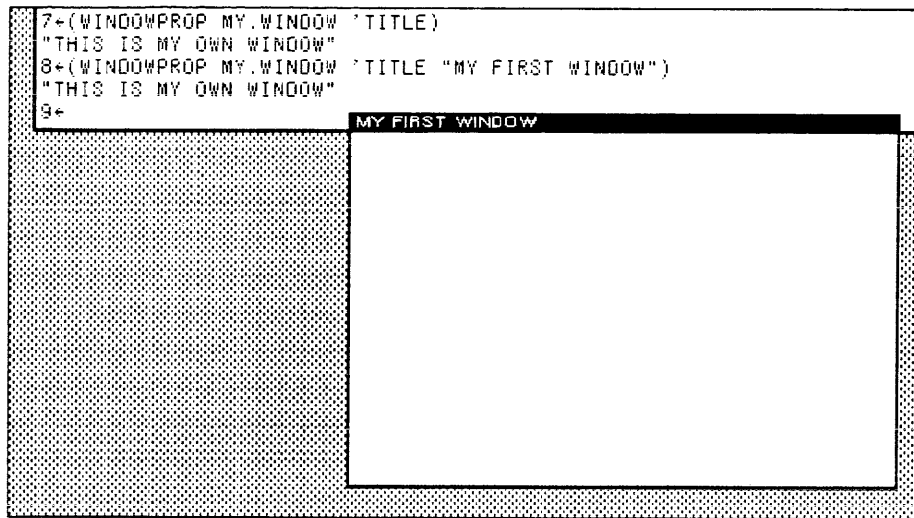


Figure 27.2. TITLE is a Window Property

Altering the region of the window, **MY.WINDOW**, is also be done with **WINDOWPROP**, in the same way you changed the title. (Note: changing either of the first two numbers of a region changes the position of the window on the screen. Changing either of the last two numbers changes the dimensions of the window itself.)

### 27.1.3 Getting windows to do things

Four basic window properties will be discussed here. They are **CURSORINFN**, **CURSOROUTFN**, **CURSORMOVEDFN**, and **BUTTONEVENTFN**.

A function can be stored as the value of the **CURSORINFN** property of a window. It is called when the mouse cursor is moved into that window.

Look at the following example:

- (1) First, create a window called **MY.WINDOW**. Type:

```
(SETQ MY.WINDOW
  (CREATEW
    (CREATEREGION 200 200 200 200)
    "THIS WINDOW WILL SCREAM!"))
```

This creates a window.

- (2) Now define the function **SCREAMER**. It will be stored on the property **CURSORINFN**. (Notice that this function has one argument, **WINDOWNAME**. All functions called from the property **CURSORINFN** are passed the window it was called from. So the value of **MY.WINDOW** is bound to **WINDOWNAME**. When it is called, **SCREAMER** simply rings bells.

```
(DEFINEQ (SCREAMER (WINDOWNAME)
  (RINGBELLS)
  (PROMPTPRINT "YAY - IT WORKS!")
  (RINGBELLS)))
```

- (3) Now, alter that window's **CURSORINFN** property, so that the system calls the function **SCREAMER** at the appropriate time. Type:

```
(WINDOWPROP MY.WINDOW 'CURSORINFN
(FUNCTION SCREAMER))
```

- (4) After this, when you move the mouse cursor into MY.WINDOW, the CURSORINFN property's function is called, and it rings bells twice.

CURSORINFN is one of the many window properties that come with each window - just as REGION and TITLE did. Other properties include:

CURSOROUTFN	The function that is the value of this property is executed when the cursor is moved out of a window;
CURSORMOVEDFN	the function that is the value of this property is executed when the cursor is moved while it is inside the window;
BUTTONEVENTFN	the function that is the value of this property is executed when either the left or middle mouse buttons are pressed (or released).

Figure 27.3 shows MY.WINDOW's properties. Notice that the CURSORINFN has the function SCREAMER stored in it. The properties were shown in this window using the function INSPECT. INSPECT is covered in Chapter 32.

Property	Value
SCREEN	NIL
WINDOWENTRYFN	GIVE.TTY.PROCESS
PROCESS	NIL
WORDER	4
NEWREGIONFN	NIL
WTITLE	"THIS WINDOW WILL SCREAM!"
MOVEFN	NIL
CLOSEFN	NIL
HORIZSCROLLWINDOW	NIL
VERTSCROLLWINDOW	NIL
SCROLLFN	NIL
HORIZSCROLLREG	NIL
VERTSCROLLREG	NIL
USERDATA	NIL
EXTENT	NIL
RESHAPEFN	NIL
REPAINTFN	NIL
CURSORMOVEDFN	NIL
CURSOROUTFN	NIL
CURSORINFN	SCREAMER
RIGHTBUTTONFN	NIL
BUTTONEVENTFN	TOTOPW
REG	(200 200 200 200)
SAVE	{BITMAP}#53,140526
NEXTW	{WINDOW}#55,171470
DSP	{STREAM}#55,114404

Figure 27.3. Inspecting MY.WINDOW for Mouse-Related Window Properties

You can define functions for the values of the properties CURSOROUTFN and CURSORMOVEDFN in much the same way as you did for CURSORINFN. The function that is the value of the property BUTTONEVENTFN, however, can be specialized to respond in different ways, depending on which mouse button is pressed. This is explained in the next section.

### 27.1.3.1 BUTTONEVENTFN

BUTTONEVENTFN is another property of a window. The function that is stored as the value of this property is called when the mouse is inside the window, and a mouse button is pressed. As an example of how to use it, type:

```
(WINDOWPROP MY.WINDOW 'BUTTONEVENTFN
(FUNCTION SCREAMER))
```

When the mouse cursor is moved into the window, bells will ring because of the `CURSORINFN`, but it will also ring bells when either the left or middle mouse button is pressed. Notice that the right mouse button functions as it usually does, with the window manipulation menu. If only the left button should evoke the function `SCREAMER`, then the function can be written to do just this, using the function `MOUSESTATE`, and a form that only `MOUSESTATE` understands, `ONLY`. For example:

```
(DEFINEQ
(SCREAMER2 (WINDOWNAME)
 (if (MOUSESTATE (ONLY LEFT))
     then (RINGBELLS))))
```

In addition to `(ONLY LEFT)`, `MOUSESTATE` can also be passed `(ONLY MIDDLE)`, `(ONLY RIGHT)` or combinations of these (e.g. `(OR (ONLY LEFT) (ONLY MIDDLE))`). You do not need to use `ONLY` with `MOUSESTATE` for every application. `ONLY` means that that button is pressed and no other.

If you do write a function using `(ONLY RIGHT)`, be sure that your function also checks position of the mouse cursor. Even if you want your function to be executed when the mouse cursor is inside the window and the right button is pressed, there is a convention that the function `DOWINDOWCOM` should be executed when the mouse cursor is in the title bar or the border of the window and the right mouse button is pressed. Please program your windows using this tradition! For more information, please see the *Interlisp-D Reference Manual*, Volume 3, Chapter 28, Pages 7 and 28.

Please refer to the *Interlisp Reference Manual*, Volume 3, Chapter 28, for more detail and other important functions.

#### 27.1.4 Looking at a window's properties

`INSPECT` is a function that displays a list of the properties of a window, and their values. Figure 27.3 shows the `INSPECT` function run with `MY.WINDOW`. Note the properties introduced in `CREATEW`: `WBORDER` is the window's border, `REG` is the region, and `WTITLE` is the window's title.

## 27.2 Regions

A region is a record, with the fields `LEFT`, `BOTTOM`, `WIDTH`, and `HEIGHT`. `LEFT` and `BOTTOM` refer to where the bottom left hand corner of the region is positioned on the screen. `WIDTH` and `HEIGHT` refer to the width and height of the region.

`CREATEREGION` creates an instance of a record of type `REGION`.  
Type:

```
(SETQ MY.REGION (CREATEREGION 15 100 200 450))
```

to create a record of type REGION that denotes a rectangle 200 pixels high, and 450 pixels wide, whose bottom left corner is at position (15, 100). This record instance can be passed to any function that requires a region as an argument, such as **CREATEW**, above.

While Interlisp-D provides a number of menus of its own (see Section 7.1, Page 7.2), this section addresses the menus you wish to create. You will learn how to create a menu, display a menu, and define functions that make your menu useful.

Menus are instances of records (see Chapter 24). There are 27 fields that determine the composition of every menu. Because Interlisp-D provides default values for most of these descriptive fields, you need to familiarize yourself with only a few that we describe in this section.

Two of these fields, the **TITLE** of your menu, and the **ITEMS** you wish it to contain, can be typed into the Interlisp-D Executive window as shown below:

```

Interlisp-D Executive Window
NIL
33+(SETQ MY.MENU (CREATE MENU
  TITLE + "PLEASE CHOOSE ONE OF THE
  ITEMS"
  ITEMS + '(QUIT NEXT-QUESTION
            NEXT-TOPIC SEE-TOPICS)))
{MENU}#54,143540
34+

```

Figure 28.1. Creating a menu

Note that creating a menu does not display it. **MY.MENU** is set to an instance of a menu record that specifies how the menu will look, but the menu is not displayed.

## 28.1 Displaying Menus

Typing either the **MENU** or **ADDMENU** functions will display your menu on the screen. **MENU** implements pop-up menus, like the Background Menu or the Window Menu. **ADDMENU** puts menus into a semi-permanent window on the screen, and lets you select items from it.

**(MENU MENU POSITION)** pops-up a menu at a particular position on the screen.

Type:

```
(MENU MY.MENU NIL)
```

to position the menu at the end of the mouse cursor. Note that the **POSITION** argument is **NIL**. In order to go on, you must either choose an item, or move outside the menu window and

press a mouse button. When you do either, the menu will disappear. If you choose an item, then want to choose another, the menu must be redisplayed.

**(ADDMENU menu window position)** positions a permanent menu on the screen, or in an existing window.

Type:

**(ADDMENU MY.MENU)**

to display the menu as shown in Figure 28.2. This menu will remain active, (will stay on the screen) without stopping all the other processes. Because **ADDMENU** can display a menu without stopping all other processes, it is very popular in users programs.

If *window* is specified, the *menu* is displayed in that window. If *window* is not specified, a window the correct size for the menu is created, and the menu is displayed in that window.

If *position* is not specified, the menu appears at the current position of the mouse cursor.

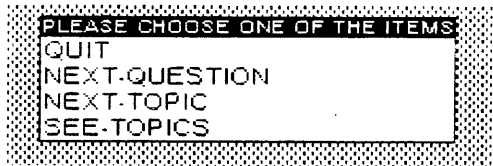


Figure 28.2. A Simple Menu, displayed with **ADDMENU**.

## 28.2 Getting Menus to DO Stuff

One way to make a menu do things is to specify more about the menu items. Instead of items simply being the strings or atoms that will appear in the menu, items can be lists, each list with three elements. (See Figure 28.3.) The first element of each list is what will appear in the menu; the second expression is what is evaluated, and the results of the evaluation returned, when the item is selected; and the third expression is the expression that should be printed in the Prompt window when a mouse button is held down while the mouse is pointing to that menu item. This third item should be thought of as help text for the user. If the third element of the list is **NIL**, the system responds with "Will select this item when you release the button".

```

Top level -- Connected to {DSK}<LISPPFILES>PRIMER>IM>
NIL
17+(SETQ MY.MENU2 (CREATE MENU
  TITLE + "PLEASE CHOOSE ONE OF THE ITEMS"
  ITEMS + '((QUIT
    (PRINT "STOPPED")
    "CHOOSE THIS TO STOP")
    (NEXT-QUESTION
    (PRINT "HERE IS THE NEXT QUESTION...")
    "CHOOSE THIS TO MOVE ON TO THE NEXT QUESTION")
    (NEXT-TOPIC
    (PRINT "HERE IS THE NEXT TOPIC...")
    "CHOOSE THIS TO MOVE ON TO THE NEXT SUBJECT")
    (SEE-TOPICS
    (PRINT "THE FOLLOWING HAVE NOT BEEN LEARNED")
    "CHOOSE THIS TO SEE THE TOPICS NOT YET LEARNED"))))
{MENU}#55,75054
18+(ADDMENU MY.MENU2)
{WINDOW}#54,175320
19+
PLEASE CHOOSE ONE OF THE ITEMS
QUIT
NEXT-QUESTION
NEXT-TOPIC
SEE-TOPICS

```

**Figure 28.3.** Creating a menu that will do things, then displaying it with the function **ADDMENU**

Now when an item is selected from **MY.MENU2**, something will happen. When a mouse button is held down, the expression typed as the third element in the item's specification will be printed in the Prompt window. (See Figure 28.4.)

```

PLEASE CHOOSE ONE OF THE ITEMS
QUIT
NEXT-QUESTION
NEXT-TOPIC
SEE-TOPICS
Prompt Window
HERE IS THE NEXT QUESTION

```

**Figure 28.4.** Mouse Button Held Down While Mouse Cursor Selects **NEXT-QUESTION**

When the mouse button is released (i.e. the item is selected) the expression that was typed as the second element of the item's specification will be run. (See Figure 28.5.)

```

PLEASE CHOOSE ONE OF THE ITEMS
QUIT
NEXT-QUESTION
NEXT-TOPIC
SEE-TOPICS
Prompt Window
TTY window for MOUSE
"HERE IS THE NEXT QUESTION..."

```

**Figure 28.5.** **NEXT-QUESTION** Selected



## 28.2.1 The WHENHELDFN and WHENSELECTEDFN fields of a menu

Another way to get a menu to do things is to define functions, and make them the values of the menu's WHENHELDFN and WHENSELECTEDFN fields. As the value of the WHENHELDFN field of a menu, the function you defined will be executed when you press and hold a mouse button inside the menu. As the value of the WHENSELECTEDFN field of a menu, the function you defined will be executed when you choose a menu item. This example has the same functionality as the previous example, where each menu item was entered as a list of three items.

As an example, type in these two functions so that they can be executed when the menu is created and displayed:

```
(DEFINEQ (MY.MENU3.WHENHELD (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
  (SELECTQ ITEM.SELECTED
    (QUIT (PROMPTPRINT "CHOOSE THIS TO STOP"))
    (NEXT-QUESTION (PROMPTPRINT "CHOOSE THIS TO BE ASKED THE NEXT QUESTION"))
    (NEXT-TOPIC (PROMPTPRINT "CHOOSE THIS TO MOVE ON TO THE NEXT SUBJECT"))
    (SEE-TOPICS (PROMPTPRINT "CHOOSE THIS TO SEE THE TOPICS NOT YET LEARNED"))
    (ERROR (PROMPTPRINT "NO MATCH FOUND")))))
(DEFINEQ (MY.MENU3.WHENSELECTED (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
  (SELECTQ ITEM.SELECTED
    (QUIT (PRINT "STOPPED"))
    (NEXT-QUESTION (PRINT "HERE IS THE NEXT QUESTION..."))
    (NEXT-TOPIC (PRINT "HERE IS THE NEXT TOPIC..."))
    (SEE-TOPICS (PRINT "THE FOLLOWING HAVE NOT BEEN LEARNED..." ))
    (ERROR (PROMPTPRINT "NO MATCH FOUND")))))
```

Now, to create the menu, type:

```
(SETQ MY.MENU3 (CREATE MENU
  TITLE ← "PLEASE CHOOSE ONE OF THE ITEMS"
  ITEMS ← '(QUIT NEXT-QUESTION NEXT-TOPIC SEE-TOPICS)
  WHENHELDFN ← (FUNCTION MY.MENU3.WHENHELD)
  WHENSELECTEDFN ← (FUNCTION MY.MENU3.WHENSELECTED)))
```

Type

```
(ADDMENU MY.MENU3)
```

to see your menu work.

Now, due to executing the WHENHELDFN function, holding down any mouse button while pointing to a menu item will display an explanation of the item in the prompt window. The screen will once again look like Figure 28.4 when the mouse button is held when the mouse cursor is pointing to the item NEXT-TOPIC.

Now due to executing the WHENSELECTEDFN function, releasing the mouse button to select an item will cause the proper actions for that item to be taken. The screen will once again look like Figure 28.5 when the item NEXT-TOPIC is selected.

The crucial thing to note is that the functions you defined for WHENHELDFN and WHENSELECTEDFN are automatically given the following arguments:

- (1) the item that was selected, *ITEM.SELECTED*;
- (2) the menu it was selected from, *MENU.FROM*;
- (3) and the mouse button that was pressed *BUTTON.PRESSED*.

Note: these functions, **MY.MENU3.WHENHELD** and **MY.MENU3.WHENSELECTED**, were quoted using **FUNCTION** instead of **QUOTE** both for program readability and so that the

compiler can produce faster code when the program is compiled. It is good style to quote functions in Interlisp by using the function **FUNCTION** instead of **QUOTE**.

## 28.3 Looking at a menu's fields

**INSPECT** is a function that displays a list of the fields of a menu, and their values. The Figure 28.6 shows the various fields of **MY.MENU3** when the function (**INSPECT MY.MENU**) was called. Notice the values that were assigned by the examples, and all the defaults.

```

98+ (INSPECT MY.MENU3)
(WINDOW)#61,34554
99+ (MENU)#62,52204 Inspector
ITEMWIDTH      195
ITEMHEIGHT     12
IMAGEWIDTH     197
IMAGEHEIGHT    59
MENUREGIONLEFT 0
MENUREGIONBOTTOM 0
IMAGE          (WINDOW)#61,165150
SAVEIMAGE      NIL
ITEMS          (QUIT NEXT-QUESTION NEXT-TOPIC SEE-T
MENUROWS       4
MENUCOLUMNS   1
MENUGRID       (1 1 195 12)
CENTERFLG      NIL
CHANGEOFFSETLG NIL
MENUFONT       (FONTDESCRIPTOR)#70,171260
TITLE          "PLEASE CHOOSE ONE OF THE ITEMS"
MENUOFFSET     (0 . 0)
WHENSELECTEDFN MY.MENU3 WHENSELECTED
MENUBORDERSIZE 0
MENUOUTLINESIZE 1
WHENHELDFN     MY.MENU3 WHENHELD
MENUPOSITION   NIL
WHENUNHELOFN   CLRPPROMPT
MENUUSERDATA   NIL
MENUTITLEFONT  NIL
SUBITEMFN      NIL
MENUFEEDBACKFLG NIL
SHADEDITEMS    NIL

```

Figure 28.6. The Fields of MY.MENU3

[This page intentionally left blank]

A bitmap is a rectangular array of dots. The dots are called pixels (for *picture elements*). Each dot, or pixel, is represented by a single bit. When a pixel or bit is turned on (i.e. that bit set to 1), a black dot is inserted into a bitmap. If you have a bitmap of a floppy on your screen, (Figure Figure 29.1), then all of the bits in the area that make up the floppy are turned on, and the surrounding bits are turned off.

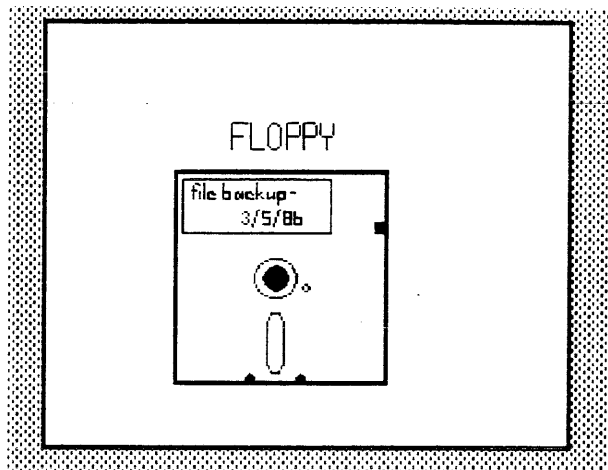


Figure 29.1. Bitmap of a Floppy

**BITMAPCREATE** creates a bitmap, even though it can't be seen.

**(BITMAPCREATE *width height*)**

If the *width* and *height* are not supplied, the system will prompt you for them.

**EDITBM** edits the bitmap. The syntax of the function is:

**(EDITBM *bitmapname*)**

Try the following to produce the results in Figure 29.4:

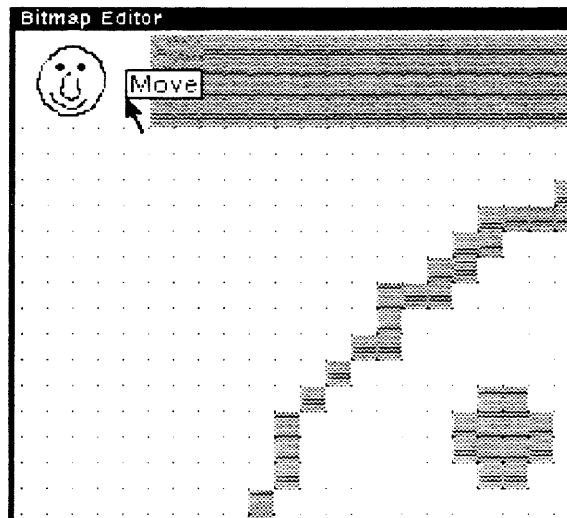
```
(SETQ MY.BITMAP (BITMAPCREATE 60 40))
(EDITBM MY.BITMAP)
```

To draw In the bitmap, move the mouse into the gridded section of the bitmap editor, and press and hold the left mouse button. Move the mouse around to turn on the bits represented by the spaces in the grid. Notice that each space in the grid represents one pixel on the bitmap

To erase Move the mouse into the gridded section of the bitmap editor, and press and hold the center mouse button. Move the mouse around to turn off the bits represented by the spaces in the gridded section of the bitmap editor.

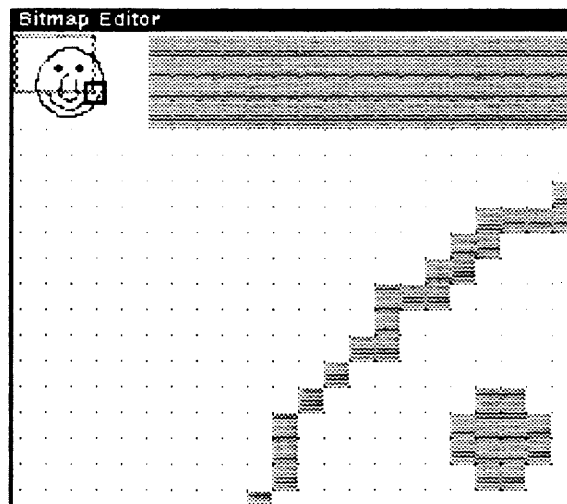
To work on a different section Point with the mouse cursor to the picture of the actual bitmap (the upper left corner of the bitmap editor). Press and hold the

left mouse button. A menu with the single item, **Move** will appear. (See Figure 29.2.) Choose this item.



**Figure 29.2.** Move the mouse cursor to the picture of the bitmap. Press and hold the left mouse button, and the Move menu will appear.

You will be asked to position a ghost window over the bitmap. This ghost window represents the portion of the bitmap that you are currently editing. Place it over the section of the bitmap that you wish to edit. (See Figure 29.3.)



**Figure 29.3.** After you choose move, you will be asked to position a ghost window like this one. Position it by clicking the left mouse button when the ghost window is over the part of the picture of the bitmap you would like to edit.

To end the session

Bring the mouse cursor into the upper-right portion of the window (the grey area) and press the center button. Select **OK** from the menu to save your artwork.

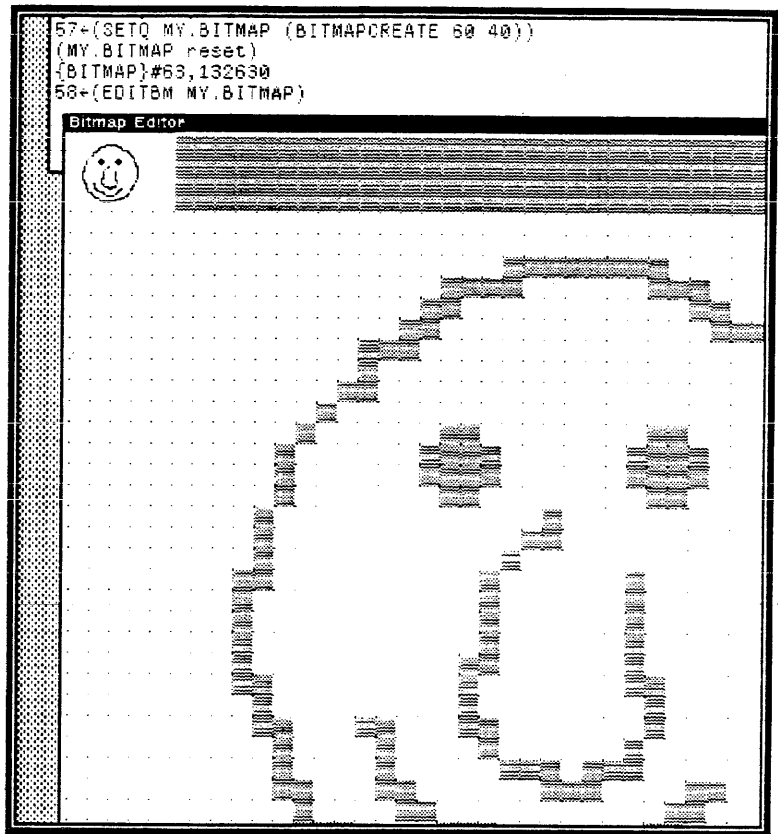


Figure 29.4. Editing a Bitmap

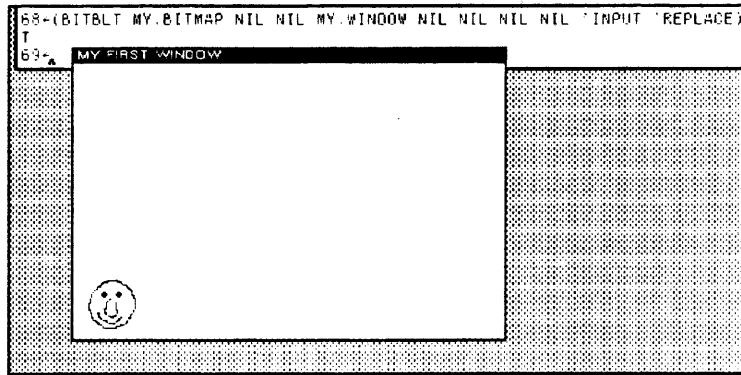
**BITBLT** is the primitive function for moving bits (or pixels) from one bitmap to another. It extracts bits from the source bitmap, and combines them in appropriate ways with those of the destination bitmap. The syntax of the function is:

```
(BITBLT sourcebitmap sourceleft sourcebottom
destinationbitmap destinationleft destinationbottom width
height sourcetype operation texture clippingregion)
```

Here's how it's done - using `MY.BITMAP` as the `sourcebitmap` and `MY.WINDOW` as the `destinationbitmap`:

```
(BITBLT MY.BITMAP NIL NIL
MY.WINDOW NIL NIL NIL NIL 'INPUT 'REPLACE)
```

Note that the destination bitmap can be, and usually is, a window. Actually, it is the bitmap of a window, but the system handles that detail for you. Because of the `NILs` (meaning "use the default"), `MY.BITMAP` will be `BITBLT`'d into the lower right hand corner of `MY.WINDOW`. (See Figure 29.5.)



**Figure 29.5.** BITBLTing a Bitmap onto a Window

Here is what each of the BITBLT arguments to the function mean:

sourcebitmap	the bitmap to be moved into the destinationbitmap
sourceleft	a number, starting at 0 for the left edge of the sourcebitmap, that tells <b>BITBLT</b> where to start moving pixels from the sourcebitmap. For example, if the leftmost 10 pixels of sourcebitmap were not to be moved, sourceleft should be 10. The default value is 0.
sourcebottom	a number, starting at 0 for the bottom edge of the sourcebitmap, that tells <b>BITBLT</b> where to start moving pixels from the sourcebitmap. For example, if the bottom 10 rows of pixels of sourcebitmap were not to be moved, sourcebottom should be 10. The default value is 0.
destinationbitmap	the bitmap that will receive the sourcebitmap. This is often a window (actually the bitmap of a window, but Interlisp-D takes care of that for you).
destinationleft	a number, starting at 0 for the left edge of the destinationbitmap, that tells <b>BITBLT</b> where to start placing pixels from the sourcebitmap. For example, to place the sourcebitmap 10 pixels in from the left, destinationleft should be 10. The default value is 0.
destinationbottom	a number, starting at 0 for the bottom edge of the destinationbitmap, that tells <b>BITBLT</b> where to start placing pixels from the sourcebitmap. For example, to place the sourcebitmap 10 pixels up from the bottom, destinationbottom should be 10. The default value is 0.
width	how many pixels in each row of sourcebitmap should be moved. The same amount of space is used in destinationbitmap to receive the sourcebitmap. If this argument is NIL, it defaults to the number of pixels from sourceleft to the end of the row of sourcebitmap.
height	how many rows of pixels of sourcebitmap should be moved. The same amount of space is used in destinationbitmap to receive the sourcebitmap. If this argument is NIL, it defaults to the number of rows from sourcebottom to the top of the sourcebitmap.
sourcetype	refers to one of three ways to convert the sourcebitmap for writing. For now, just use <b>'INPUT</b> .

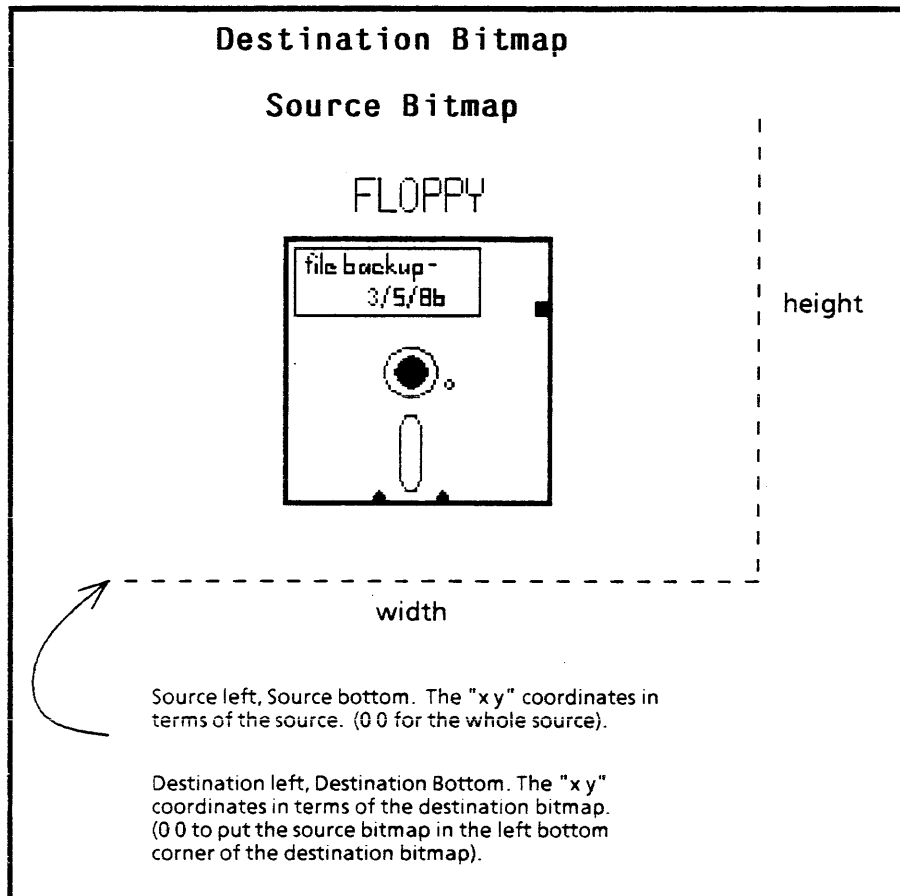
operation refers to how the sourcebitmap gets BITBLT'd on to the destinationbitmap. 'REPLACE will BLT the exact sourcebitmap. Other operations allow you to AND, OR or XOR the bits from the sourcebitmap onto the bits on the destinationbitmap.

texture Just use **NIL** for now.

clippingregion just use **NIL** for now.

For more information on these operations, see the *Interlisp-D Reference Manual*, Volume 3, Chapter 27, Page 14.

Sourcebitmap, sourceleft, sourcebottom, destinationbitmap, destinationleft, destinationbottom, width and height are shown in Figure 29.6.



**Figure 29.6.** BITBLT'ed Bitmap of a Floppy



[This page intentionally left blank]

A displaystream is a generalized "place to display". They determine exactly what is displayed where. One example of a displaystream is a window. Windows are the only displaystreams that will be used in this chapter. If you want to draw on a bitmap that is not a window, other than with **BITBLT**, or want to use other types of displaystreams, please refer to the *Interlisp-D Reference Manual*, Volume 3, Chapter 27.

This chapter explains functions for drawing on displaystreams: **DRAWLINE**, **DRAWTO**, **DRAWCIRCLE.**, and **FILLCIRCLE**. In addition, functions for locating and changing your current position in the displaystream are covered: **DSPXPOSITION**, **DSPYPOSITION**, and **MOVETO**.

---

## 30.1 Drawing on a Displaystream

---

Examples will show you how the functions for drawing on a display stream work. First, create a window. Windows are displaystreams, and the one you create will be used for the examples in this chapter. Type:

```
(SETQ EXAMPLE.WINDOW (CREATEW))
```

### 30.1.1 DRAWLINE

---

**DRAWLINE** draws a line in a displaystream. For example, type:

```
(DRAWLINE 10 15 100 150 5 'INVERT EXAMPLE.WINDOW)
```

The results should look like this:

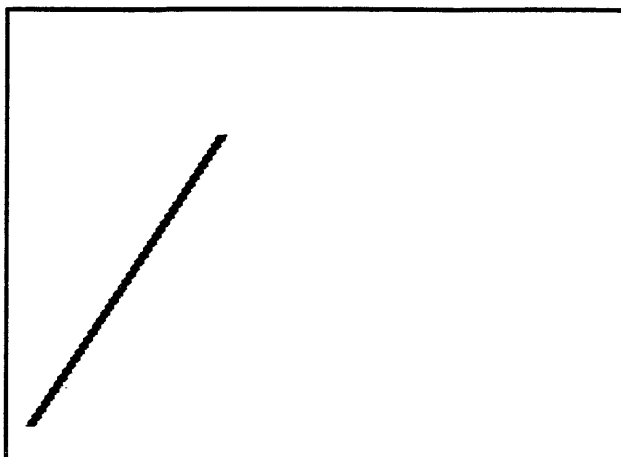


Figure 30.1. The line drawn onto the displaystream, EXAMPLE.WINDOW.

The syntax of **DRAWLINE** is

**(DRAWLINE x1 y1 x2 y2 width operation stream |)**

The coordinates of the left bottom corner of the displaystream are 0 0.

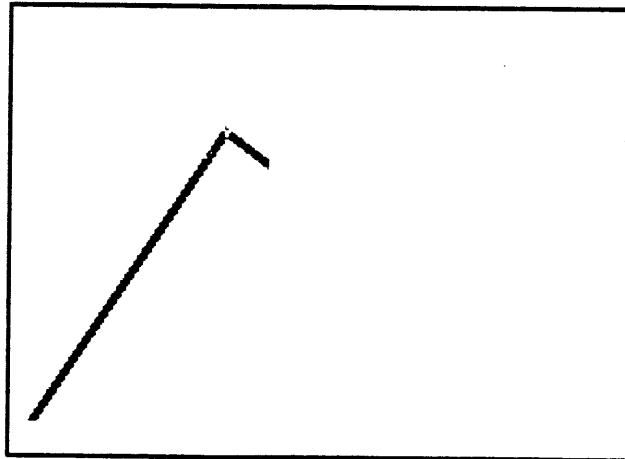
- x1 and y1** are the x and y coordinates of the beginning of the line;
- x2 and y2** are the ending coordinates of the line;
- width** is the width of the line, in pixels
- operation** is the way the line is to be drawn. **INVERT** causes the line to invert the bits that are already in the displaystream. Drawing a line the second time using **INVERT** erases the line. For other operations, see the *Interlisp-D Reference Manual*, Volume III, Page 27.15.
- stream** is the displaystream. In this case, you used a window.

### 30.1.2 DRAWTO

**DRAWTO** draws a line that begins at your current position in the displaystream. For example, type:

**(DRAWTO 120 135 5 'INVERT EXAMPLE.WINDOW)**

The results should look like this:



**Figure 30.2.** Another line drawn onto the displaystream, **EXAMPLE.WINDOW**.

The syntax of **DRAWTO** is

**(DRAWTO x y width operation stream |)**

The line begins at the current position in the displaystream.

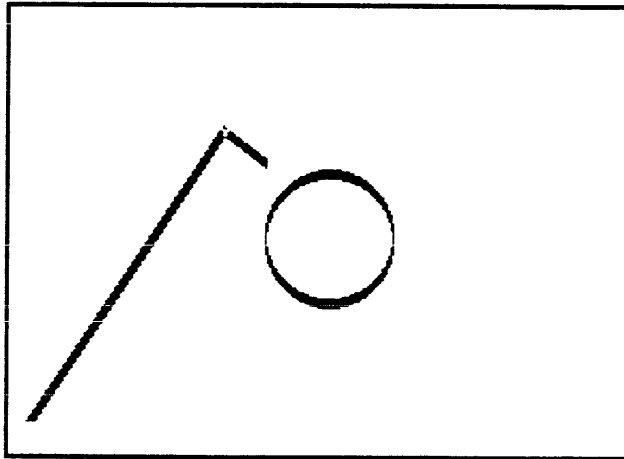
- x** is the x coordinate of the end of the line;
- y** is the y coordinate of the end of the line;
- width** is the width of the line
- operation** is the way the line is to be drawn. **INVERT** causes the line to invert the bits that are already in the displaystream. Drawing a line the second time using **INVERT** erases the line. For other operations, see the *Interlisp-D Reference Manual*, Volume III, Page 27.15.
- stream** is the displaystream. In this case, you used a window.

### 30.1.3 DRAWCIRCLE

**DRAWCIRCLE** draws a circle on a displaystream. To use it, type:

```
(DRAWCIRCLE 150 100 30 '(VERTICAL 5) NIL EXAMPLE.WINDOW)
```

Now your window, **EXAMPLE.WINDOW**, should look like this:



**Figure 30.3.** The circle drawn onto the displaystream, **EXAMPLE.WINDOW**.

The syntax of **DRAWCIRCLE** is

```
(DRAWCIRCLE centerx centery radius brush dashing stream)
```

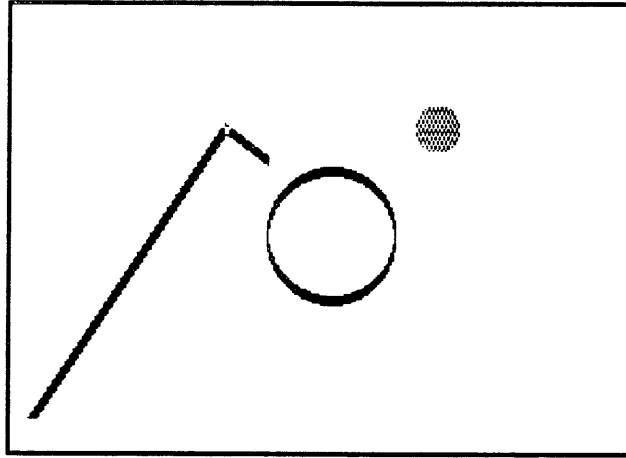
- centerx* is the x coordinate of the center of the circle
- centery* is the y coordinate of the center of the circle
- radius* is the radius of the circle in pixels
- brush* is a list. The first item of the list is the shape of the brush. Some of your options include **ROUND**, **SQUARE**, and **VERTICAL**. The second item of that list is the width of the brush in pixels.
- dashing* is a list of positive integers. The brush is "on" for the number of units indicated by the first element of the list, "off" for the number of units indicated by the second element of the list. The third element specifies how long it will be on again, and so forth. The sequence is repeated until the circle has been drawn.
- stream* is the displaystream. In this case, you used a window.

#### 30.1.3.1 FILLCIRCLE

**FILLCIRCLE** draws a filled circle on a displaystream. To use it, type:

```
(FILLCIRCLE 200 150 10 GRAYSHADE EXAMPLE.WINDOW)
```

**EXAMPLE.WINDOW** now looks like this:



**Figure 30.4.** A filled circle drawn onto the displaystream, EXAMPLE.WINDOW.

The syntax of **FILLCIRCLE** is

(**FILLCIRCLE** *centerx centery radius texture stream*)

- centerx* is the x coordinate of the center of the circle
- centery* is the y coordinate of the center of the circle
- radius* is the radius of the circle in pixels
- texture* is the shade that will be used to fill in the circle. Interlisp-D provides you with three shades, **WHITESHAD**, **BLACKSHAD**, and **GRAYSHAD**. You can also create your own shades. For more information on how to do this, see the *Interlisp-D Reference Manual*, Volume III, Page 27.7.
- stream* is the displaystream. In this case, you used a window.

There are many other functions for drawing on a displaystream. Please refer to the *Interlisp-D Reference Manual*, Volume III, Chapter 27.

Text can also be placed into displaystreams. To do this, use printing functions such as **PRIN1** and **PRIN2**, but supply the name of the displaystream as the "file" to print to. To place the text in the proper position in the displaystream, see Section 30.2, Page 30.4.

---

## 30.2 Locating and Changing Your Position in a Displaystream

---

There are functions provided to locate, and to change your current position in a displaystream. This can help you place text, and other images where you want them in a displaystream. This primer will only discuss three of these. There are others, and they can be found in the *Interlisp-D Reference Manual*, Volume III, Chapter 27.

### 30.2.1 DSPXPOSITION

---

**DSPXPOSITION** is a function that will either change the current x position in a displaystream, or simply report it. To have the function report the current x position in `EXAMPLE.WINDOW`, type:

**(DSPXPOSITION NIL EXAMPLE.WINDOW)**

**DSPXPOSITION** expects two arguments. The first is the new x position. If this argument is `NIL`, the current position is not changed, merely reported. The second argument is the displaystream.

### 30.2.2 DSPYPOSITION

---

**DSPYPOSITION** is an analogous function, but it changes or reports the current y position in a displaystream. As with **DSPXPOSITION**, if the first argument is a number, the current y position will be changed to that position. If it is `NIL`, the current position is simply reported. To have the function report the current y position in `EXAMPLE.WINDOW`, type:

**(DSPYPOSITION NIL EXAMPLE.WINDOW)**

### 30.2.3 MOVETO

---

The function **MOVETO** always changes your position in the displaystream. It expects three arguments:

**(MOVETO *x y stream*)**

*x* is the new x position in the display stream

*y* is the new y position in the display stream

*stream* is the display stream. The examples so far have used a window.

[This page intentionally left blank]

This chapter explains fonts and fontdescriptors, what they are and how to use them, so that you can use functions requiring fontdescriptors.

You have already been exposed to many fonts in Interlisp-D. For example, when you use the structure editor, DEdit, (See Section 11.3.), you noticed that the comments were printed in a smaller font than the code, and that CLISP words (See Section 13.1, Page 13.1.) were printed in a darker font than the other words in the function. These are only some of the fonts that are available in Interlisp-D.

In addition to the fonts that appear on your screen, Interlisp-D uses fonts for printers that are different than the ones used for the screen. The fonts used to print to the screen are called DISPLAYFONTS. The fonts used for printing are called INTERPRESSFONTS, or PRESSFONTS, depending on the type of printer.

---

## 31.1 What makes up a FONT?

---

Fonts are described by family, weight, slope, width, and size. This section discusses each of these, and describes how they affect the font you see on the screen.

Family is one way that fonts can differ. Here are some examples of how "family" affects the look of a font:

CLASSIC	This family makes the word "Able" look like this: Able
MODERN	This family makes the word "Able" look like this: Able
TERMINAL	This family makes the word "Able" look like this: Ab1e

Weight also determines the look of a font. Once again, "Able" will be used as an example, this time only with the Classic family. A font's weight can be:

BOLD	and look like this: <b>Able</b>
MEDIUM or REGULAR	and look like this: Able

The slope of a font is italic or regular. Using the Classic family font again, in a regular weight, the slope affects the font like this:

ITALIC	looks like this: <i>Able</i>
REGULAR	looks like this: Able



The width of a font is called its "expansion". It can be COMPRESSED, REGULAR, or EXPANDED.

Together, the weight, slope, and expansion of a font specifies the font's "face". Specifically, the face of a font is a three element list:

*(weight slope expansion)*

To make it easier to type, when a function requires a font face as an argument, it can be abbreviated with a three character atom. The first specifies the weight, the second the slope, and the third character the expansion. For example, some common font faces are abbreviated:

- MRR This is the usual face, MEDIUM, REGULAR, REGULAR;
- MIR makes an italic font. It stands for: MEDIUM, ITALIC, REGULAR;
- BRR makes a bold font. The abbreviation means: BOLD, REGULAR, REGULAR;
- BIR means that the font should be both bold and italic. BIR stands for BOLD, ITALIC, REGULAR.

The above examples are used so often, that there are also more mnemonic abbreviations for them. They can also be used to specify a font face for a function that requires a face as an argument. They are:

- STANDARD This is the usual face: MEDIUM, REGULAR, REGULAR. It was abbreviated above, MRR;
- ITALIC This was abbreviated above as MIR, and specifies an italic font;
- BOLD of course, makes a bold font. It was abbreviated above, BRR;
- BOLDITALIC means that the font should be both bold and italic: BOLD, ITALIC, REGULAR. It was abbreviated above, BIR.

A font also has a size. It is a positive integer that specifies the height of the font in printers points. A point is, on an 1108 screen, about 1/72 of an inch. On the screen of an 1186, a point is 1/80 of an inch. The size of the font used in this chapter is 10. For comparison, here is an example of a TERMINAL, MRR, size 12 font: Ab 1 e.

---

## 31.2 Fontdescriptors, and FONTCREATE

---

For Interlisp-D to use a font, it must have a fontdescriptor. A fontdescriptor is a datatype in Interlisp-D that holds all the information needed in order to use a particular font. When you print out a fontdescriptor, it looks like this:

```
{FONTDESCRIPTOR}#74,45540
```

Fontdescriptors are created by the function FONTCREATE. For example,

```
(FONTCREATE 'HELVETICA 12 'BOLD)
```

creates a fontdescriptor that, when used by other functions, prints in HELVETICA BOLD size 12. Interlisp-D functions that work with fonts expect a fontdescriptor produced with the **FONTCREATE** function.

The syntax of **FONTCREATE** is:

**(FONTCREATE family size face)**

Remember from the previous section, *face* is either a three element list, (weight slope expansion), a three character atom abbreviation, e.g. MRR, or one of the mnemonic abbreviations, e.g. STANDARD.

If **FONTCREATE** is asked to create a fontdescriptor that already exists, the existing fontdescriptor is simply returned.

### 31.3 Display Fonts - Their files, and how to find them

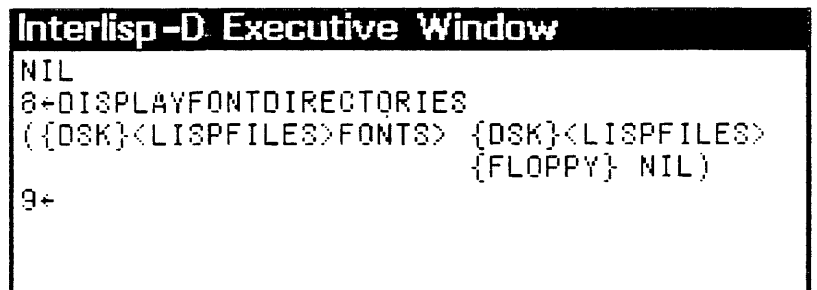
Display fonts require files that contain the bitmaps used to print each character on the screen. All of these files have the extension **.DISPLAYFONT**. The file name itself describes the font style and size that uses its bitmaps. For example:

**MODERN12.DISPLAYFONT**

contains bitmaps for the font family MODERN in size 12 points.

Initially, these files are on floppies. The files that are used most often should be copied onto a directory of your hard disk or fileserver. Usually, this directory is called **FONT**S.

Wherever you put your **.DISPLAYFONT** files, you should make this one of the values of the variable **DISPLAYFONTDIRECTORIES**. Its value is a list of directories to search for the bitmap files for display fonts. Usually, it contains the "FONT" directory where you copied the bitmap files, the device {FLOPPY}, and the current connected directory. The current connected directory is specified by the atom NIL. Here is an example value of **DISPLAYFONTDIRECTORIES**:



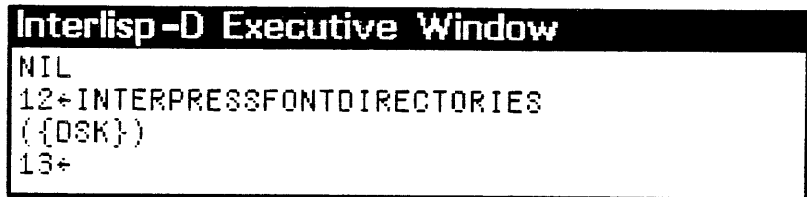
**Figure 31.1.** A value for the atom **DISPLAYFONTDIRECTORIES**. When looking for a **.DISPLAYFONT** file, the system will check the FONT directory on the hard disk, then the top level directory on the hard disk, then the floppy, then the current connected directory.

## 31.4 Interpress Fonts - Their files, and how to find them

Interpress is the format that is used by Xerox laser printers. These printers normally have a resolution that is much higher than that of the screen: 300 points per inch.

In order to format files appropriately for output on such a printer, Interlisp must know the actual size for each character that is to be printed. This is done through the use of width files that contain font width information for fonts in Interpress format. Initially, these files (with extension .WD) are on floppies. The files should be copied onto a directory of your hard disk or fileserver.

For Interpress fonts, you should make the location of these files one of the values of the variable **INTERPRESSFONTDIRECTORIES**. Its value is a list of directories to search for the font widths files for Interpress fonts. Here is an example value of **INTERPRESSFONTDIRECTORIES**:



```

Interlisp-D Executive Window
NIL
12+INTERPRESSFONTDIRECTORIES
({DSK})
13+

```

**Figure 31.2.** A value for the atom **INTERPRESSFONTDIRECTORIES**. When looking for a font widths file for an Interpress font, Interlisp-D will check the hard disk.

## 31.5 Functions for Using Fonts

### 31.5.1 FONTPROP - Looking at Font Properties

It is possible to see the properties of a fontdescriptor. This is done with the function **FONTPROP**. For the following examples, the fontdescriptor used will be the one returned by the function (**DEFAULTFONT 'DISPLAY**). In other words, the fontdescriptor examined will be the default display font for the system.

There are many properties of a font that might be useful for you. Some of these are:

- FAMILY** To see the family of a font descriptor, type:  
(**FONTPROP (DEFAULTFONT 'DISPLAY) 'FAMILY**)
- SIZE** As above, this is a positive integer that determines the height of the font in printer's points. As an example, the **SIZE** of the current default font is:

```

Interlisp-D Executive Window
NIL
61+(FONTPROP (DEFAULTFONT 'DISPLAY)
              'SIZE)
10
62+

```

Figure 31.3. The value of the font property **SIZE** of the default font

#### ASCENT

The value of this property is a positive integer, the maximum height of any character in the specified font from the baseline (bottom). The top of the tallest character in the font, then, will be at (BASELINE + ASCENT - 1). For example, the **ASCENT** of the default font is:

```

Interlisp-D Executive Window
NIL
64+(FONTPROP (DEFAULTFONT 'DISPLAY)
              'ASCENT)
9
65+

```

Figure 31.4. The value of the font property **ASCENT** of the default font

#### DESCENT

The **DESCENT** is an integer that specifies the maximum number of points that a character in the font descends below the baseline (e.g. letters such as "p" and "g" have tails that descend below the baseline.). The bottom of the lowest character in the font will be at (BASELINE - DESCENT). To see the **DESCENT** of the default font, type:

```
(FONTPROP (DEFAULTFONT 'DISPLAY) 'DESCENT)
```

#### HEIGHT

**HEIGHT** is equal to (DESCENT - ASCENT).

#### FACE

The value of this property is a list of the form, (*weight slope expansion*). These are the weight, slope, and expansion described above. You can see each one separately, also. Use the property that you are interested in, **WEIGHT**, **SLOPE**, or **EXPANSION**, instead of **FACE** as the second argument to **FONTPROP**.

For other font properties, see the *Interlisp-D Reference Manual*, Volume III, Pages 27.27 - 27.28.

## 31.5.2 STRINGWIDTH

It is often useful to see how much space is required to print an expression in a particular font. The function **STRINGWIDTH** does this. For example, type:

```
(STRINGWIDTH "Hi there!" (FONTCREATE 'GACHA 10 'STANDARD))
```

The number returned is how many left to right pixels would be needed if the string were printed in this font. (Note that this

doesn't just work for pixels on the screen, but for all kinds of streams. For more information about streams, see Chapter 30.) Compare the number returned from the example call with the number returned when you change GACHA to TIMESROMAN.

### 31.5.3 DSPFONT - Changing the Font in One Window

The function **DSPFONT** changes the font in a single window. As an example of its use, first create a window to write in. Type:

```
(SETQ MY.FONT.WINDOW (CREATEW))
```

in the Interlisp-D Executive window. Sweep out the window. To print something in the default font, type:

```
(PRINT 'HELLO MY.FONT.WINDOW)
```

in the Interlisp-D Executive window. Your window, MY.FONT.WINDOW, will look something like this:

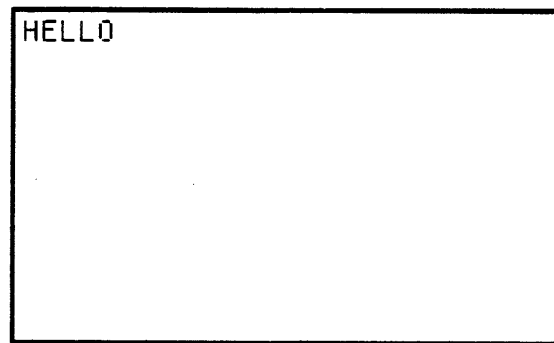


Figure 31.5. HELLO, printed with the default font in MY.FONT.WINDOW

Now change the font in the window. Type:

```
(DSPFONT (FONTCREATE 'HELVETICA 12 'BOLD) MY.FONT.WINDOW)
```

in the Interlisp-D Executive window. The arguments to **FONTCREATE** can be changed to create any desired font. Now retype the **PRINT** statement, and your window will look something like this:

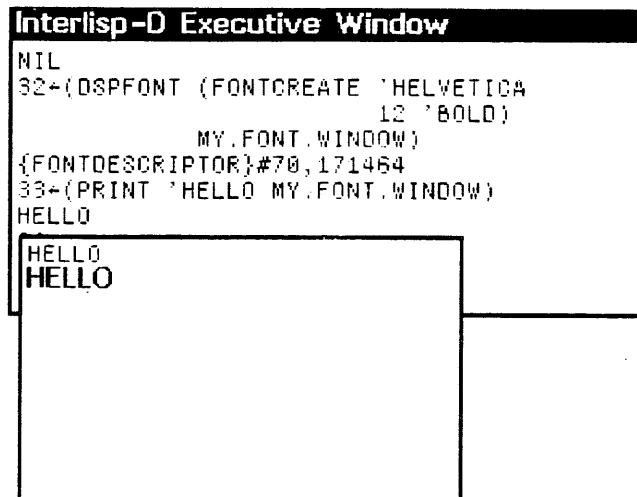


Figure 31.6. The font in MY.FONT.WINDOW, changed  
Notice the font has been changed!

---

### 31.5.4 Globally Changing Fonts

---

There is a library package to globally change the fonts in all the windows. To use it, first load BIG.DCOM. (See Section 8.6, Page 8.4 for how to load a file.)

To change fonts in all windows using the package BIG.DCOM, type

**(NEWFONT <keyword>)**

There are four keywords for size of fonts to specify. They are HUGE, BIG, STANDARD, and MEDIUM. For example:

**(NEWFONT 'BIG)**

sets the fonts in ALL the windows to be a larger size. Note: this package changes the fonts everywhere, including the editor window and system menus. It is particularly useful to change the size of the font for demos.

---

### 31.5.5 Personalizing Your Font Profile

---

Interlisp-D keeps a list of default font specifications. This list is used to set the font in all windows where the font is not specifically set by the user (Section 31.5.3). The value of the atom FONTPROFILE is this list. (See Figure 31.7.)

A FONTPROFILE is a list of font descriptions that certain system functions access when printing output. It contains specifications for big fonts (used when pretty printing a function to type the function name), small fonts (used for printing comments in the editor), and various other fonts.

```

Interlisp-D Executive Window
53+FONTPROFILE
((DEFAULTFONT 1 (GACHA 10)
  (GACHA 8)
  (TERMINAL 8))
 (BOLDFONT 2 (HELVETICA 10 BRR)
  (HELVETICA 8 BRR)
  (MODERN 8 BRR))
 (LITTLEFONT 3 (HELVETICA 8)
  (HELVETICA 8 MIR)
  (MODERN 8 MIR))
 (BIGFONT 4 (HELVETICA 12 BRR)
  (HELVETICA 10 BRR)
  (MODERN 10 BRR))
 (USERFONT BOLDFONT)
 (COMMENTFONT LITTLEFONT)
 (LAMBDAFONT BIGFONT)
 (SYSTEMFONT)
 (CLISPFONT BOLDFONT)
 (CHANGEFONT)
 (PRETTYCOMFONT BOLDFONT)
 (FONT1 DEFAULTFONT)
 (FONT2 BOLDFONT)
 (FONT3 LITTLEFONT)
 (FONT4 BIGFONT)
 (FONT5 5 (HELVETICA 10 BIR)
  (HELVETICA 8 BIR)
  (MODERN 8 BIR))
 (FONT6 6 (HELVETICA 10 BRR)
  (HELVETICA 8 BRR)
  (MODERN 8 BRR))
 (FONT7 7 (GACHA 12)
  (GACHA 12)
  (TERMINAL 12)))
54+

```

Figure 31.7. The value of the atom FONTPROFILE

The list is in the form of an association list. The font class names, (e.g. **DEFAULTFONT**, or **BOLDFONT**) are the keywords of the association list. When a number follows the keyword, it is the font number for that font class.

The lists following the font class name or number are the font specifications, in a form that the function **FONTCREATE** can use. The first font specification list after a keyword is the specification for printing to windows. The list, (GACHA 10), in the figure above is an example of the default specification for the printing to windows. The last two font specification lists are for Press and Interpress file printing, respectively. For more information, see the *Interlisp-D Reference Manual*, Volume 3, Chapter 27.

Now, to change your default font settings, change the value of the variable **FONTPROFILE**. Interlisp-D has a list of profiles stored as the value of the atom **FONTDEFS**. Choose the profile to use, then install it as the default **FONTPROFILE**.

Evaluate the atom **FONTDEFS** and notice that each profile list begins with a keyword. (See Figure 31.8.) This keyword corresponds to the size of the fonts included. **BIG**, **SMALL**, and **STANDARD** are some of the keywords for profiles on this list - **SMALL** and **STANDARD** appear in Figure 31.8.

```

[[SMALL (FONTPROFILE
  (DEFAULTFONT 1 (TERMINAL
    8)
    (GACHA 8)
    (TERMINAL 8))
  (BOLDFONT 2 (MODERN 8 BRR)
    (HELVETICA 8 BRR)
    (MODERN 8 BRR))
  (LITTLEFONT 3
    (MODERN 8 MIR)
    (HELVETICA 8 MIR)
    (MODERN 8 MIR))
  (TINYFONT 6 (MODERN 6)
    (GACHA 6)
    (MODERN 6))
  (BIGFONT 4 (MODERN 10 BRR)
    (HELVETICA 10 BRR)
    (MODERN 10 BRR))
  (TEXTFONT 5 (CLASSIC 10)
    (TIMESROMAN 10)
    (CLASSIC 10))
  (TEXTBOLDFONT 7
    (CLASSIC 10 BRR)
    (TIMESROMAN
      10 BRR)
    (CLASSIC 10 BRR)
[STANDARD (FONTPROFILE
  (DEFAULTFONT 1

```

Figure 31.8. Part of the value of the atom **FONTDEFS**

To install a new profile from this list, follow the following example, but insert any keyword for **BIG**.

To use the profile with the keyword **BIG** instead of the standard one, evaluate the following expression

```
(FONTSET 'BIG)
```

Now the fonts are permanently replaced. (That is, until another profile is installed.)



[This page intentionally left blank]

The Inspector is a window-oriented tool designed to examine data structures. Because Interlisp-D is such a powerful programming environment, many types of data structures would be difficult to see in any other way.

## 32.1 Calling the Inspector

Take as an example an object defined through a sequence of pointers (i.e. a bitmap on the property list of a window on the property list of an atom in a program.)

To inspect an object named NAME, type:

```
(INSPECT 'NAME)
```

If NAME has many possible interpretations, an option menu will appear. For example, in Interlisp-D, a litatom can refer to both an atom and a function. For example, if NAME was a record, had a function definition, and had properties on its property list, then the menu would appear as in Figure 32.1.

```
Which defn of NAME?
PROPS
FNS
FIELDS
```

Figure 32.1. Option Window For Inspection of NAME

If NAME were a list, then the option menu shown in Figure 32.2 would appear. The options include:

- calling the display editor on the list;
- calling the TTY editor (the "Typing Shortcuts", Chapter 6);
- seeing the list's elements in a display window. If you choose this option, each element in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of numbers. (See Figure 32.3.)
- inspecting the list as a record type (this last option would produce a menu of known record types). If you choose a record type, the items in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of the field names of the record.

```
DisplayEdit
TtyEdit
Inspect
As a record
```

Figure 32.2. Option Window For Inspection of List

## 32.2 Using the Inspector

If you choose to display your data structure in an edit window, simply edit the structure and exit in the normal manner when done. If you choose to display the data structure in an inspect window, then follow these instructions:

- To select an item, point the mouse cursor at it and press the left mouse button.
- Items in the right column of an Inspector window can themselves be inspected. To do this, choose the item, and press the center mouse button.
- Items in the right column of an Inspector window can be changed. To do this, choose the corresponding item in the left column, and press the center mouse button. You will be prompted for the new value, and the item will be changed. The sequence of steps is shown in Figure 32.3.

```
(INSPECT-ME-TOO1 INSPECT-ME-TOO2)
1 INSPECT-ME-TOO1
2 INSPECT-ME-TOO2
3 INSPECT-ME-TOO3
```

The item in the left column is selected, and the middle mouse button pressed. Select the SET option from the menu that pops up.

```
The expression read will be EVALuated.
> 'CHANGED-VALUE
(INSPECT-ME-TOO1 INSPECT-ME-TOO2)
1 INSPECT-ME-TOO1
2 INSPECT-ME-TOO2
3 INSPECT-ME-TOO3
```

You will then be prompted for the new value. Type it in.

```
(INSPECT-ME-TOO1 INSPECT-ME-TOO2)
1 INSPECT-ME-TOO1
2 INSPECT-ME-TOO2
3 CHANGED-VALUE
```

The item in the right column is updated to the value of what you typed in.

**Figure 32.3.** The sequence of steps involved in changing a value in the right column of an Inspector window.

## 32.3 Inspector Example

This example will use ideas discussed in Section 37.1. An example, ANIMAL.GRAPH, is created in that section. You do not need to know the details of how it was created, but the structure will be examined in this chapter.

If you type

```
(INSPECT ANIMAL.GRAPH)
```

and then choose the **Inspect** option from the menu, a display appears as shown in Figure 32.4. ANIMAL.GRAPH is being

inspected as a list. Note the numbers in the left column of the inspector window.

```

(((FISH & NIL NIL NIL --) (BIRD & NIL NIL --) (CAT & NIL
1 ((FISH & NIL NIL --) (BIRD & NIL NIL
2 T
3 NIL
4 NIL
5 NIL
6 NIL
7 NIL
8 NIL
9 NIL
10 NIL
11 NIL
12 NIL

```

**Figure 32.4.** Inspector Window For ANIMAL.GRAPH, inspected as a list.

If you choose the "As A Record" option, and choose "GRAPH" from the menu that appears, the inspector window looks like Figure 32.5. Note the fieldnames in the left column of the inspector window.

```

(((FISH & NIL NIL NIL --) (BIRD & NIL NIL --) (CAT & NIL --) (DOG & --) (& --) --)
GRAPH.CHANGELABELFN NIL
GRAPH.INVERTLABELFN NIL
GRAPH.INVERTBORDERFN NIL
GRAPH.FONTCHANGEFN NIL
GRAPH.DELETELINKFN NIL
GRAPH.ADDLINKFN NIL
GRAPH.DELETENODEFN NIL
GRAPH.ADDNODEFN NIL
GRAPH.MOVENODEFN NIL
DIRECTEDFLG NIL
SIDESFLG T
GRAPHNODES ((FISH & NIL NIL --) (BIRD & NIL NIL

```

**Figure 32.5.** Inspector Window For ANIMAL.GRAPH, inspected as an instance of a "GRAPH" record.

The remaining examples will use ANIMAL.GRAPH inspected as a list. When the first item in the Inspector window is chosen with the left mouse button, the Inspector window looks like Figure 32.6.

```

(((FISH & NIL NIL NIL --) (BIRD & NIL NIL --) (CAT & NIL
1 ((FISH & NIL NIL --) (BIRD & NIL NIL
2 T
3 NIL
4 NIL
5 NIL
6 NIL
7 NIL
8 NIL
9 NIL
10 NIL
11 NIL
12 NIL

```

**Figure 32.6.** Inspector Window For ANIMAL.GRAPH With First Element Selected  
When you use the middle mouse button to inspect the selected list element, the display looks like Figure 32.7.

```

(((FISH & NIL NIL NIL --) (BIRD & NIL NIL --) (CAT & NIL
1 ((FISH & NIL NIL --) (BIRD & NIL NIL
2 T
3 NIL
4 NIL
5 NIL
6 NIL
7 NIL
8 NIL
9 NIL
10 NIL
11 NIL
12 NIL
)) (FISH (102 . 44) NIL NIL NIL --) (BIRD (102 . 29) NIL NI
1 (FISH (102 . 44) NIL NIL NIL --)
2 (BIRD (102 . 29) NIL NIL NIL --)
3 (CAT (186 . 22) NIL NIL NIL --)
4 (DOG (186 . 7) NIL NIL NIL --)
5 ((MAMMAL DOG CAT) (109 . 14) NIL NIL
6 ((ANIMAL & BIRD FISH) (22 . 29) NIL

```

**Figure 32.7.** Inspector Window For ANIMAL.GRAPH and For the First Element of ANIMAL.GRAPH

Now you can see that six items make up the list, and you can further choose to inspect one of these items. Notice that this is also inspected as a list. As usual, it could also have been inspected as a record.

Select item 5 - MAMMAL DOG CAT - with the left mouse button. Press the middle mouse button. Choose "Inspect" to inspect your choice as a list. The Inspector now displays the values of the structure that makes up MAMMAL DOG CAT. (See Figure 32.8.)

```

((MAMMAL DOG CAT) (109 . 14) N
1 (MAMMAL DOG CAT)
2 (109 . 14)
3 NIL
4 NIL
5 NIL
6 45
7 15
8 (DOG CAT)
9 ((ANIMAL & BIRD FISH)
10 {FONTCLASS}#70,172764
11 MAMMAL
12 NIL

```

**Figure 32.8.** Inspector Window for Element 5 From Figure 32.7 That Begins ((MAMMAL DOG CAT).

Masterscope is a tool that allows you to quickly examine the structure of complex programs. As your programs enlarge, you may forget what variables are global, what functions call other functions, and so forth. Masterscope keeps track of this for you.

Suppose that JVTO is the name of a file that contains many of the functions involved in a complex system and that LINTRANS is the file containing the remaining functions. The first step is to ask Masterscope to analyze these files. These files must be loaded. All Masterscope queries and commands begin with a period followed by a space, as in

**. ANALYZE FNS ON JVTO**

The ANALYZE process takes a while, so the system prints a period on the screen for each function it has analyzed. (See Figure 33.1.)

```
62+. ANALYZE FNS ON JVTO
.....done
63+. ANALYZE FNS ON LINTRANS
.....done
```

**Figure 33.1.** The Interlisp-D Executive Window after analyzing the files

If you are not quite sure what functions were just analyzed, type the file's **COMS** variable (See Section 11.5, Page 11.7.) into the Interlisp-D Executive Window. The names of the functions stored on the file will be a part of the value of this variable.

A variety of commands are now possible, all referring to individual functions within the analyzed files. Substantial variation in exact wording is permitted. Some commands are:

- . **SHOW PATHS FROM ANY TO ANY**
- . **EDIT WHERE ANY CALLS *functionname***
- . **EDIT WHERE ANY USES *variablename***
- . **WHO CALLS WHOM**
- . **WHO CALLS *functionname***
- . **BY WHOM IS *functionname* CALLED**
- . **WHO USES *variablename* AS FIELD**

Note that the function **.** is being called to invoke each command. Refer to the *Interlisp-D Reference Manual* for commands not listed here.

Figure 33.2 shows the Interlisp-D Executive Window after the commands **. WHO CALLS GobbleDump** and **. WHO DOES JVLinScan CALL**.

```

NIL
77+. WHO CALLS GobbleDump
(JVchapterTO JVdgd JVfndefTO JVnilTO DoArgSpec GobbleFlush GobbleString
JVdumpTO)
78+. WHO DOES JVLinScan CALL
(LinScan JVCTtable JVTotable)
79+^

```

Figure 33.2. Sample Masterscope Output

### 33.1 The SHOW DATA command and GRAPHER

When the library package GRAPHER is loaded, (to load this package, type (FILESLOAD GRAPHER).) Masterscope's SHOWPATHS command is modified. The command will be changed to generate a tree structure showing how the program's functions interact instead of a tabular printout into the Interlisp-D Executive window. For example, typing:

. SHOW PATHS FROM ProcessEND.

produced the display shown in Figure 33.3.

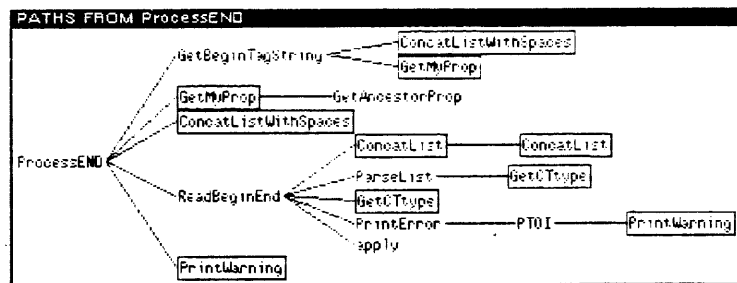


Figure 33.3. SHOW PATHS Display Example

All the functions in the display are part of this analyzed file or a previously analyzed file. Boxed functions indicate that the function name has been duplicated in another place on the display.

Selecting any function name on the display will pretty print the function in a window. (See Figure 33.4.)

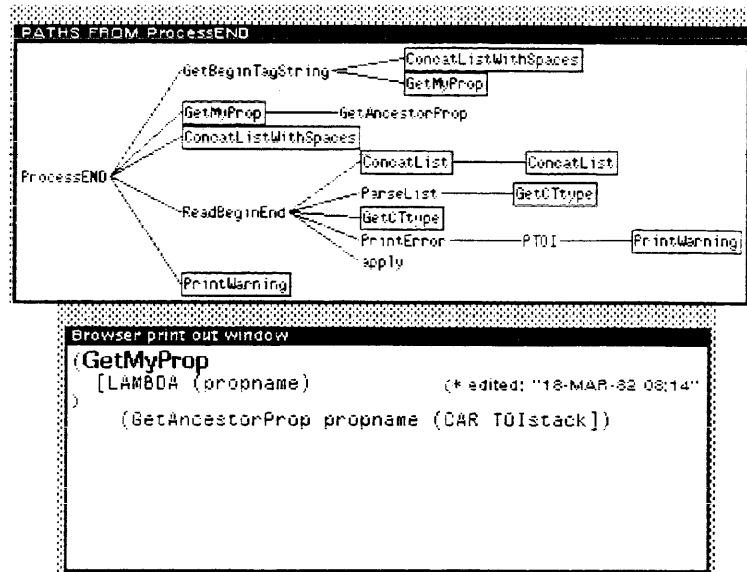


Figure 33.4. Browser Printout Example.

Selecting it again with the left mouse button will produce a description of the function's role in the overall system. (See Figure 33.4.)

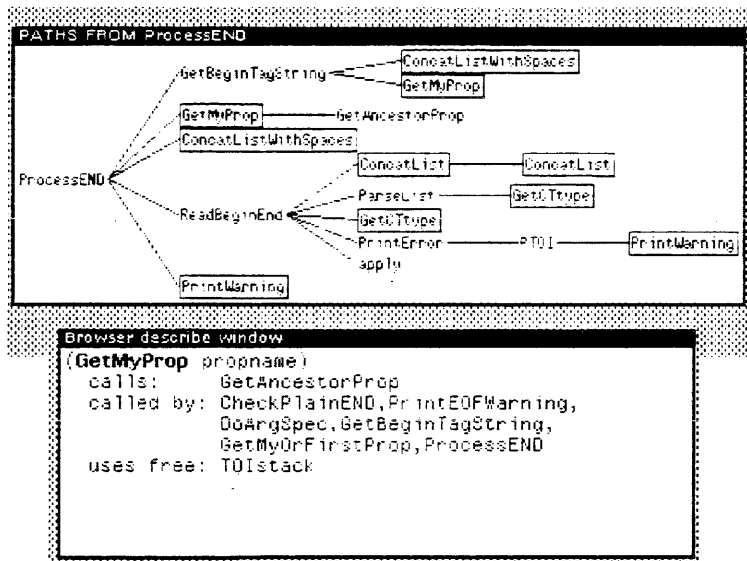


Figure 33.5. Browser Description Example.

## 33.2 DatabaseFns: Automatic Construction and Upkeep of a Masterscope Database

DataBaseFns is a separate library package that allows you to automatically construct and maintain Masterscope databases of your files. The package is contained in the DATABASEFNS.DCOM file.

When DATABASEFNS.DCOM is loaded, a Masterscope database will be automatically maintained for every file whose DATABASE



property has the value YES. If this property's value is not set, you will be asked when you save the file "Do you want a Masterscope Database for this file?". Saying YES enables the DabaBaseFns to construct a Masterscope database of the file you are saving.

Each time the function **MAKEFILE** is used on a file whose **DATABASE** property has a value YES, Masterscope will analyze your file and update its own database. Each file's masterscop database is kept in a separate file whose name has the form *FILE.DATABASE*. Whenever you load a file with a YES value for its **DATABASE** property, you will be asked whether you also want the database file loaded.

# 34. WHERE DOES ALL THE TIME GO? SPY

---

SPY is an Interlisp-D library package that shows you where you spend your time when you run your system. It is easy to learn, and very useful when trying to make programs run faster.

---

## 34.1 How to use Spy with the SPY Window

---

The function `SPY .BUTTON` brings up a small window which you will be prompted to position. Using the mouse buttons in this window controls the action of the SPY program. When you are not using SPY, the window appears as in Figure 34.1.



**Figure 34.1.** The SPY window when SPY is not being used.

To use SPY, click either the left or middle mouse button with the mouse cursor in the SPY window. The window will appear as in Figure 34.2, and means that SPY is accumulating data about your program.



**Figure 34.2.** The SPY window when SPY is being used.

To turn off SPY after the program has run, again click a mouse button in the SPY window. The eye closes, and you are asked to position another window. This window contains SPY's results. An example of result window is shown in Figure 34.3.

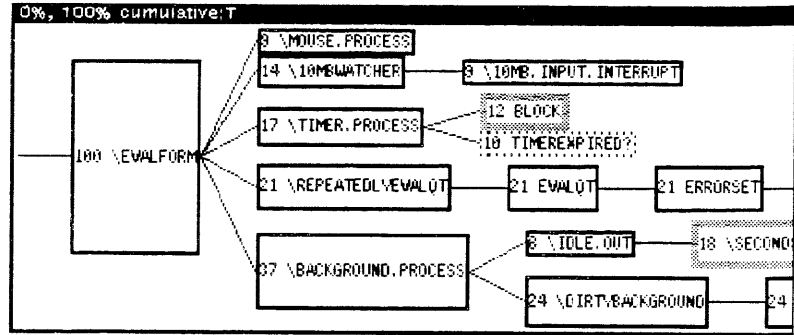


Figure 34.3. The window produced after running SPY

This window is scrollable in two directions, horizontally, and vertically. This is useful, since the whole tree does not fit in the window. If a part that you want to see is not shown, then you can scroll the window to show the part you want to see.

## 34.2 How to use SPY from the Lisp Top Level

SPY can also be run while a specific function or system is being used. To do this, type the function `WITH.SPY`:

(`WITH.SPY form`)

The expression used for *form* should be the call to begin running the function or system that SPY is to watch. If you watch the SPY window, the eye will blink! To see your results, run the function `SPY.TREE`. To do this, type:

(`SPY.TREE`)

The results of the last running of SPY will be displayed. If you do this, and `SPY.TREE` returns (**no SPY samples have been gathered**), your function ran too fast for SPY to follow.

## 34.3 Interpreting SPY's Results

Each node in the tree is a box that contains, first, the percentage of time spent running that particular function, and second, the function name. There are two modes that can be used to display this tree.

The default mode is cumulative. In this mode, each percentage is the amount of time that function spent on top of the stack, plus the amount of time spent by the functions it calls.

The second mode is individual. To change the mode to individual, point to the title bar of the window, and press the middle mouse button. Choose Individual from the menu that appears. In this mode, the percentage shown is the amount of time that the function spent on the top of the stack.

To look at a single branch of the tree, point with the mouse cursor at one of the nodes of the tree, and press the right mouse button. From the menu that appears, choose the option SubTree. Another SPY window will appear, with just this branch of the tree in it.

Another way to focus within the tree is to remove branches from the tree. To do this, point to the node at the top of the branch you would like to delete. Press the middle mouse button, and choose Delete from the menu that appears.

There are also different amounts of "merging" of functions that can be done in the window. A function can be called by another function more than once. The amount of merging determines where the subfunction, and the functions that it calls, appear in the tree, and how often. (For a detailed explanation of merging, see the *Lisp Library Packages Manual*.)

[This page intentionally left blank]

Sketch is a Xerox package that was developed for constructing pictures. Unlike bitmaps, you do not need to draw every pixel for the shape you want. One example will guide you through various sketch capabilities. However, not everything that it can do can be shown here. To learn more about it, refer to *A User's Guide to Sketch: The Interlisp Drawing System*. The manual is very clear, and contains many figures drawn with Sketch to illustrate its points.

---

### 35.1 Starting Sketch

---

To start sketch, type (**FILESLOAD SKETCH**) into the Interlisp-D Executive window. This loads the necessary files, and adds SKETCH to the right button background menu (the menu that appears when you press the right button of the mouse outside any window).

Choose Sketch from this menu. You will be prompted to sweep out a window. This window will hold the figure that you draw.

---

### 35.2 Selecting Sketch elements

---

A sketch is a picture that consists of sketch elements. Each element has one or more control points, that determine its location, its shape, and other properties that determine how it looks. Many times in the example that follows you will be asked to select an element. You should first select a command to perform on an element, then select one or more elements as arguments for that command.

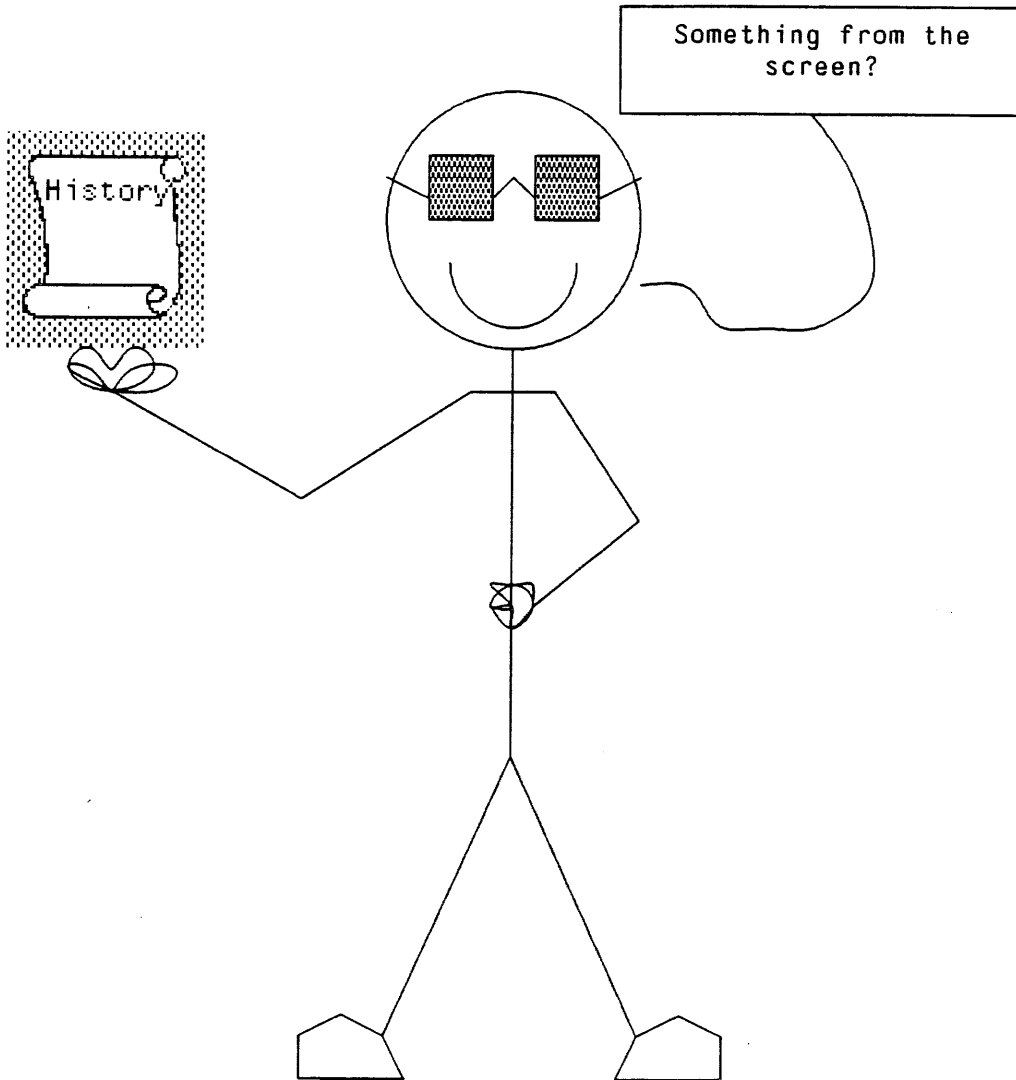
After a command is selected, each element displays a control box showing where to click to select that element. To choose one element, click the left mouse button when it is pointing to the control box of the element. To choose a group of elements, move the mouse to a point that is to the upper left of the group. Press and hold down the left mouse button. Sweep out an area that contains the control boxes of each element you would like to select. Release the button to choose this area.

You can always abort a selection by holding the left mouse button down, moving the mouse cursor outside the Sketch window, and then releasing the mouse button. Also note that if

you select a command and then press the left mouse button in the sketch window, when the mouse cursor is not in the control box of an element, the command is aborted.

### 35.3 Drawing with Sketch

If you haven't done so already, choose SKETCH from the background menu, and sweep out a window. You will be drawing the following picture:



**Figure 35.1.** The completed figure that will be drawn as an example

You will use two menus extensively: the command menu for sketch, which appears to the right of the sketch window (See Figure 35.2.); and the default right button window menu, which appears when you point the mouse cursor to the sketch window's title bar, and hold down the right mouse button.

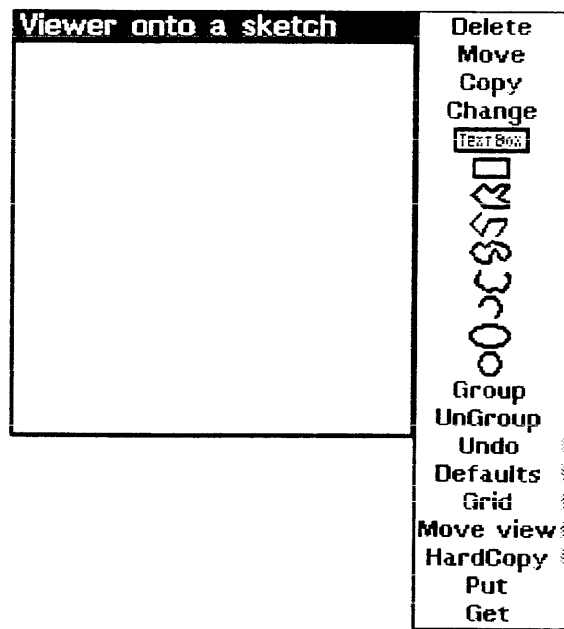


Figure 35.2. An example Sketch window, with its command menu

### 35.3.1 Simple Shapes: Circles, Ellipses, and Boxes

#### 35.3.1.1 Drawing Circles

To draw a circle, choose the circle from the command menu. You will be prompted for the point at the center of the circle, then a point on the circle itself. Do this on your sketch to draw the head of the man.

#### 35.3.1.2 Ellipses

To draw an ellipse, choose that shape from the command menu. You will be prompted for three points. The first is the center of the ellipse. The second is the length of its long radius, and the third is the length of its short radius. Do this twice, once for each eye, or, if you prefer square eyes (or perhaps the frames of glasses) see Section 35.3.1.3.

#### 35.3.1.3 Boxes

A box can be added to a sketch by choosing the box from the command menu. You will be asked to sweep out a box on the sketch window. Do this in the same way you sweep out a window.

- (1) Choose the position of the corner of the box
- (2) Point to this spot with the mouse, and press and hold down its left button



- (3) Move the mouse until the box is the correct size and shape.
- (4) Release the left button to make this box appear.

#### 35.3.1.4 Changing a Box's Filling

---

If you chose to give your drawing glasses, you may want to make them sunglasses by changing the filling of the box. To do this, choose **Change** from the command menu, then select the boxes. Choose **Filling** from the menu that will appear. Yet another menu, one that shows the various fillings, will appear. Choose one of those. Your stick man may now look like this:

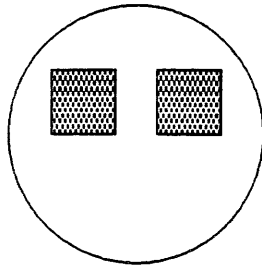


Figure 35.3. The SKETCH man's head, with glasses

### 35.3.2 Lines, Curves, and Arcs

---

#### 35.3.2.1 A Single Line

---

To add a single line to a sketch, to add the side bars to the sunglasses for example,

- (1) Press and hold down the middle mouse button inside the sketch window.
- (2) Move the mouse cursor so that it is pointing to the place where the line should start, and release the middle mouse button.
- (3) Press and hold the middle mouse button again, and move the mouse until the mouse cursor is pointing to the place where the line should end.
- (4) When you release the mouse button, the line will be placed between the two endpoints.

#### 35.3.2.2 A Series of Lines

---

For the body of the man, you can draw a series of lines either singly, or all at once. To add them all at once,

- (1) Choose the jagged line from the command menu.
- (2) Move the cursor to each point the line should go through, and click the left button.

- (3) When all of the points have been chosen, click the left button outside the sketch window, and the line will appear.

Do this to form the body of the man. Your figure may now look like this:

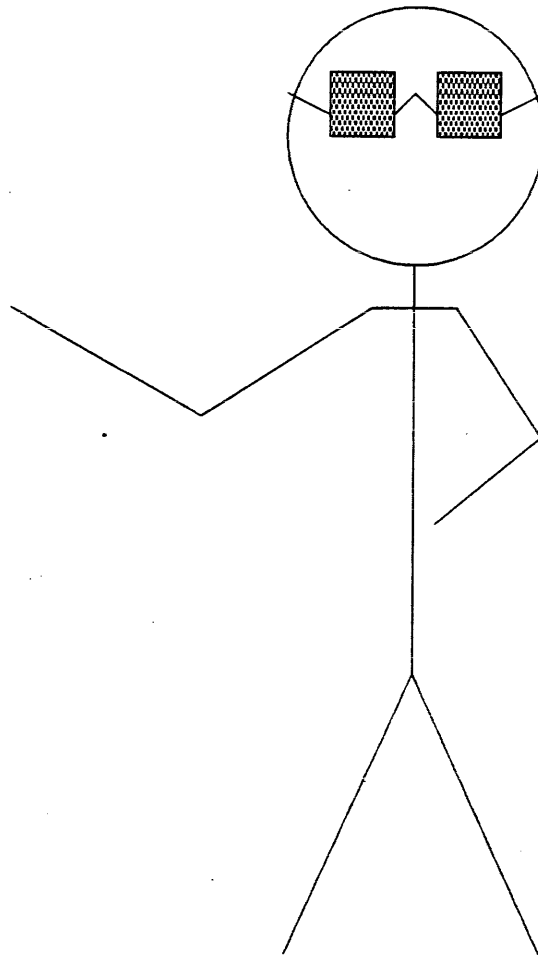


Figure 35.4. The Man, with lines for his body, and glasses frames

### 35.3.2.3 Drawing an Open Curve

The nose of the man is drawn with an open curve. To draw an open curve,

- (1) Choose this item from the command menu.
- (2) Click the left button at each point that the curve should go through.
- (3) When all the points have been chosen, click the left button outside the sketch window, and the curve will appear.

### 35.3.2.4 An Arc

Part of a circle, a simple arc, works well for the mouth. To draw an arc,

- (1) Choose the semi-circle from the command menu.

- (2) You will be prompted to choose the center point of the arc first.
- (3) The second point to choose determines both the radius and one end of the arc.
- (4) The third point to be selected is the other end of the arc.

To add an arrowhead to the arc, or to change its angle, direction, or another property,

- (1) Choose Change from the command menu;
- (2) Select one of the control points of the arc;
- (3) Choose the property of the arc that you would like to change from the menu that appears.

### **35.3.3 Closed Curves and Polygons**

---

Both of these elements make wonderful hands and feet. Select the appropriate item from the command menu. The polygon looks like an angular closed shape (it is not the box), and the closed curve is a closed curvey item.

For the polygon, select its vertices by clicking the left mouse button at the appropriate points. Select them in the order that the line should be drawn through them to create the structure.

For the closed curve, select the points that the curve should go through in the same way. You can use either of these elements to add hands and feet to the figure. It may now look like this:

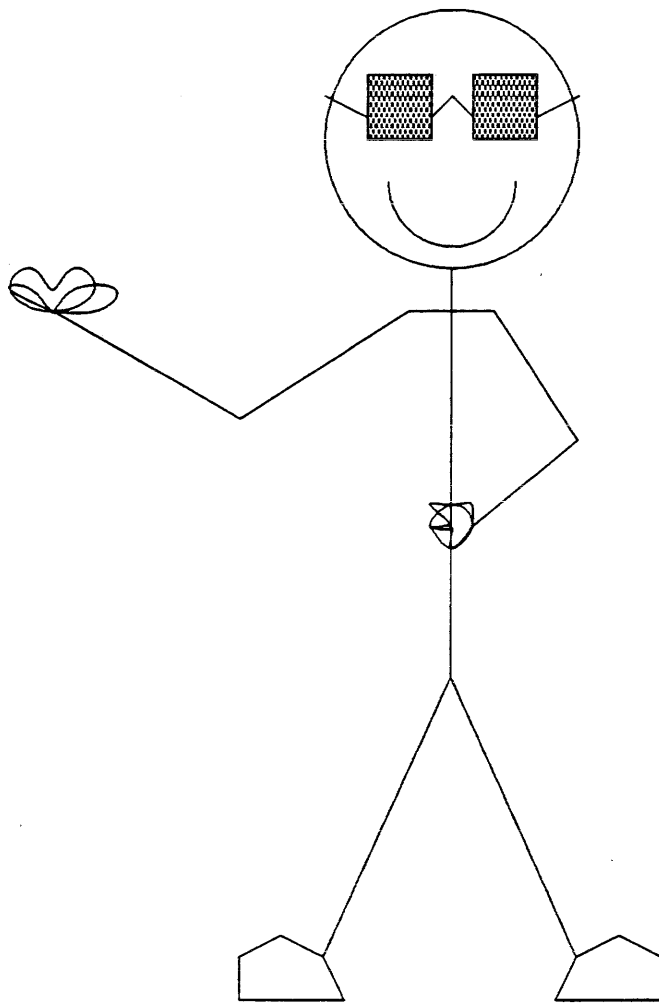


Figure 35.5. The sketch, after the addition of an arc, closed curves, and polygons

---

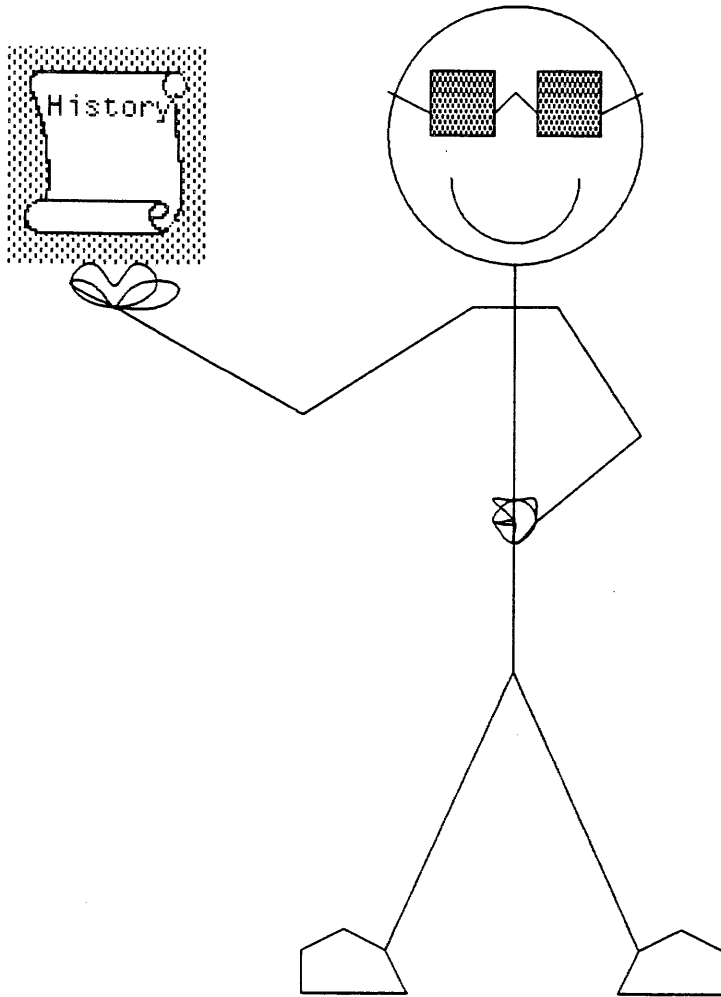
## 35.4 Adding a Bitmap to a Sketch

---

To insert a bitmap from the screen,

- (1) Make a caret appear by clicking the left button in the sketch window.
- (2) Move the mouse cursor onto the grey background, and hold down the COPY or the SHIFT key.
- (3) Choose Snap from the menu that will appear.
- (4) Sweep out the area of the screen that you would like to insert into the sketch.
- (5) After you have done this, you will be prompted to "Move the picture into place, and press the left button." When you do this, the bitmap will appear in your sketch.

To continue drawing the example picture, do this, and place the bitmap on top of the figure's raised hand. Your bitmap might look something like this:



**Figure 35.6.** The man, holding a bitmap

To edit the bitmap, position the mouse cursor in the bitmap, and press the left button. A menu will appear. Choose HAND.EDIT, and you will be placed into the bitmap editor. Use this in the normal way (for how to edit bitmaps, see Chapter 29).

---

## 35.5 To Add Text to a Sketch

---

You can either add text directly to a Sketch window, or you can place text inside a text box. To add text directly, click the left mouse button in the window, and a caret will appear. Anything you type will appear at the caret. If the caret does not appear, but a vertical line appears instead, you are inside a text element. (If you type, the text will be added to the text element that already exists. Another one will not be created.) To create a new

text element, move the mouse, and click the left button again, in an area where there is no text.

If you prefer to begin by creating a box for the text, choose the TextBox item from the command menu. Sweep out the area for the box, and it will appear. Clicking the mouse inside this box will cause a vertical bar to appear. Type, and the text will be placed at the caret. The text will be centered, and the lines will wrap automatically.

### 35.5.1 Editing Text

The editing commands to change text are much like those used in TEdit. (see TEdit, Chapter 23).

- |                             |  |
|-----------------------------|--|
| Moving the caret            | to a certain point in the text, point the mouse to the new position for the caret, and click the left button.  |
| Deleting a single character | can be done with the backspace key. The character before the caret will be deleted.  |
| Deleting the word           | behind the caret is done by typing Control-W.  |
| Deleting a block of text    | is done with the following set of steps: <ol style="list-style-type: none"> <li>(1) Move the caret to one end of the block of text;</li> <li>(2) Press and hold the right mouse button. Move the mouse so that it points to the other end of the text. The text will be highlighted.</li> <li>(3) Press the delete key.</li> </ol>   |
| Replacing a block of text   | is done by selecting the text as though it is to be deleted. Instead of pressing the delete key, simply type in the replacement text. <p>If the text is boxed, you can change either the box, or the text inside it. To do either,</p> <ol style="list-style-type: none"> <li>(1) Choose Change from the Sketch command menu(Figure 35.2)</li> <li>(2) Choose one of the corners of the box</li> <li>(3) From the menu that appears, choose "The text" to change the text inside the box (for example, to change its font). The other items on the menu are ways to change the box itself. Choose the appropriate item for the change that you would like to make.</li> </ol> <p>If the text is unboxed, you can choose to box the text, or to change the entire text item in some way (for example, to change the font). To do either,</p> <ol style="list-style-type: none"> <li>(1) Choose Change from the Sketch command menu (Figure 35.2)</li> <li>(2) Select the control point of the text item</li> <li>(3) From the menu that appears, you can choose the appropriate change for the text.</li> </ol> |

The final addition to the drawing then, is to add the little man's words by adding a text box. Do this, and perhaps an arc from the box to the man, and your bitmap might look something like this:

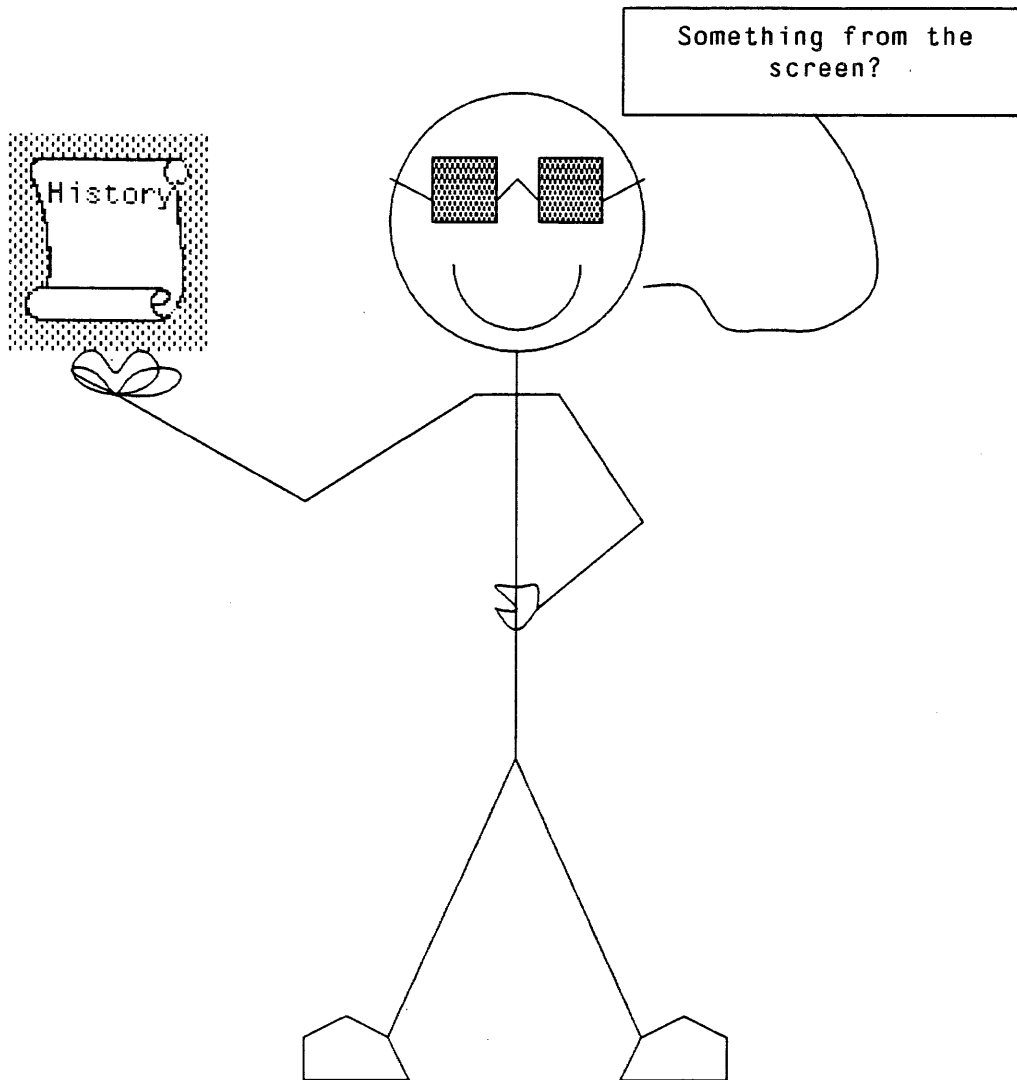


Figure 35.7. The man, speaking

## 35.6 Editing a Sketch

Deleting an Item	Choose the delete command, then select the item or items to be deleted. These items will be removed from the sketch.
To Move an item	Select the Move command from the Sketch command menu. To choose the control point of the element to be moved, press and hold the left button of the mouse down over one of the control points of the item. If there is more than one for the sketch item, when you hold down the left button and move the mouse off of the control point, the others should darken also. Release the left button to choose this item. You'll then be prompted to move it to its new location.
To Move multiple items	Select the move command, and select the group of control points in the usual way (see section Section 35.2, above). Once they are

	chosen, you will be prompted to move them to their new location.
Copying items	First, select the Copy command, then choose the item or items you wish to copy. You will then be prompted to "Move the figure into place, and press the left button." A copy of the item(s) will appear at the location you chose.
Changing the look of an item	Choose change, then choose the item(s) that the change should affect. A menu will appear that includes such items as Size (to change the thickness of the item) and Dashing (to change to a dashed line from a solid line). Choose the change that you would like to make.
Grouping items	Select the Group command to treat a set of items like a single Sketch element. Choose the Sketch elements that should be in the group. Until you Ungroup them, this group of elements will be treated as a single item. When you choose a command, such as Delete, a single control box will appear for the group.

---

## 35.7 Saving Your Work

---

When you are finished making a sketch, save it in a file by choosing the PUT command. You will be prompted for a filename. Type it in, and press **CR** when you are done. If a filename already appears after the prompt, and it is the correct name, simply type **CR**. If it is not the correct filename, a back space will delete a single character, and Control-W will delete the word behind the cursor.

To stop sketch without saving your work, choose close from the default right button menu (move the mouse cursor so that it points to the title bar of the sketch window, and press the right mouse button). You will be asked to press the left mouse button to confirm that you want to stop. You will leave sketch without saving your work. Otherwise, press any other mouse button to continue without stopping.

To print a copy of your sketch, choose Hardcopy from the default right button window menu. Often a printer can not accurately print your drawing; don't be surprised if the hardcopy drawing differs from the screen drawing.

---

## 35.8 To Continue a Sketch That Has Been Saved on a File

---

To continue a Sketch that has been saved on a file, open a Sketch window, and choose Get from the command menu. You will be prompted for a filename. Once it appears, you can continue your work on the drawing. Using the function GET merges the file with whatever is currently in the Sketch window. This will be



changed in future versions of Sketch, but for now, be careful about using this command.

Free Menu is a library package that is even more flexible than the regular menu package. It allows you to create menus with different types of items in them, and will format them as you would like. Free menus are particularly useful when you want a "fill in the form" type interaction with the user.

Each menu item is described with a list of properties and values. The following example will give you an idea of the structure of the description list, and some of your options. The most commonly used properties, and each type of menu item will be described in Section 36.2 and Section 36.3.

---

### 36.1 An Example Free Menu

---

Free menus can be created and formatted automatically! It is done with the function `FM.FORMATMENU`. This function takes one argument, a description of the menu. The description is a list of lists; each internal list describes one row of the free menu. A free menu row can have more than one item in it, so there are really lists of lists of lists! It really isn't hard, though, as you can see from the following example:

```
(SETQ ExampleMenu
  (FM.FORMATMENU
    '(((TYPE TITLE LABEL TitlesDoNothing)
      (TYPE 3STATE LABEL Example3State))
      ((TYPE EDITSTART LABEL PressToStartEditing
        ITEMS (EDITITEM))
        (TYPE EDIT ID EDITEM LABEL ""))
      (WINDOWPROPS TITLE "Example Does Nothing"))))
```

The first row has 2 items in it; one is a `TITLE`, and the second is a `3STATE` item. The second row also has 2 items. The second, the `EDIT` item, is invisible, because its label is an empty string. The caret will appear for editing, however, if the `EDITSTART` item is chosen. `Windowprops` can appear as part of the description of the menu, because a menu is, after all, just a special window. You can specify not only the title with `WINDOWPROPS`, but also the position of the free menu, using the "left" and "bottom" properties, and the width of the border in pixels, with the "border" property. Evaluating this expression will return a window. You can see the menu by using the function `OPENW`. The following example illustrates this:

```

88+(SETQ ExampleMenu
  (FM.FORMATMENU '(((TYPE TITLE LABEL TitlesDoNothing)
                  (TYPE 3STATE LABEL Example3State))
                ((TYPE EDITSTART
                  LABEL PressToStartEditing
                  ITEMS (EDITEM))
                 (TYPE EDIT ID EDITEM LABEL ""))
                (WINDOWPROPS
                  TITLE "Example Does Nothing"))))
(ExampleMenu reset)
{WINDOW}#54,121404
89+(OPENW ExampleMenu)
{WINDOW}#54,121404
90^A

```

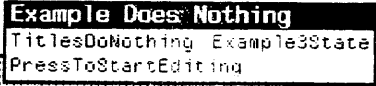


Figure 36.1. An example free menu

The next example shows you what the menu looks like after the EDITSTART item, PressToStartEditing, has been chosen.

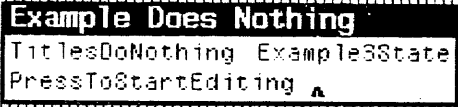


Figure 36.2. Free menu after the EDITSTART item has been chosen

The following example shows the menu with the 3STATE item in its T state, with the item highlighted (In the previous bitmaps, it was in its neutral state.)

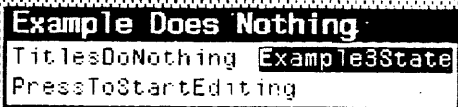


Figure 36.3. Free menu with the 3STATE item in its T state

Finally, Figure 36.4 shows the 3STATE item in its NIL state, with a diagonal line through the item

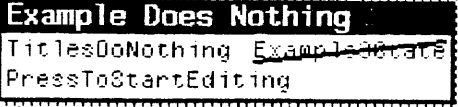


Figure 36.4. Free menu with the 3STATE item in its NIL state

If you would like to specify the layout yourself, you can do that too. See the *Lisp Library Packages Manual* for more information.

## 36.2 Parts of a Free Menu Item

There are 8 different types of items that you can use in a free menu. No matter what type, the menu item is easily described by a list of properties, and values. Some of the properties you will use most often are:

LABEL	Required for every type of menu item. It is the atom, string, or bitmap that appears as a menu selection.
TYPE	One of eight types of menu items. Each of these are described below.
MESSAGE	The message that will appear in the prompt window if a mouse button is held down over the item.
ID	An item's unique identifier. An ID is needed for certain types of menu items.
ITEMS	Used to list a series of choices for an NCHOOSE item, and to list the ID's of the editable items for an EDITSTART item.
SELECTEDFN	The name of the function to be called if the item is chosen

### 36.3 Types of Free Menu Items

	Each type of menu item is described in the following list, including an example description list for each one.
Momentary	<p>This is the familiar sort of menu item. When it is selected, the function stored with it is called. A description for the function that creates and formats the menu looks like this:</p> <pre>(TYPE MOMENTARY   LABEL Blink-N-Ring   MESSAGE "Blinks the screen and rings bells"   SELECTEDFN RINGBELLS)</pre>
TOGGLE	<p>This menu item has two states, T and NIL. The default state is NIL, but choosing the item toggles its state. The following is an example description list, without code for the SELECTEDFN function, for this type of item:</p> <pre>(TYPE TOGGLE   LABEL DwinDisable   SELECTEDFN ChangeDWIMState)</pre>
3STATE	<p>This type of menu item has 3 states, NUETRAL, T, AND NIL. Neutral is the default state. T is shown by highlighting the item, and NIL is shown with diagonal lines. The following is an example description list, without code for the SELECTEDFN function, for this type of item:</p> <pre>(TYPE 3STATE   LABEL CorrectProgramA110rNoSpelling   SELECTEDFN ToggleSpellingCorrection)</pre>
TITLE	<p>This menu item appears on the menu as dummy text. It does nothing when chosen. An example of its description:</p> <pre>(TYPE TITLE LABEL "Choices:")</pre>
NWAY	<p>A group of items, only one of which can be chosen at a time. The items in the NWAY group should all have an ID field, and the ID's should be the same. For example, to set up a menu that would allow the user to chose between Helvetica, Gacha, Modern, and Classic fonts, the descriptions might look like this (Once again, without the code for the SELECTEDFN):</p> <pre>(TYPE NWAY ID FONTCHOICE   LABEL Helvetica   SELECTEDFN ChangeFont)</pre>

```
(TYPE NWAY ID FONTCHOICE
 LABEL Gacha
 SELECTEDFN ChangeFont)
(TYPE NWAY ID FONTCHOICE
 LABEL Modern
 SELECTEDFN ChangeFont)
(TYPE NWAY ID FONTCHOICE
 LABEL Classic
 SELECTEDFN ChangeFont)
```

NCHOOSE This type of menu item is like NWAY except that the choices are given to the user in a submenu. The list to specify an NCHOOSE menu item that is analogous to the NWAY item above might look like this:

```
(TYPE NCHOOSE
 LABEL FontChoices
 ITEMS (Helvetica Gacha Modern Classic)
 SELECTEDFN ChangeFont)
```

EDITSTART When this type of menu item is chosen, it activates another type of item, an EDIT item. The EDIT item or items associated with an EDITSTART item have their ID's listed on the EDITSTART's ITEMS property. An example description list is:

```
(TYPE EDITSTART LABEL "Function to add?" ITEMS (Fn))
```

EDIT This type of menu item can actually be edited by you. It is often associated with an EDITSTART item (see above), but the caret that prompts for input will also appear if the item itself is chosen. An EDIT item follows the same editing conventions as editing in Interlisp-D Executive window:

Add Characters by typing them at the caret.

Move the caret by pointing the mouse at the new position, and clicking the left button.

Delete Characters from the caret to the mouse by pressing the right button of the mouse. Delete a character behind the caret by pressing the back space key.

Stop editing by typing a carriage return, a Control-X, or by choosing another item from the menu.

An example description list for this type of item is:

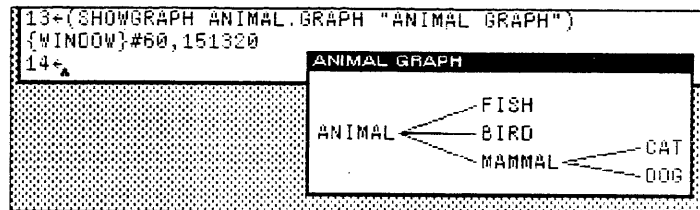
```
(TYPE EDIT ID Fn LABEL "")
```

## 37.1 Say it with Graphs

Grapher is a collection of functions for creating and displaying graphs, networks of nodes and links. Grapher also allows you to associate program behavior with mouse selection of graph nodes. To load this package, type

**(FILESLOAD GRAPHER)**

Figure 37.1 shows a simple graph.



**Figure 37.1.** A Simple Graph

In Figure 37.1 there are six nodes (ANIMAL, MAMMAL, DOG, CAT, FISH, and BIRD) connected by five links.

A GRAPH is a record containing several fields. Perhaps the most important field is GRAPHNODES - which is itself a list of GRAPHNODE records. Figure 37.2 illustrates these data structures. The window on top contains the fields from the simple graph. The window on the bottom is an inspection of the node, DOG.

```

19+ (INSPECT ANIMAL.GRAPH)
{WINDOW}#60,146470
20+ (((FISH & NIL NIL NIL --) (BIRD & NIL NIL --) (CAT & NIL --) (DOG & --) (& --) --
GRAPH.CHANGELABELFN  NIL
GRAPH.INVERTLABELFN  NIL
GRAPH.INVERTBORDERFN NIL
GRAPH.FONTCHANGEFN  NIL
GRAPH.DELETELINKFN  NIL
GRAPH.ADDLINKFN     NIL
GRAPH.DELETENODEFN  NIL
GRAPH.ADDNODEFN     NIL
GRAPH.MOVENODEFN    NIL
DIRECTEDFLG        NIL
SIDESFLG           T
GRAPHNODES         ((FISH & NIL NIL --) (BIRD & NIL NIL
(DOG (188 . 7) NIL NIL NIL --) Inspector
NODEBORDER         NIL
NODELABEL          DOG
NODEFONT           {FONTCLASS}#70,172764
FROMNODES         ((MAMMAL DOG CAT))
TONODES            NIL
NODEHEIGHT         15
NODEWIDTH          24
NODELABELSHADE    NIL
NODELABELBITMAP   NIL
NODEPOSITION      (188 . 7)
NODEID             DOG

```

Figure 37.2. Inspecting a Graph and a Node

The GRAPHNODE data structure is described by its text (NODEID), what goes into it (FROMNODES), what leaves it (TONODES), and other fields that specify its looks. The basic model of graph building is to create a bunch of nodes, then layout the nodes into a graph, and finally display the resultant graph. This can be done in a number of ways. One is to use the function **NODECREATE** to create the nodes, **LAYOUTGRAPH** to lay out the nodes, and **SHOWGRAPH** to display the graph. The primer shows you two simpler ways, but please see the *Library Packages Manual* for more information about these other functions. The primer's first method is to use **SHOWGRAPH** to display a graph with no nodes or links, then interactively add them. The second is to use the function **LAYOUTSEXPR**, which does the appropriate **NODECREATES** and a **LAYOUTGRAPH**, with a list.

The function **SHOWGRAPH** displays graphs and allows you to edit them. The syntax of **SHOWGRAPH** is

```
(SHOWGRAPH graph window leftbuttonfn middlebuttonfn
topjustifyflg alloweditflg copybuttoneventfn)
```

Obviously the graph structure is very complex. Here's the easiest way to create a graph.

```
(SETQ MY.GRAPH NIL)
(SHOWGRAPH MY.GRAPH "My Graph" NIL NIL NIL T)
```

```

22+ (SHOWGRAPH MY.GRAPH "My Graph" NIL NIL NIL T)
{WINDOW}#60,146150
23+

```




Figure 37.3. My Graph

You will be prompted to create a small window as in Figure 37.3. This graph has the title My Graph.

Hold down the right mouse button in the window. A menu of graph editing operations will appear as in Figure 37.4.

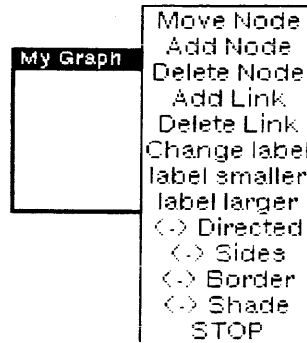


Figure 37.4. A Menu of Graph Editing Operations

Here's how to use this menu to:

#### Add a Node

Start by selecting **Add Node**. Grapher will prompt you for the name of the node (See Figure 37.5.) and then its position.

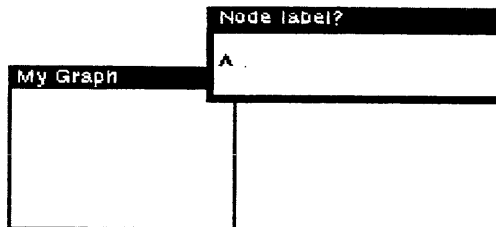


Figure 37.5. Grapher prompts for the name of the node to add after **Add Node** is chosen from the graph editing menu.

Position the node by moving the mouse cursor to the desired location and clicking a mouse button. Figure 37.6 shows the graph with two nodes added using this menu.

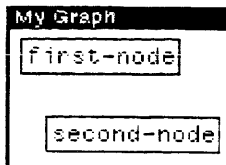


Figure 37.6. Two nodes added to MY.GRAPH using the graph editing menu

#### Add a Link

Select **Add Link** from the graph editing menu. The Prompt window will prompt you to select the two nodes to be linked. (See Figure 37.7.) Do this, and the link will be added.

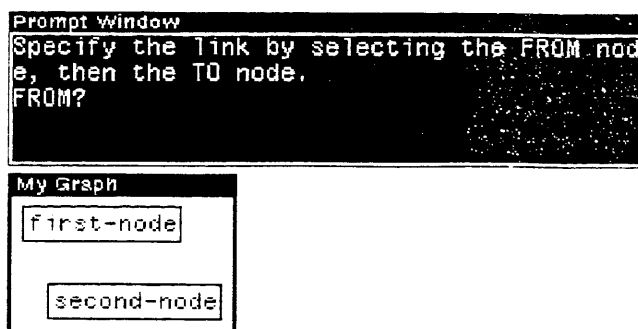
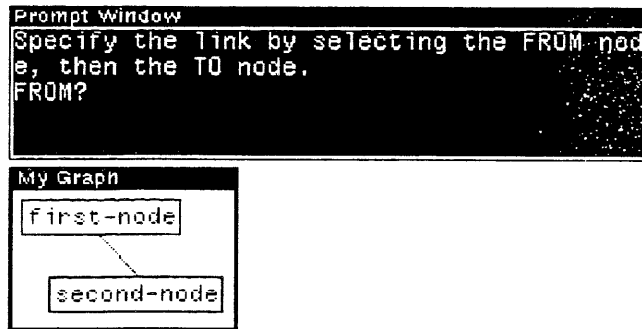


Figure 37.7. The Prompt window will prompt you to select the two nodes to link.



**Delete A Link**

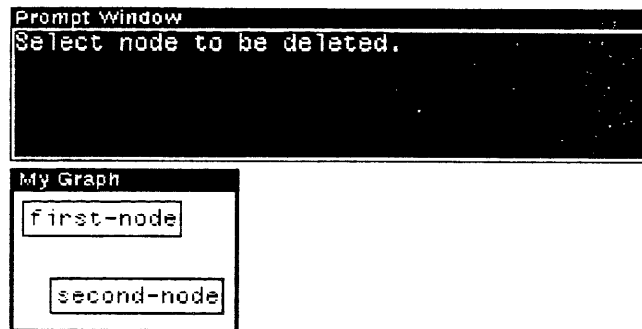
Select **Delete Link** from the graph editing menu. The Prompt window will prompt you to select the two nodes that should no longer be linked. (See Figure 37.8.) Do this, and the link will be deleted.



**Figure 37.8.** The Prompt window will prompt you to select the two nodes that should no longer be linked.

**Delete A Node**

Select **Delete Node** from the graph editing menu. The Prompt window will prompt you to select the node to be deleted. (See Figure 37.9.) Do this, and the node will be deleted.



**Figure 37.9.** The prompt to delete a node.

**Moving a Node**

Select "Delete Node" from the graph editing menu. Choose a node pointing to the it with the mouse cursor, and pressing and holding the left mouse button. When you move the mouse cursor, the node will be dragged along. When the node is at the new position, release the mouse button to deposit the node.

The commands in this menu are easy to learn. Experiment with them!

---

## 37.2 Making a Graph from a List

---

Typically, a graph is used to display one of your program's data structures. Here is how that is done.

**LAYOUTSEXP** takes a list and returns a GRAPH record. The syntax of the function is

*(LAYOUTSEXP sexpr format boxing font motherd  
personald familyd)*

For example:

```
(SETQ ANIMAL.TREE '(ANIMAL (MAMMAL DOG CAT) BIRD FISH))
(SETQ ANIMAL.GRAPH
```

```
(LAYOUTSEXP ANIMAL.TREE 'HORIZONTAL))
(SHOWGRAPH ANIMAL.GRAPH "My Graph" NIL NIL NIL T)
```

This is how Figure 37.1 was produced.

---

### 37.3 Incorporating Grapher into Your Program

---

The Grapher is designed to be built into other programs. It can call functions when, for example, a mouse button is clicked on a node. The function **SHOWGRAPH** does this:

```
(SHOWGRAPH graph window leftbuttonfn middlebuttonfn
topjustifyflg alloweditflg copybuttoneventfn)
```

For example, the third argument to **SHOWGRAPH**, *leftbuttonfn*, is a function that is called when the left mouse button is pressed in the graph window. Try this:

```
(DEFINEQ (MY.LEFT.BUTTON.FUNCTION
          (THE.GRAPHNODE THE.GRAPH.WINDOW)
          (INSPECT THE.GRAPHNODE)))

(SHOWGRAPH FAMILY.GRAPH "Inspectable family"
 (FUNCTION MY.LEFT.BUTTON.FUNCTION)
 NIL NIL T)
```

In the example above, **MY.LEFT.BUTTON.FUNCTION** simply calls the inspector. Note that the function should be written assuming it will be passed a graphnode and the window that holds the graph. Try adding a function of your own.

---

### 37.4 More of Grapher

---

Some other Library packages make use of the Grapher. (Note: Grapher needs to be loaded with the packages to use these functions.)

- **MASTERSCOPE**: The Browser package modifies the Masterscope command, `. SHOW PATHS`, so that its output is displayed as a graph (using Grapher) instead of simply printed.
- **GRAPHZOOM**: allows a graph to be redisplayed larger or smaller automatically.

[This page intentionally left blank]

## 38. VIRTUAL KEYBOARDS, AND THE KEYBOARD EDITOR

---

There are 2 library packages, the Virtual Keyboards Package and the Keyboard Editor, that make it possible to change the configuration of your keyboard. You can change the character of your keyboard to the configuration of another keyboard provided by the package, or you create an keyboard configuration that is uniquely yours.

If you don't want to change your keyboard, you can still simulate having a different keyboard by using your mouse to select from a keyboard displayed on the screen. Whichever way you choose, they will work with any software package that requires keyboard input. The following subsections demonstrate how to use the Virtual Keyboards package and the Keyboard Editor package.

---

### 38.1 Using the Virtual Keyboards Package

---

Load the files VIRTUALKEYBOARDS.DCOM and KEYBOARDEDITOR.DCOM. (See Section 8.6, Page 8.4 for how to load a file.) The right button background menu will display another item called Keyboard. At the right of the Keyboard item, you will notice a grey triangle. Hold down the mouse button, with the Keyboard item blackened, and move the mouse to "follow the arrow". Another menu (a submenu) will appear. Using this submenu (and some of the submenu's of those items) is an easy and effective way to use this library package, and is what will be done in this chapter.

To display a keyboard, choose "Display only" from the Keyboard submenu. A menu of provided virtual keyboards will appear. Right now, this includes: Default, European Logic, Math, Office, Dvorak, Greek, Italian, Spanish, French, German, and Standard Russian. Figure 38.1 shows an example of the Dvorak keyboard, displayed by using this menu option.

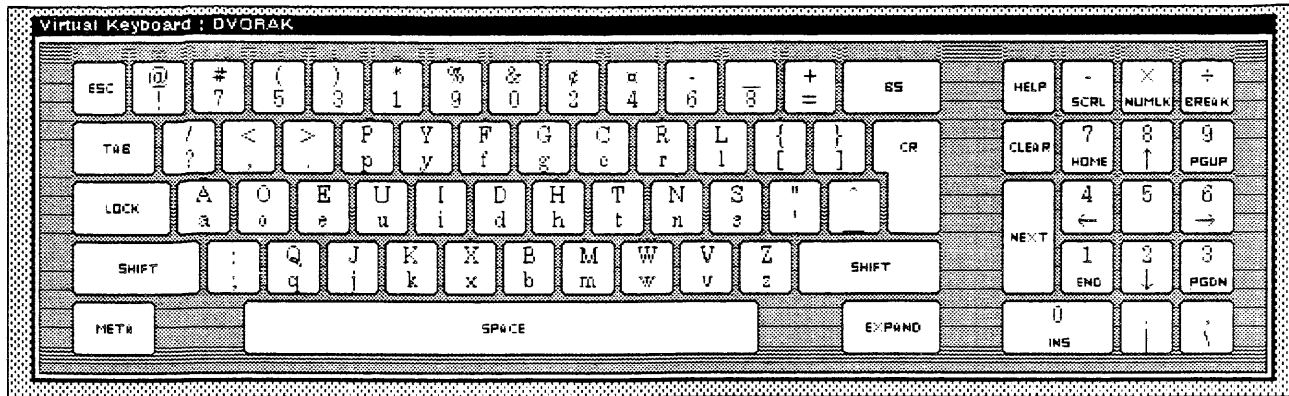


Figure 38.1. A Dvorak Keyboard displayed using the Virtual Keyboards Package

To use a key, point to it with the mousecursor , and click the left mouse button. For upper case letters, hold down the shift key on the keyboard while you choose the key on the screen with the mouse.

To replace the character set of your hardware keyboard, choose either "Switch keyboard" or "Switch and display" from the submenu of Keyboard on the background menu. Both of these options replace the keyboard's character set, but "Switch and display" will also show the keyboard on the screen, so that it can be referenced for key placement. With either choice you must choose the keyboard from a menu of the available keyboards.

## 38.2 Using the Keyboard Editor

If none of the keyboards is exactly what you want, you can use the keyboard editor to create the keyboard you need. To do this, notice that the "Edit" item of the submenu of the background menu item "Keyboard" ALSO has a submenu! This menu's items are:

- New keyboard, default initial
- New keyboard, other initial
- Existing Keyboard

Choosing "Edit", without looking at its submenu, is the same as choosing "New keyboard, default initial". Unless you choose "Existing keyboard" you will be prompted for a name for your new keyboard (existing keyboards have already been assigned names).

As an example, choose "Edit" from the submenu of "Keyboard". The keyboard editor will appear on your screen, as in figure Figure 38.2.

CharSet	Stop	Quit	Define												
SHIFT LOCKDOWN	CTRL LOCKUP	META EVENT	LOCK												
Character set 0															
0	20	40	60	100	120	140	160	200	220	240	260	300	320	340	360
0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17

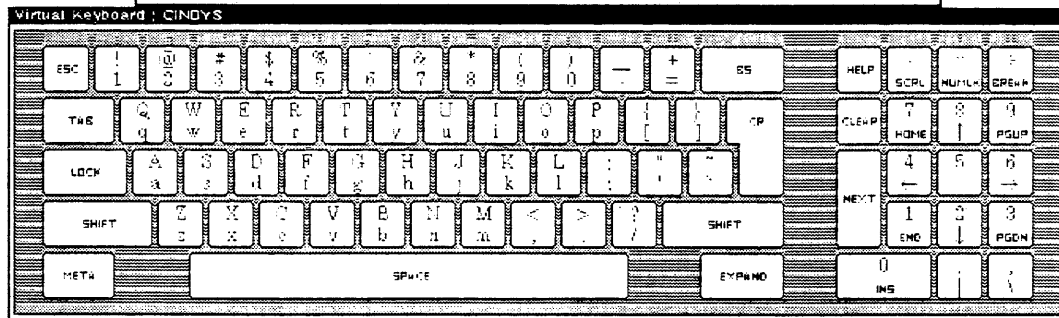
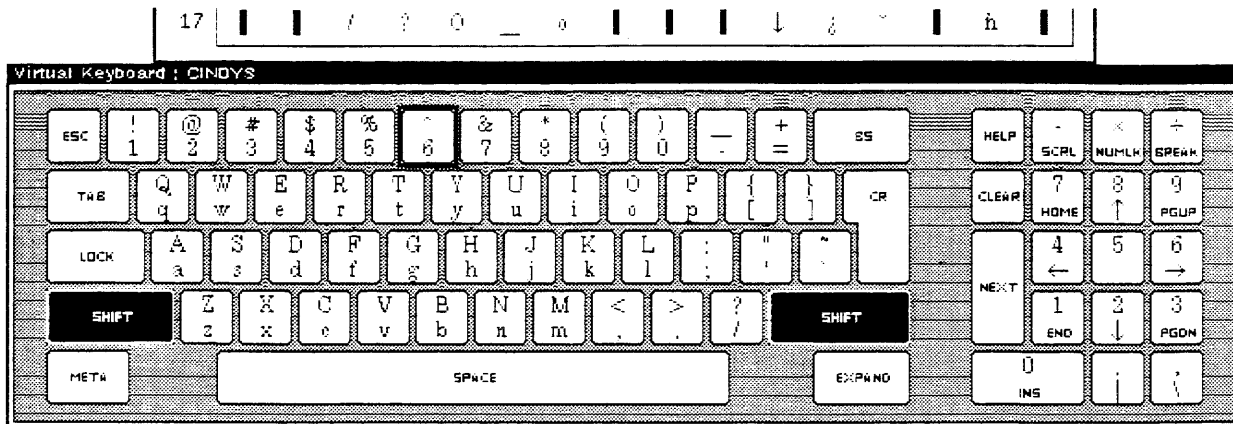


Figure 38.2. The Keyboard Editor

To change a key, select it from the keyboard with the mouse. It will be shown with a heavy black border.

To change the upper case character, hold the shift key down while you select a key. For this example, hold down the shift key, and select the number 6. The keyboard should look like this:



**Figure 38.3.** The Keyboard after the shifted 6 key is selected.

After you have chosen the key, you want to change, choose a replacement character from the character menu above. (If you want a character that is not shown on the character menu, choose "CharSet" from command menu at the top. You will be given a menu of the character sets available. Choose one of these to see other characters.). The character you chose will appear on the keyboard.

When you are done, choose either:

- Define to add this keyboard to the menu of known keyboards.
- Quit to exit the editor, and save the changes on the virtual Keyboard. If you are editing an existing keyboard, the definition of that keyboard will be changed. If you are editing a new keyboard, the name you supplied for the keyboard will be added to the list of known keyboards. You will be able to use the commands, such as "Display Only", with your new keyboard.
- Stop to exit the editor without changing the existing keyboard.

Each window has a property, **ICONW**, that determines what icon is used when you shrink the window. **ICONW** is a function that can produce an interesting icon. It is usually called from the **ICONFN** of a window and its syntax is:

**(ICONW image mask)**

See Section 27.1.2, Page 27.2 for how to include a function in a window's property list.

Every icon is made from two bitmaps, an *image* and a *mask*. The mask allows the background to show through some parts of the image bitmap so that the image need not appear to be rectangular.

As an example, create a bitmap called **desk**, by typing the function

**(SETQ DESK (EDITBM))**

The image for the example bitmap **DESK** looks like this:



Figure 39.1. The DESK bitmap

Now create the mask. Make the mask black everywhere that you want to "mask" the background (everywhere the background should not show through). To do this, type:

**(SETQ DESKMASK (EDITBM))**

The mask for the example bitmap **DESKMASK** looks like this:



Figure 39.2. The DESKMASK bitmap

Now, to see the icon, type

**(ICONW DESK DESKMASK)**

into the Interlisp-D Executive Window. The resulting icon using **DESK** and **DESKMASK** above looks like this:

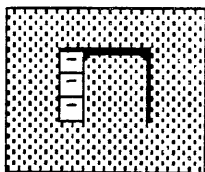


Figure 39.3. The ICON that resulted from executing **ICONW** with the **DESK** and **DESKMASK** bitmaps.



The *Lisp Library Packages Manual* describes some images, with their associated masks, provided in the file STOCKICONS. They include FOLDER and FOLDERMASK, which make a "file folder" icon, PAPERICON and PAPERICONMASK, which make an icon that looks like a piece of paper with the corner folded over, and FILEDRAWER and FILESDRAWERMASK that make an icon that looks like the front of a filedrawer.

There is also a function to produce icons that include text. See the *Lisp Library Packages Manual* for more information.

Teleraid is a Library package. It has two purposes. One purpose is to look at the virtual memory of another machine, when both machines are on a network, and the other is to look inside a SYSOUT file.

The file, TELERAID.DCOM, needs to be loaded to use Teleraid. Type (**FILESLOAD TELERAID**).

When your 1108 maintenance panel does not say 1108 or 1109, or your 1186 mouse cursor has changed to a number, you first need to check your *User's Guide* to see what to do for your particular problem. Usually the fix involves entering Teleraid. Do this either by pressing the UNDO key, or, if that does not work, press Control-Shift-Delete (all three at once). The mouse cursor should change to a cursor that says "Teleraid". Follow the instructions in your *User's Guide*.

When your 1108 is on a network, you can use teleraid to debug an 1108 whose maintenance panel does not say 1108 or 1109. First press the halted machine's UNDO key. This changes the mouse cursor into "Teleraid", and begins running the teleraid server.

Run the function **TELERAID**, with the host name or pup address of the machine running the Teleraid server as an argument. The following teleraid commands are useful:

- L shows the stack.
- F *num* shows the frame number *num*.
- Control-J shows the next frame of the stack.
- ↑ shows the previous stack frame.
- D *atom* shows the function definition of *atom*. Returns 0 if there is none.
- A *atom* shows the top level value of *atom*.
- P *atom* shows the property list of *atom*.
- U displays the screen's bitmap of the machine running the teleraid server.
- Q quits teleraid without affecting the machine being debugged.
- Control-N causes the machine being debugged to resume execution. Do Not use this unless you are sure that the problem has been solved.

There are many more commands available to you. See the *Lisp Library Packages Manual* for more information.

[This page intentionally left blank]

---

## 41.1 Naming Variables and Records

---

You will find times when one environment simultaneously hosts a number of different programs. Running a demo of several programs, or reloading the entire Interlisp-D environment from floppies when it contains several different programs, are two examples that could, if you aren't careful, provide a few problems. Here are a few tips on how to prevent problems:

- If you change the value of a system variable, **MENUHELDDWAIT** for example, or connect to a directory other than `{DSK}<LISPPFILES>`, write a function to reset the variable or directory to its original value. Run this function when you are finished working. This is especially important if you change any of the system menus.

- Don't redefine Interlisp-D functions or CLISP words.

Remember, if you reset an atom's value or function definition at the top level (in the Interlisp-D Executive Window), the message (*Some.Crucial.Function.Or.Variable redefined*), appears. If this is not what you wanted, type **UNDO** immediately!

If, however, you reset the value or function definition of an atom inside your program, a warning message will not be printed.

- Make the atom names in your programs as unique as possible. To do this without filling your program with unreadable names that noone, including you, can remember, prefix your variable names with the initials of your program. Even then, check to see that they are not already being used with the function **BOUNDP**. For example, type:

```
(BOUNDP 'BackgroundMenu)
```

This atom is bound to the menu that appears when you press the left mouse button when the mouse cursor is not in any window. **BOUNDP** returns T. **BOUNDP** returns **NIL** if its argument does not currently have a value.

- Make your function names as unique as possible. Once again, prefixing function names with the initials of your program can be helpful in making them unique, but even so, check to see that they are not already being used. **GETD** is the Interlisp-D function that returns the function definition of an atom, if it has one. If an atom has no function definition, **GETD** returns **NIL**. For example, type:

```
(GETD 'CAR)
```

A non-NIL value is returned. The atom CAR already has a function definition.

- Use complete record field names in record FETCHes and REPLACEs when your code is not compiled. A Complete record field name is a list consisting of the record declaration name and the field name. Consider the following example:

```
(RECORD NAME (FIRST LAST))
(SETQ MyName (create NAME FIRST ← 'John LAST ← 'Smith))
(FETCH (NAME FIRST) OF MyName)
```

- Avoid reusing names that are field names of Interlisp-D System records. A few examples of system records follow. Do not reuse these names.

```
(RECORD REGION (LEFT BOTTOM WIDTH HEIGHT))
(RECORD POSITION (XCOORD YCOORD))
(RECORD IMAGEOBJ (- BITMAP -))
```

- When you select a record name and field names for a new record, check to see whether those names have already been used.

Call the function **RECLook**, with your record name as an argument, in the Interlisp-D Executive Window. (See Figure 41.1.) If your record name is already a record, the record definition will be returned; otherwise the function will return **NIL**.

```
Interlisp-D Executive Window
50←(RECLook 'POSITION)
(RECORD
 POSITION
 (XCOORD . YCOORD)
 [TYPE? (AND (LISTP DATUM)
              (NUMBERP (CAR DATUM))
              (NUMBERP (CDR DATUM))
              (SYSTEM))
51←(RECLook 'NewPos)
NIL
52←
```

**Figure 41.1.** **RECLook** returns the record definition if its argument is already declared as a record, **NIL** otherwise.

Call the function **FIELDLOOK** with your new field name in the Interlisp-D Executive Window. (See Figure 41.2.) If your field name is already a field name in another record, the record definition will be returned; otherwise the function will return **NIL**.

```

Interlisp-D Executive Window
54←(FIELDLOOK 'XCOORD)
((RECORD
  POSITION
  (XCOORD . YCOORD)
  [TYPE? (AND (LISTP DATUM)
              (NUMBERP (CAR DATUM))
              (NUMBERP (CDR DATUM))
              (SYSTEM)))
  55←(FIELDLOOK 'XPos)
NIL
56←

```

**Figure 41.2.** **FIELDLOOK** returns the record definition if its argument is already the field of a record, **NIL** otherwise.

---

## 41.2 Some Space and Time Considerations

---

In order for your program to run at maximum speed, you must efficiently use the space available on the system. The following section points out areas that you may not know are wasting valuable space, and tips on how to prevent this waste.

Often programs are written so that new data structures are created each time the program is run. This is wasteful. Write your programs so that they only create new variables and other data structures conditionally. If a structure has already been created, use it instead of creating a new one.

Some time and space can be saved by changing your **RECORD** and **TYPERECORD** declarations to **DATATYPE**. **DATATYPE** is used the same way as the functions **RECORD** and **TYPERECORD**. (See Chapter 24.) In addition, the same **FETCH** and **REPLACE** commands can be used with the data structure **DATATYPE** creates. The difference is that the data structure **DATATYPE** creates cannot be treated as a list the way **RECORDs** and **TYPERECORDs** can.

---

### 41.2.1 Global Variables

---

Once defined, global variables remain until Interlisp-D is reloaded. Avoid using global variables if at all possible!

One specific problem arises when programs use the function **GENSYM**. In program development, many atoms are created that may no longer be useful. Hints:

- Use
  - (DELDEF *atomname* 'PROP)
  - to delete property lists, and
  - (DELDEF *atomname* 'VARS)

to have the atom act like it is not defined.

These not only remove the definition from memory, but also change the appropriate `fileCOMS` that the deleted object was associated with so that the file package will not attempt to save the object (function, variable, record definition, and so forth) the next time the file is made. Just doing something like

```
(SETQ (arg atomname) 'NOBIND)
```

looks like it will have the same effect as the second `DELDEF` above, but the `SETQ` doesn't update the file package.

- If you are generating atom names with `GENSYM`, try to keep a list of the atom names that are no longer needed. Reuse these atom names, before generating new ones. There is a (fairly large) maximum to the number of atoms you can have, but things slow down considerably when you create lots of atoms.
- When possible, use a data structure such as a list or an array, instead of many individual atoms. Such a structure has only one pointer to it. Once this pointer is removed, the whole structure will be garbage collected and space reclaimed.

## 41.2.2 Circular Lists

---

If your program is creating circular lists, a lot of space may be wasted. (Note that many cross linked data structures end up having circularities.) Hints when using circular lists:

- Write a function to remove pointers that make lists circular when you are through with the circular list.
- If you are working with circular lists of windows, bind your main window to a unique global variable. Write window creation conditionally so that if the binding of that variable is already a window, use it, and only create a new window if that variable is unbound or `NIL`.

Here is an example that illustrates the problem. When several auxiliary windows are built, pointers to these windows are usually kept on the main window's property list. Each auxiliary window also typically keeps a pointer to the main window on its property list. If the top level function creates windows rather than reusing existing ones, there will be many lists of useless windows cluttering the work space. Or, if such a main window is closed and will not be used again, you will have to break the links by deleting the relevant properties from both the main window and all of the auxiliary windows first. This is usually done by putting a special `CLOSEFN` on the main window and all of its auxiliary windows.

## 41.2.3 When You Run Out Of Space

---

Typically, if you generate a lot of structures that won't get garbage collected, you will eventually run out of space. The important part is being able to track down those structures and

the code that generates them in order to become more space efficient.

The Lisp Library Package GCHAX.DCOM can be used to track down pointers to data structures. The basic idea is that GCHAX will return the number of references to a particular data structure.

A special function exists that allows you to get a little extra space so that you can try to save your work when you get toward the edge (usually noted by a message indicating that you should save your work and `sysin` a fresh Lisp). The **GAINSPACE** function allows you to delete non-essential data structures. To use it, type:

**(GAINSPACE)**

into the Interlisp-D Executive Window. Answer "N" to all questions except the following.

- Delete edit history
- Delete history list.
- Delete values of old variables.
- Delete your MASTERSCOPE datadase
- Delete information for undoing your greeting.

Save your work and reload Lisp as soon as possible.



[This page intentionally left blank]

## 42. SIMPLE INTERACTIONS WITH THE CURSOR, A BITMAP, AND A WINDOW

---

The purpose of this chapter is to show you how to build a moderately tricky interactive interface with the various Interlisp-D display facilities. In particular how to move a large bitmap (larger than 16 x 16 pixels) around inside a window. To do this, you will change the **CURSORINFN** and **CURSOROUTFN** properties of the window. If you would also like to then set the bitmap in place in the window, you must reset the **BUTTONEVENTFN**. This chapter explains how to create the mobile bitmap.

---

### 42.1 An Example Function Using GETMOUSESTATE

---

One function that you will use to "trace the cursor" (have a bitmap follow the cursor around in a window) is **GETMOUSESTATE**. This function finds the current state of the mouse, and resets global system variables, such as **LASTMOUSEX** and **LASTMOUSEY**.

As an example of how this function works, create a window by typing

```
(SETQ EXAMPLE.WINDOW (CREATEW))
```

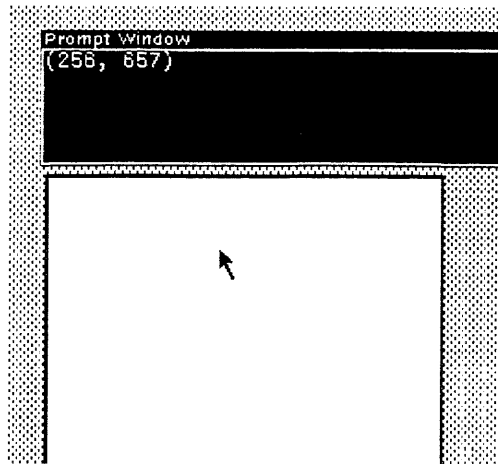
into the Interlisp-D Executive window, and sweeping out a window. Now, type in the function

```
(DEFINEQ (PRINTCOORDS (W)
  (PROMPTPRINT "(" LASTMOUSEX ", " LASTMOUSEY ")")
  (BLOCK)
  (GETMOUSESTATE)))
```

This function calls **GETMOUSESTATE** and then prints the new values of **LASTMOUSEX** and **LASTMOUSEY** in the promptwindow. To use it, type

```
(WINDOWPROP EXAMPLE.WINDOW 'CURSORMOVEDFN 'PRINTCOORDS)
```

The window property **CURSORMOVEDFN**, used in this example, will evaluate the function **PRINTCOORDS** each time the cursor is moved when it is inside the window. The position coordinates of the mouse cursor will appear in the prompt window. (See Figure 42.1.)



**Figure 42.1.** The current position coordinates of the mouse cursor are shown in the prompt window

## 42.2 Advising GETMOUSESTATE

For the bitmap to follow the moving mouse cursor, the function `GETMOUSESTATE` is advised. When you advise a function, you can add new commands to the function without knowing how it is actually implemented. The syntax for advise is

`(ADVISE fn when where what)`

*fn* is the name of the function to be augmented.

*when* and *where* are optional arguments. *when* specifies whether the change should be made before, after, or around the body of the function. The values expected are `BEFORE`, `AFTER`, or `AROUND`.

*what* specifies the additional code.

In the example, the additional code, *what*, moves the bitmap to the position of the mouse cursor. The function `GETMOUSESTATE` will be `ADVISED` when the mouse moves into the window. This will cause the bitmap to follow the mouse cursor. `ADVISE` will be undone when the mouse leaves the window or when a mouse button is pushed. The `ADVISEing` will be done and undone by changing the `CURSORINFN`, `CURSOROUTFN`, and `BUTTONEVENTFN` for the window.

## 42.3 Changing the Cursor

One last part of the example, to give the impression that a bitmap is dragged around a window, the original cursor should disappear. Try typing:

```
(CURSOR (CURSORCREATE (BITMAPCREATE 1 1) 1 1])
```

into the Interlisp-D Executive Window. This causes the original cursor to disappear. It reappears when you type

```
(CURSOR T)
```

When the cursor is invisible, and the bitmap moves as the cursor moves, the illusion is given that the bitmap is dragged around the window.

## 42.4 Functions for "Tracing the cursor"

To actually have a bitmap trace (follow) the cursor, the environment must be set up so that when the cursor enters the tracing region the trace is turned on, and when the cursor leaves the tracing region the trace is turned off. The function `Establish/Trace/Data` will do this. Type it in as it appears (note: including the comments will help you remember what the function does later).

```
(DEFINEQ (Establish/Trace/Data
  [LAMBDA (wnd tracebitmap cursor/rightoffset cursor/heightoffset GCGAGP)

    (* * This function is called to establish the data to trace
    the desired bitmap. "wnd" is the window in which the tracing
    is to take place. "tracebitmap" is the tracing bitmap,
    "cursor/rightoffset" and "cursor/heightoffset" are integers
    which determine the hotspot of the tracing bitmap.
    As "cursor/heightoffset" and "cursor/rightoffset" increase
    the cursor hotspot moves up and to the right.
    If GCGAGP is non-NIL, GCGAG will be disabled.)

  (PROG NIL
    (if (OR (NULL wnd)
            (NULL tracebitmap))
        then (PLAYTUNE (LIST (CONS 1000 4000)))
            (RETURN))
        (if GCGAGP
            then (GCGAG))

        (* * Create a blank cursor.)

        (SETQ *BLANKCURSOR*(BITMAPCREATE 16 16))
        (SETQ *BLANKTRACECURSOR*(CURSORCREATE *BLANKCURSOR*))

        (* * Set the CURSOR IN and OUT FNS for wnd to the
        following:)

        (WINDOWPROP wnd (QUOTE CURSORINFN)
                     (FUNCTION SETUP/TRACE))
        (WINDOWPROP wnd (QUOTE CURSOROUTFN)
                     (FUNCTION UNTRACE/CURSOR))

        (* * To allow the bitmap to be set down in the window by
        pressing a mouse button, include this line.
        Otherwise, it is not needed)

        (WINDOWPROP wnd (QUOTE BUTTONEVENTFN)
                     (FUNCTION PLACE/BITMAP/IN/WINDOW))

        (* * Set up Global Variables for the tracing operation)

        (SETQ *TRACEBITMAP* tracebitmap)
        (SETQ *RIGHTTRACE/OFFSET*(OR cursor/rightoffset 0))
        (SETQ *HEIGHTTRACE/OFFSET*(OR cursor/heightoffset 0))
        (SETQ *OLDBITMAPPOSITION*(BITMAPCREATE (BITMAPWIDTH tracebitmap)
                                               (BITMAPHEIGHT tracebitmap)))
        (SETQ *TRACEWINDOW* wnd]))
```

When the function `Establish/Trace/Data` is called, the functions `SETUP/TRACE` and `UNTRACE/CURSOR` will be installed as the values of the window's `WINDOWPROPS`, and will be used to turn the trace on and off. Those functions should be typed in, then:

```
(DEFINEQ (SETUP/TRACE
[LAMBDA (wnd)
  (* * This function is wnd's CURSORINFN.
  It simply resets the last trace position and the current
  tracing region. It also readvises GETMOUSESTATE to perform
  the trace function after each call.)

  (if *TRACEBITMAP*
    then (SETQ *LAST-TRACE-XPOS* -2000)
          (SETQ *LAST-TRACE-YPOS* -2000)
          (SETQ *WNDREGION* (WINDOWPROP wnd (QUOTE REGION)))
          (WINDOWPROP wnd (QUOTE TRACING)
                      T))

    (* * make the cursor disappear)

    (CURSOR *BLANKTRACECURSOR*)
    (ADVISE (QUOTE GETMOUSESTATE)
            (QUOTE AFTER)
            NIL
            (QUOTE (TRACE/CURSOR)))
```

```
(DEFINEQ (UNTRACE/CURSOR
[LAMBDA (wnd)
  (* * This function is wnd's CURSOROUTFN.
  The function first checks if the cursor is currently being
  traced; if so, it replaces the tracing bitmap with what is
  under it and then turns tracing off by unadvising
  GETMOUSESTATE and setting the TRACING window property of
  *TRACEWINDOW* TO NIL.)

  (if (WINDOWPROP *TRACEWINDOW*(QUOTE TRACING))
    then (BITBLT *OLDBITMAPPOSITION* 0 0 (SCREENBITMAP)
              (IPLUS (CAR *WNDREGION*)*LAST-TRACE-XPOS*)
              (IPLUS (CADR *WNDREGION*)*LAST-TRACE-YPOS*))
          (WINDOWPROP *TRACEWINDOW*(QUOTE TRACING)
                      NIL))

    (* * replace the original cursor shape)

  (CURSOR T)

  (* * unadvise GETMOUSESTATE)

  (UNADVISE (QUOTE GETMOUSESTATE]))
```

The function `SETUP/TRACE` has a helper function that you must also type in. It is `TRACE/CURSOR`:

```
(DEFINEQ (TRACE/CURSOR
[LAMBDA NIL
  (* * This function does the actual BITBLTing of the tracing
  bitmap. This function is called after a GETMOUSESTATE, while
  tracing.)

  (PROG ((xpos (IDIFFERENCE (LASTMOUSEX *TRACEWINDOW*)*RIGHTTRACE/OFFSET*))
         (ypos (IDIFFERENCE (LASTMOUSEY *TRACEWINDOW*)*HEIGHTTRACE/OFFSET*)))

    (* * If there is an error in the function, press the right
    button to unadvise the function. This will keep the machine
    from locking up.)

    (if (LASTMOUSESTATE RIGHT)
      then (UNADVISE (QUOTE GETMOUSESTATE)))
    (if (AND (NEQ xpos *LAST-TRACE-XPOS*)
             (NEQ ypos *LAST-TRACE-YPOS*))
      then

        (* * Restore what was under the old position of the trace
        bitmap)

        (BITBLT *OLDBITMAPPOSITION* 0 0 (SCREENBITMAP)
```

```

(IPLUS (CAR *WNDREGION*)*LAST-TRACE-XPOS*)
(IPLUS (CADR *WNDREGION*)*LAST-TRACE-YPOS*))

(* * Save what will be under the position of the new trace
bitmap)

(BITBLT (SCREENBITMAP)
(IPLUS (CAR *WNDREGION*)
xpos)
(IPLUS (CADR *WNDREGION*)
ypos)*OLDBITMAPPOSITION* 0 0)

(* * BITBLT the trace bitmap onto the new position of the
mouse.)

(BITBLT *TRACEBITMAP* 0 0 (SCREENBITMAP)
(IPLUS (CAR *WNDREGION*)
xpos)
(IPLUS (CADR *WNDREGION*)
ypos)
NIL NIL (QUOTE INPUT)
(QUOTE PAINT))

(* * Save the current position as the last trace position.)

(SETQ *LAST-TRACE-XPOS* xpos)
(SETQ *LAST-TRACE-YPOS* ypos)))

```

The helper function for UNTRACE/CURSOR, called UNDO/TRACE/DATA, must also be added to the environment:

```

(DEFINEQ (UNDO/TRACE/DATA
[LAMBDA NIL

(* * The purpose of this function is to turn tracing off and
to free up the global variables used to trace the bitmap, so
that they can be garbage collected.)

(* * Check if the cursor is currently being traced.
If so, turn it off.)

(UNTRACE/CURSOR)
(WINDOWPROP *TRACEWINDOW*(QUOTE CURSORINFN)
NIL)
(WINDOWPROP *TRACEWINDOW*(QUOTE CURSOROUTFN)
NIL)
(SETQ *TRACEBITMAP* NIL)
(SETQ *RIGHTTRACE/OFFSET* NIL)
(SETQ *HEIGHTTRACE/OFFSET* NIL)
(SETQ *OLDBITMAPPOSITION* NIL)
(SETQ *TRACEWINDOW* NIL)

(* * Turn GCGAG on)

(GCGAG T]))

```

Finally, if you included the WINDOWPROP to allow the user to place the bitmap in the window by pressing a mouse button, you must also type this function:

```

(DEFINEQ (PLACE/BITMAP/IN/WINDOW
[LAMBDA (wnd)
(UNADVISE (GETMOUSESTATE))
(BITBLT *TRACEBITMAP* 0 0 (SCREENBITMAP)
(IPLUS (CAR *WNDREGION*)
xpos)
(IPLUS (CADR *WNDREGION*)
ypos)
NIL NIL (QUOTE INPUT)
(QUOTE PAINT)]

```

That's all the functions!

## 42.5 Running the Functions

---

To run the functions you just typed in, first set a variable to a window by typing something like

```
(SETQ EXAMPLE.WINDOW (CREATEW))
```

into the Interlisp-D Executive window, and sweeping out a new window. Now, set a variable to a bitmap, by typing, perhaps,

```
(SETQ EXAMPLE.BTM (EDITBM))
```

Type

```
(Establish/Trace/Data EXAMPLE.WINDOW EXAMPLE.BTM)
```

When you move the cursor into the window, the cursor will drag the bitmap.

(Note: If you want to be able to make menu selections while tracing the cursor, make sure that the hotspot of the cursor is set to the extreme right of the bitmap. Otherwise, the menu will be destroyed by the BITBLTs of the trace functions.)

To stop tracing, either

- move the mouse cursor out of the window;
- press the right mouse button;
- call the function **UNTRACE/CURSOR**.

# 43. GLOSSARY OF GLOBAL SYSTEM VARIABLES

---

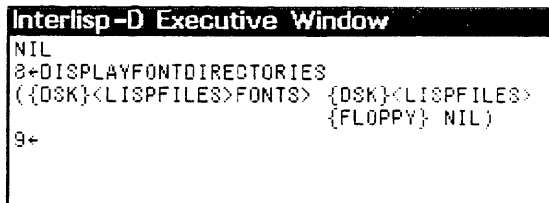
As you can tell by now, there are many system variables in Interlisp that are useful to know the meaning of. The following sections gathers many of the important variables together into groups relating to directory searching, system flags, history lists, system menus, windows, and, of course, the catchall miscellaneous category.

---

## 43.1 Directories

---

**DISPLAYFONTDIRECTORIES** Its value is a list of directories to search for the bitmap files for display fonts. Usually, it contains the "FONT" directory where you copied the bitmap files, (See Chapter 31.), the device {FLOPPY}, and the current connected directory. The current connected directory is specified by the atom NIL. Here is an example value of **DISPLAYFONTDIRECTORIES**:



```
Interlisp-D Executive Window
NIL
8+DISPLAYFONTDIRECTORIES
({{DSK}<LISPFILES>FONTS} {DSK}<LISPFILES>
                               {FLOPPY} NIL)
9+
```

**Figure 43.1.** A value for the atom **DISPLAYFONTDIRECTORIES**. When looking for a **.DISPLAYFONT** file, the system will check the **FONTS** directory on the hard disk, then the top level directory on the hard disk, then the floppy, then the current connected directory.

**INTERPRESSFONTDIRECTORIES** Is set to a list of directories to search for the font width files for Interpress fonts.

**PRESSFONTWIDTHFILES** The value of this variable is a list of files, not directories, to search for widths for Press fonts. Press font widths are usually large files named **FONTS.WIDTHS**.

**DIRECTORIES** This variable is bound to a list of the directories you will be using. (See Figure 43.2.) The system uses this variable when it is trying to find a file to load - it checks each directory in the list, until the file is found. NIL in list means to check the current connected directory.



```

Interlisp-D Executive Window
NIL
59-DIRECTORIES
(NIL {OSK}<LISPPFILES>
      {OSK}<LISPPFILES>LIBRARY>
      {OSK}<LISPPFILES>LOOPS> {FLOPPY})
60+

```

Figure 43.2. The value of **DIRECTORIES**.

**LISPUSERSDIRECTORIES** Its value is a list of directories to search for library package files.

## 43.2 Flags

<b>DWIMIFYCOMPFLG</b>	<p>This flag, if set to <b>T</b>, will cause all expressions to be completely dwimified before the expression is compiled. (See Section 13.2, Page 13.2.) In this state, when the system does not recognize a function of keyword, it will compare the word to a system maintained list to determine whether the word is a macro, CLISP word, or misspelled user-defined variable.</p> <p>An example of dwimifying before compilation is to convert an <b>IF</b> call to a <b>COND</b>. before they are compiled. Undwimified expressions can cause inaccurate compilation. This flag is set by the system to <b>NIL</b>. Normally, you want this set to <b>T</b>. For more information on DWIM, refer to the <i>Interlisp-D Reference Manual</i>, Volume 2, Chapter 20.</p>
<b>SYSPRETTYFLG</b>	When set to <b>T</b> , all lists returned to the Interlisp-D Executive window are pretty printed. This flag is originally set by the system to <b>NIL</b> .
<b>CLISPIFTRANFLG</b>	When set to <b>T</b> , keep the <b>IF</b> expression, rather than the <b>COND</b> translation in your code.
<b>PRETTYTABFLG</b>	When set to <b>T</b> , the pretty printer puts out a tab character rather than several spaces to try to make code align. If <b>NIL</b> , it uses space characters instead.
<b>FONTCHANGEFLG</b>	If <b>NIL</b> , then when prettyprinting no font changes will happen (e.g. a smaller font for comments, bold for clisp words, and so forth). The default is the atom <b>ALL</b> , so different fonts are used where appropriate.
<b>DEditLinger</b>	Its initial value is <b>T</b> , which means that the <b>DEdit</b> window stays open after you exit <b>DEdit</b> . Set it to <b>NIL</b> if you want the <b>DEdit</b> window will be closed when you exit <b>DEdit</b> .
<b>PROMPT#FLG</b>	Its initial value is <b>T</b> , so the history list number is printed before the "←" prompt.
<b>AUTOBACKTRACEFLG</b>	<p>There are many possible values for this variable. They affect when the back trace window appears with the break window, and how much detail is included in it. The values of this variable include:</p> <ul style="list-style-type: none"> <li>• <b>NIL</b>, its initial value. The back trace window is not brought up when an error is generated, until you open it yourself.</li> </ul>

- **T**, which means that the back trace, BT, window is opened for error breaks.
- **BT!** brings up a back trace window with more detail, BT!, window for error breaks.
- **ALWAYS** brings up a back trace, BT, window for both error breaks, and breaks caused by calling the function **BREAK**.
- **ALWAYS!** brings up a back trace window with more detail, BT!, for both error breaks, and breaks caused by calling the function **BREAK**.

**NOSPELLFLG** is initially bound to **NIL**, so that DWIM tries to correct all spelling errors, whether they are in a form you just typed in or within a function being run. If the variable is **T**, then no spelling correction is performed. This variable is automatically reset to **T** when you are compiling a file. If it has some other non-**NIL** value, then spelling correction is only performed on type-in.

---

### 43.3 History Lists

---

**LISPXHISTORY** Originally set to the list (NIL 0 30 100), with the following argument interpretation. The **NIL** is the list (implemented as a circular queue) to which the top level commands append, 0 is the current prompt number, 30 is the maximum length of the history list, and 100 is the highest number used as a prompt. This is a system maintained list used by the Programmers Assistant commands **REDO**, **UNDO**, **FIX**, and **??** use to retrieve past function calls.

To delete the history list, just reset the variable **LISPXHISTORY** to its original value, (NIL 0 30 100).

Setting this variable to **NIL**, disables all the Programmers Assistant features.

**EDITHISTORY** This is also set to (NIL 0 30 100) and has the same description as **LISPXHISTORY**. This list allows you to **UNDO** edits. You reset this the same way as **LISPXHISTORY**.

---

### 43.4 System Menus

---

System menus are all bound to global variables and are easy to modify. If the menu name is set to the **NIL** value, the menu will be recreated using an items list bound to a global variable.

To change a system menu, edit the items list bound to the appropriate global variable (system menus use this items list with the default **WHENSELECTEDFN**), then set the value of the menu name to **NIL**. The next time you need the menu, it will be created

	from the items list you just edited. The names of system menus and items lists follow.
BackgroundMenu	This is the variable bound to the menu that displays when you press the right button in the grey background area of the screen.
BackgroundMenuCommands	This list used for the list of ITEMS for the background menu when it is created.
WindowMenu	WindowMenu is the variable bound to the default window menu displayed when the right mouse button is pressed inside of a window.
WindowMenuCommands	This is the list of ITEMS for the WindowMenu.
BreakMenu	The menu displayed when the middle mouse button is pressed in a break window.
BreakMenuCommands	The list of ITEMS for the BreakMenu.

---

## 43.5 Windows

---

PROMPTWINDOW	Global name of the prompt window.
T	Although the value T has several meanings (such as universal TRUE), it also stands for the standard output stream. As this is usually the Interlisp-D Executive Window, it may be used as the name for the TTY Window at the top level. Mouse processes have their own TTY Windows. A reference to the window T in a mouse driven function (e.g. a WHENSELECTFN, See Section 27.1.2, Page 27.2), will open a "TTY Window for Mouse".

---

## 43.6 Miscellaneous

---

CLEANUPOPTIONS	This is a list of options that you set to automate clean up after a work session. Example options are listing files, or recompilation. You will want to keep this set to <b>NIL</b> until you become comfortable with the machine.
FILELST	The list of all the files you loaded.
SYSFILES	The list of all the files loaded for the SYSOUT file.
INITIALS	This is an atom that you can bind to your name. If bound, the editor will add your name, in addition to the date, in the editor comment at the beginning of each function.
FIRSTNAME	If this variable is set, the system will use it to greet you personally when you log on to your machine.
INITIALSLST	A list of elements of the form: (USERNAME . INITIALS) or (USERNAME FIRSTNAME INITIALS). This list is used by the function <b>GREET</b> to set your <b>INITIALS</b> , and your <b>FIRSTNAME</b> when you login.

- #CAREFULCOLUMNS** Is an integer. **PRETTYPRINT** estimates the number of characters in an atom, instead of computing it, for efficiency. Unfortunately, for very long atom names, errors can occur. **#CAREFULCOLUMNS** is the number of columns from the right within which **PRETTYPRINT** should compute the number of characters in each atom, to prevent these errors. Initially this is set to zero, (0), **PRETTYPRINT** never computes the number of characters in an atom. If you set it to 20 or 30, when **PRETTYPRINT** comes within 20 or 30 columns of the right of the window, it will begin computing exactly how many characters are in each atom. This will prevent errors.
- DWIMWAIT** is bound to the number of seconds DWIM should wait before it uses the default response, **FIXSPELLDEFAULT**, to answer its question.
- FIXSPELLDEFAULT** is bound to either **Y** or **N**. Its value is used as the default answer to questions asked by DWIM that you don't answer in DWIMWAIT seconds. It is initially bound to **Y**, but is rebound to **N** when DWIMIFYing.
- \TimeZoneComp** This is a global variable set to the absolute value of the time offset from Greenwich. For EST, **\TimeZoneComp** should be set to 5.

[This page intentionally left blank]

## 44. OTHER REFERENCES THAT WILL BE USEFUL TO YOU

---

Here are some references to works that will be useful to you in addition to this primer. Some of these you have already been referred to, such as:

- The Interlisp-D Reference Manual
- The Library Packages Manual
- The User's Guide to SKETCH
- The 1186 or 1108 User's Guide

In addition, you can learn more about LISP with the books:

- **Interlisp-D: The language and its usage** by Steven H. Kaisler. This book was published in 1986 by John Wiley and Sons, NY.
- **Essential LISP** by John Anderson, Albert Corbett, and Brian Reiser. This book was published in 1986 by Addison Wesley Publishing Company, Reading, MA. It was informed by research on how beginners learn LISP.
- **The Little Lisper** by Daniel P. Friedman and Matthias Felleisen. The second edition of this book was published in 1986 by SRA Associates, Chicago. This book is a deceptively simple introduction to recursive programming and the flexible data structures provided by LISP.
- **LISP** by Patrick Winston and Berthold Horn. The second edition of this book was published in 1985 by the Addison Wesley Publishing Company, Reading, MA.
- **LISP: A Gentle Introduction to Symbolic Computation** by David S. Touretzky. This book was published in 1984 by the Harper and Row Publishing Company, NY.

Finally, there are three articles about the Interlisp Programming environment:

- **Power Tools for Programmers** by Beau Sheil. It appeared in *Datamation* in February, 1983, Pages 131 - 144.
- **The Interlisp Programming Environment** by Warren Teitelman and Larry Masinter. It appeared in April, 1981, in *IEEE Computer*, Volume 14:1, Pages 25 - 34.
- **Programming in an Interactive Environment, the LISP Experience** by Erik Sandewall. It appeared in March, 1978, in the *ACM Computing Surveys*, Volume 10:1, pages 35 - 71.

Each of these articles was reprinted in the book **Interactive Programming Environments** by David R. Barstow, Howard E.

Shrobe, and Erik Sandewall. This book was published in 1984 by McGraw Hill, NY. The first article can be found on pages 19 - 30, the second on pages 83 - 96, and the third on pages 31 - 80.

**Xerox Artificial Intelligence Systems**  
**250 North Halstead Street**  
**P.O. Box 7018**  
**Pasadena, California 91109-7018**