<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.1.8. INSPECTWs</td>
<td>26.6</td>
</tr>
<tr>
<td>26.2. PROMPTFORWORD</td>
<td>26.9</td>
</tr>
<tr>
<td>26.3. ASKUSER</td>
<td>26.12</td>
</tr>
<tr>
<td>26.3.1. Format of KEYLST</td>
<td>26.13</td>
</tr>
<tr>
<td>26.3.2. Options</td>
<td>26.15</td>
</tr>
<tr>
<td>26.3.3. Operation</td>
<td>26.17</td>
</tr>
<tr>
<td>26.3.4. Completing a Key</td>
<td>26.18</td>
</tr>
<tr>
<td>26.3.5. Special Keys</td>
<td>26.19</td>
</tr>
<tr>
<td>26.3.6. Startup Protocol and Typeahead</td>
<td>26.20</td>
</tr>
<tr>
<td>26.4. TTYIN Display Typein Editor</td>
<td>26.22</td>
</tr>
<tr>
<td>26.4.1. Entering Input With TTYIN</td>
<td>26.22</td>
</tr>
<tr>
<td>26.4.3. Display Editing Commands</td>
<td>26.25</td>
</tr>
<tr>
<td>26.4.4. Using TTYIN for Lisp Input</td>
<td>26.28</td>
</tr>
<tr>
<td>26.4.5. Useful Macros</td>
<td>26.29</td>
</tr>
<tr>
<td>26.4.6. Programming With TTYIN</td>
<td>26.29</td>
</tr>
<tr>
<td>26.4.7. Using TTYIN as a General Editor</td>
<td>26.32</td>
</tr>
<tr>
<td>26.4.8. ? = Handler</td>
<td>26.33</td>
</tr>
<tr>
<td>26.4.9. Read Macros</td>
<td>26.34</td>
</tr>
<tr>
<td>26.4.10. Assorted Flags</td>
<td>26.36</td>
</tr>
<tr>
<td>26.4.11. Special Responses</td>
<td>26.38</td>
</tr>
<tr>
<td>26.4.12. Display Types</td>
<td>26.38</td>
</tr>
<tr>
<td>26.5. Prettyprint</td>
<td>26.39</td>
</tr>
<tr>
<td>26.5.1. Comment Feature</td>
<td>26.42</td>
</tr>
<tr>
<td>26.5.2. Comment Pointers</td>
<td>26.44</td>
</tr>
<tr>
<td>26.5.3. Converting Comments to Lower Case</td>
<td>26.46</td>
</tr>
<tr>
<td>26.5.4. Special Prettyprint Controls</td>
<td>26.47</td>
</tr>
<tr>
<td>27. Graphics Output Operations</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1. Primitive Graphics Concepts</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1.1. Positions</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1.2. Regions</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1.3. Bitmaps</td>
<td>27.3</td>
</tr>
<tr>
<td>27.1.4. Textures</td>
<td>27.6</td>
</tr>
<tr>
<td>27.2. Opening Image Streams</td>
<td>27.8</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>27.3</td>
<td>Accessing Image Stream Fields</td>
</tr>
<tr>
<td>27.4</td>
<td>Current Position of an Image Stream</td>
</tr>
<tr>
<td>27.5</td>
<td>Moving Bits Between Bitmaps With BITBLT</td>
</tr>
<tr>
<td>27.6</td>
<td>Drawing Lines</td>
</tr>
<tr>
<td>27.7</td>
<td>Drawing Curves</td>
</tr>
<tr>
<td>27.8</td>
<td>Miscellaneous Drawing and Printing Operations</td>
</tr>
<tr>
<td>27.9</td>
<td>Drawing and Shading Grids</td>
</tr>
<tr>
<td>27.10</td>
<td>Display Streams</td>
</tr>
<tr>
<td>27.12</td>
<td>Fonts</td>
</tr>
<tr>
<td>27.13</td>
<td>Font Files and Font Directories</td>
</tr>
<tr>
<td>27.15</td>
<td>Font Profiles</td>
</tr>
<tr>
<td>27.16</td>
<td>Image Objects</td>
</tr>
<tr>
<td>27.16.1</td>
<td>IMAGEFNS Methods</td>
</tr>
<tr>
<td>27.16.2</td>
<td>Registering Image Objects</td>
</tr>
<tr>
<td>27.16.3</td>
<td>Reading and Writing Image Objects on Files</td>
</tr>
<tr>
<td>27.16.4</td>
<td>Copying Image Objects Between Windows</td>
</tr>
<tr>
<td>27.17</td>
<td>Implementation of Image Streams</td>
</tr>
<tr>
<td>28.1</td>
<td>Using The Window System</td>
</tr>
<tr>
<td>28.2</td>
<td>Changing Window Command Menus</td>
</tr>
<tr>
<td>28.3</td>
<td>Interactive Display Functions</td>
</tr>
<tr>
<td>28.4</td>
<td>Windows</td>
</tr>
<tr>
<td>28.4.1</td>
<td>Window Properties</td>
</tr>
<tr>
<td>28.4.2</td>
<td>Creating Windows</td>
</tr>
<tr>
<td>28.4.3</td>
<td>Opening and Closing Windows</td>
</tr>
<tr>
<td>28.4.4</td>
<td>Redisplaying Windows</td>
</tr>
<tr>
<td>28.4.5</td>
<td>Reshaping Windows</td>
</tr>
<tr>
<td>28.4.6</td>
<td>Moving Windows</td>
</tr>
<tr>
<td>28.4.7</td>
<td>Exposing and Burying Windows</td>
</tr>
<tr>
<td>28.4.8</td>
<td>Shrinking Windows Into Icons</td>
</tr>
<tr>
<td>28.4.9</td>
<td>Coordinate Systems, Extents, And Scrolling</td>
</tr>
<tr>
<td>28.4.10</td>
<td>Mouse Activity in Windows</td>
</tr>
<tr>
<td>28.4.11</td>
<td>Terminal I/O and Page Holding</td>
</tr>
<tr>
<td>28.4.12</td>
<td>The TTY Process and the Caret</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

## Chapter 28: Windows

28.4.13. Miscellaneous Window Functions 28.31  
28.4.15. Example: A Scrollable Window 28.34

## Chapter 28: Menus

28.5.1. Menu Fields 28.38  
28.5.2. Miscellaneous Menu Functions 28.42  
28.5.3. Examples of Menu Use 28.43

## Chapter 28: Attached Windows

28.6.1. Attaching Menus To Windows 28.48  
28.6.2. Attached Prompt Windows 28.50  
28.6.3. Window Operations And Attached Windows 28.50  
28.6.4. Window Properties Of Attached Windows 28.53

## Chapter 29: Hardcopy Facilities

29.1. Low-level Hardcopy Variables 29.5

## Chapter 30: Terminal Input/Output

30.1. Interrupt Characters 30.1  
30.2. Terminal Tables 30.4  
30.2.1. Terminal Syntax Classes 30.5  
30.2.2. Terminal Control Functions 30.6  
30.2.3. Line-Buffering 30.9  
30.3. Dribble Files 30.12  
30.4. Cursor and Mouse 30.13  
30.4.1. Changing the Cursor Image 30.13  
30.4.2. Flashing Bars on the Cursor 30.16  
30.4.3. Cursor Position 30.17  
30.4.4. Mouse Button Testing 30.17  
30.4.5. Low Level Mouse Functions 30.18  
30.5. Keyboard Interpretation 30.19  
30.6. Display Screen 30.22  
30.7. Miscellaneous Terminal I/O 30.24

## Chapter 31: Ethernet

31.1. Ethernet Protocols 31.1  
31.1.1. Protocol Layering 31.1  
31.1.2. Level Zero Protocols 31.2
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.1.3. Level One Protocols</td>
<td>31.3</td>
</tr>
<tr>
<td>31.1.4. Higher Level Protocols</td>
<td>31.4</td>
</tr>
<tr>
<td>31.1.5. Connecting Networks: Routers and Gateways</td>
<td>31.4</td>
</tr>
<tr>
<td>31.1.6. Addressing Conflicts with Level Zero Mediums</td>
<td>31.5</td>
</tr>
<tr>
<td>31.1.7. References</td>
<td>31.5</td>
</tr>
<tr>
<td>31.2. Higher-level PUP Protocol Functions</td>
<td>31.6</td>
</tr>
<tr>
<td>31.3. Higher-level NS Protocol Functions</td>
<td>31.7</td>
</tr>
<tr>
<td>31.3.1. Name and Address Conventions</td>
<td>31.7</td>
</tr>
<tr>
<td>31.3.2. Clearinghouse Functions</td>
<td>31.9</td>
</tr>
<tr>
<td>31.3.3. NS Printing</td>
<td>31.12</td>
</tr>
<tr>
<td>31.3.4. SPP Stream Interface</td>
<td>31.12</td>
</tr>
<tr>
<td>31.3.5. Courier Remote Procedure Call Protocol</td>
<td>31.15</td>
</tr>
<tr>
<td>31.3.5.1. Defining Courier Programs</td>
<td>31.15</td>
</tr>
<tr>
<td>31.3.5.2. Courier Type Definitions</td>
<td>31.17</td>
</tr>
<tr>
<td>31.3.5.2.1. Pre-defined Types</td>
<td>31.17</td>
</tr>
<tr>
<td>31.3.5.2.2. Constructed Types</td>
<td>31.18</td>
</tr>
<tr>
<td>31.3.5.2.3. User Extensions to the Type Language</td>
<td>31.19</td>
</tr>
<tr>
<td>31.3.5.3. Performing Courier Transactions</td>
<td>31.20</td>
</tr>
<tr>
<td>31.3.5.3.1. Expedited Procedure Call</td>
<td>31.22</td>
</tr>
<tr>
<td>31.3.5.3.2. Expanding Ring Broadcast</td>
<td>31.23</td>
</tr>
<tr>
<td>31.3.5.3.3. Using Bulk Data Transfer</td>
<td>31.24</td>
</tr>
<tr>
<td>31.3.5.3.4. Courier Subfunctions for Data Transfer</td>
<td>31.25</td>
</tr>
<tr>
<td>31.4. Level One Ether Packet Format</td>
<td>31.26</td>
</tr>
<tr>
<td>31.5. PUP Level One Functions</td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.1. Creating and Managing Pups</td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.2. Sockets</td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.3. Sending and Receiving Pups</td>
<td>31.29</td>
</tr>
<tr>
<td>31.5.4. Pup Routing Information</td>
<td>31.30</td>
</tr>
<tr>
<td>31.5.5. Miscellaneous PUP Utilities</td>
<td>31.31</td>
</tr>
<tr>
<td>31.5.6. PUP Debugging Aids</td>
<td>31.32</td>
</tr>
<tr>
<td>31.6. NS Level One Functions</td>
<td>31.36</td>
</tr>
<tr>
<td>31.6.1. Creating and Managing XIPs</td>
<td>31.36</td>
</tr>
<tr>
<td>31.6.2. NS Sockets</td>
<td>31.37</td>
</tr>
<tr>
<td>31.6.3. Sending and Receiving XIPs</td>
<td>31.37</td>
</tr>
<tr>
<td>31.6.4. NS Debugging Aids</td>
<td>31.38</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>31.7. Support for Other Level One Protocols</td>
<td>31.38</td>
</tr>
<tr>
<td>31.8. The SYSQUEUE mechanism</td>
<td>31.41</td>
</tr>
</tbody>
</table>
[This page intentionally left blank]
### TABLE OF CONTENTS

#### 24. Streams and Files

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.1. Opening and Closing File Streams</td>
<td>24.2</td>
</tr>
<tr>
<td>24.2. File Names</td>
<td>24.5</td>
</tr>
<tr>
<td>24.3. Incomplete File Names</td>
<td>24.9</td>
</tr>
<tr>
<td>24.4. Version Recognition</td>
<td>24.11</td>
</tr>
<tr>
<td>24.5. Using File Names Instead of Streams</td>
<td>24.13</td>
</tr>
<tr>
<td>24.5.1. File Name Efficiency Considerations</td>
<td>24.14</td>
</tr>
<tr>
<td>24.5.2. Obsolete File Opening Functions</td>
<td>24.14</td>
</tr>
<tr>
<td>24.5.3. Converting Old Programs</td>
<td>24.15</td>
</tr>
<tr>
<td>24.6. Using Files with Processes</td>
<td>24.16</td>
</tr>
<tr>
<td>24.7. File Attributes</td>
<td>24.17</td>
</tr>
<tr>
<td>24.8. Closing and Reopening Files</td>
<td>24.20</td>
</tr>
<tr>
<td>24.9. Local Hard Disk Device</td>
<td>24.21</td>
</tr>
<tr>
<td>24.10. Floppy Disk Device</td>
<td>24.24</td>
</tr>
<tr>
<td>24.11. I/O Operations to and from Strings</td>
<td>24.28</td>
</tr>
<tr>
<td>24.12. Temporary Files and the CORE Device</td>
<td>24.29</td>
</tr>
<tr>
<td>24.13. NULL Device</td>
<td>24.30</td>
</tr>
<tr>
<td>24.15. Deleting, Copying, and Renaming Files</td>
<td>24.31</td>
</tr>
<tr>
<td>24.16. Searching File Directories</td>
<td>24.31</td>
</tr>
<tr>
<td>24.17. Listing File Directories</td>
<td>24.33</td>
</tr>
<tr>
<td>24.18. File Servers</td>
<td>24.36</td>
</tr>
<tr>
<td>24.18.1. Pup File Server Protocols</td>
<td>24.36</td>
</tr>
<tr>
<td>24.18.2. Xerox NS File Server Protocols</td>
<td>24.37</td>
</tr>
<tr>
<td>24.18.3. Operating System Designations</td>
<td>24.38</td>
</tr>
<tr>
<td>24.18.4. Logging In</td>
<td>24.39</td>
</tr>
<tr>
<td>24.18.5. Abnormal Conditions</td>
<td>24.41</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>26.1.8</td>
<td>INSPECTWs</td>
</tr>
<tr>
<td>26.2</td>
<td>PROMPTFORWORD</td>
</tr>
<tr>
<td>26.3</td>
<td>ASKUSER</td>
</tr>
<tr>
<td></td>
<td>26.3.1 Format of KEYLST</td>
</tr>
<tr>
<td></td>
<td>26.3.2 Options</td>
</tr>
<tr>
<td></td>
<td>26.3.3 Operation</td>
</tr>
<tr>
<td></td>
<td>26.3.4 Completing a Key</td>
</tr>
<tr>
<td></td>
<td>26.3.5 Special Keys</td>
</tr>
<tr>
<td></td>
<td>26.3.6 Startup Protocol and Typeahead</td>
</tr>
<tr>
<td>26.4</td>
<td>TTYIN Display Typein Editor</td>
</tr>
<tr>
<td></td>
<td>26.4.1 Entering Input With TTYIN</td>
</tr>
<tr>
<td></td>
<td>26.4.2 Mouse Commands [Interlisp-D Only]</td>
</tr>
<tr>
<td></td>
<td>26.4.3 Display Editing Commands</td>
</tr>
<tr>
<td></td>
<td>26.4.4 Using TTYIN for Lisp Input</td>
</tr>
<tr>
<td></td>
<td>26.4.5 Useful Macros</td>
</tr>
<tr>
<td></td>
<td>26.4.6 Programming With TTYIN</td>
</tr>
<tr>
<td></td>
<td>26.4.7 Using TTYIN as a General Editor</td>
</tr>
<tr>
<td></td>
<td>26.4.8 ? = Handler</td>
</tr>
<tr>
<td></td>
<td>26.4.9 Read Macros</td>
</tr>
<tr>
<td></td>
<td>26.4.10 Assorted Flags</td>
</tr>
<tr>
<td></td>
<td>26.4.11 Special Responses</td>
</tr>
<tr>
<td></td>
<td>26.4.12 Display Types</td>
</tr>
<tr>
<td>26.5</td>
<td>Prettyprint</td>
</tr>
<tr>
<td></td>
<td>26.5.1 Comment Feature</td>
</tr>
<tr>
<td></td>
<td>26.5.2 Comment Pointers</td>
</tr>
<tr>
<td></td>
<td>26.5.3 Converting Comments to Lower Case</td>
</tr>
<tr>
<td></td>
<td>26.5.4 Special Prettyprint Controls</td>
</tr>
<tr>
<td>27.</td>
<td>Graphics Output Operations</td>
</tr>
<tr>
<td>27.1</td>
<td>Primitive Graphics Concepts</td>
</tr>
<tr>
<td></td>
<td>27.1.1 Positions</td>
</tr>
<tr>
<td></td>
<td>27.1.2 Regions</td>
</tr>
<tr>
<td></td>
<td>27.1.3 Bitmaps</td>
</tr>
<tr>
<td></td>
<td>27.1.4 Textures</td>
</tr>
<tr>
<td>27.2</td>
<td>Opening Image Streams</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS

27.3. Accessing Image Stream Fields 27.10  
27.5. Moving Bits Between Bitmaps With BITBLT 27.14  
27.6. Drawing Lines 27.17  
27.7. Drawing Curves 27.18  
27.8. Miscellaneous Drawing and Printing Operations 27.20  
27.9. Drawing and Shading Grids 27.22  
27.10. Display Streams 27.23  
27.12. Fonts 27.25  
27.13. Font Files and Font Directories 27.31  
27.15. Font Profiles 27.32  
27.16. Image Objects 27.35  
	a. IMAGEFNS Methods 27.36  
	b. Registering Image Objects 27.39  
c. Reading and Writing Image Objects on Files 27.40  
d. Copying Image Objects Between Windows 27.41  
27.17. Implementation of Image Streams 27.42  

### 28. Windows and Menus 28.1  
28.2. Changing Window Command Menus 28.7  
28.3. Interactive Display Functions 28.9  
	a. Window Properties 28.13  
	b. Creating Windows 28.13  
	c. Opening and Closing Windows 28.15  
d. Redisplaying Windows 28.16  
e. Reshaping Windows 28.16  
f. Moving Windows 28.19  
g. Exposing and Burying Windows 28.20  
h. Shrinking Windows Into Icons 28.21  
i. Coordinate Systems, Extents, And Scrolling 28.23  
j. Mouse Activity in Windows 28.27  
k. Terminal I/O and Page Holding 28.29  
l. The TTY Process and the Caret 28.30
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.4.13. Miscellaneous Window Functions</td>
<td>28.31</td>
</tr>
<tr>
<td>28.4.15. Example: A Scrollable Window</td>
<td>28.34</td>
</tr>
<tr>
<td>28.5. Menus</td>
<td>28.37</td>
</tr>
<tr>
<td>28.5.1. Menu Fields</td>
<td>28.38</td>
</tr>
<tr>
<td>28.5.2. Miscellaneous Menu Functions</td>
<td>28.42</td>
</tr>
<tr>
<td>28.5.3. Examples of Menu Use</td>
<td>28.43</td>
</tr>
<tr>
<td>28.6. Attached Windows</td>
<td>28.45</td>
</tr>
<tr>
<td>28.6.1. Attaching Menus To Windows</td>
<td>28.48</td>
</tr>
<tr>
<td>28.6.2. Attached Prompt Windows</td>
<td>28.50</td>
</tr>
<tr>
<td>28.6.3. Window Operations And Attached Windows</td>
<td>28.50</td>
</tr>
<tr>
<td>28.6.4. Window Properties Of Attached Windows</td>
<td>28.53</td>
</tr>
<tr>
<td>29. Hardcopy Facilities</td>
<td>29.1</td>
</tr>
<tr>
<td>29.1. Low-level Hardcopy Variables</td>
<td>29.5</td>
</tr>
<tr>
<td>30. Terminal Input/Output</td>
<td>30.1</td>
</tr>
<tr>
<td>30.1. Interrupt Characters</td>
<td>30.1</td>
</tr>
<tr>
<td>30.2. Terminal Tables</td>
<td>30.4</td>
</tr>
<tr>
<td>30.2.1. Terminal Syntax Classes</td>
<td>30.5</td>
</tr>
<tr>
<td>30.2.2. Terminal Control Functions</td>
<td>30.6</td>
</tr>
<tr>
<td>30.2.3. Line-Buffering</td>
<td>30.9</td>
</tr>
<tr>
<td>30.3. Dribble Files</td>
<td>30.12</td>
</tr>
<tr>
<td>30.4. Cursor and Mouse</td>
<td>30.13</td>
</tr>
<tr>
<td>30.4.1. Changing the Cursor Image</td>
<td>30.13</td>
</tr>
<tr>
<td>30.4.2. Flashing Bars on the Cursor</td>
<td>30.16</td>
</tr>
<tr>
<td>30.4.3. Cursor Position</td>
<td>30.17</td>
</tr>
<tr>
<td>30.4.4. Mouse Button Testing</td>
<td>30.17</td>
</tr>
<tr>
<td>30.4.5. Low Level Mouse Functions</td>
<td>30.18</td>
</tr>
<tr>
<td>30.5. Keyboard Interpretation</td>
<td>30.19</td>
</tr>
<tr>
<td>30.6. Display Screen</td>
<td>30.22</td>
</tr>
<tr>
<td>30.7. Miscellaneous Terminal I/O</td>
<td>30.24</td>
</tr>
<tr>
<td>31. Ethernet</td>
<td>31.1</td>
</tr>
<tr>
<td>31.1. Ethernet Protocols</td>
<td>31.1</td>
</tr>
<tr>
<td>31.1.1. Protocol Layering</td>
<td>31.1</td>
</tr>
<tr>
<td>31.1.2. Level Zero Protocols</td>
<td>31.2</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.1.3. Level One Protocols</td>
<td>31.3</td>
</tr>
<tr>
<td>31.1.4. Higher Level Protocols</td>
<td>31.4</td>
</tr>
<tr>
<td>31.1.5. Connecting Networks: Routers and Gateways</td>
<td>31.4</td>
</tr>
<tr>
<td>31.1.6. Addressing Conflicts with Level Zero Mediums</td>
<td>31.5</td>
</tr>
<tr>
<td>31.1.7. References</td>
<td>31.5</td>
</tr>
<tr>
<td><strong>31.2. Higher-level PUP Protocol Functions</strong></td>
<td>31.6</td>
</tr>
<tr>
<td><strong>31.3. Higher-level NS Protocol Functions</strong></td>
<td>31.7</td>
</tr>
<tr>
<td>31.3.1. Name and Address Conventions</td>
<td>31.7</td>
</tr>
<tr>
<td>31.3.2. Clearinghouse Functions</td>
<td>31.9</td>
</tr>
<tr>
<td>31.3.3. NS Printing</td>
<td>31.12</td>
</tr>
<tr>
<td>31.3.4. SPP Stream Interface</td>
<td>31.12</td>
</tr>
<tr>
<td>31.3.5. Courier Remote Procedure Call Protocol</td>
<td>31.15</td>
</tr>
<tr>
<td>31.3.5.1. Defining Courier Programs</td>
<td>31.15</td>
</tr>
<tr>
<td>31.3.5.2. Courier Type Definitions</td>
<td>31.17</td>
</tr>
<tr>
<td>31.3.5.2.1. Pre-defined Types</td>
<td>31.17</td>
</tr>
<tr>
<td>31.3.5.2.2. Constructed Types</td>
<td>31.18</td>
</tr>
<tr>
<td>31.3.5.2.3. User Extensions to the Type Language</td>
<td>31.19</td>
</tr>
<tr>
<td>31.3.5.3. Performing Courier Transactions</td>
<td>31.20</td>
</tr>
<tr>
<td>31.3.5.3.1. Expedited Procedure Call</td>
<td>31.22</td>
</tr>
<tr>
<td>31.3.5.3.2. Expanding Ring Broadcast</td>
<td>31.23</td>
</tr>
<tr>
<td>31.3.5.3.3. Using Bulk Data Transfer</td>
<td>31.24</td>
</tr>
<tr>
<td>31.3.5.3.4. Courier Subfunctions for Data Transfer</td>
<td>31.25</td>
</tr>
<tr>
<td><strong>31.4. Level One Ether Packet Format</strong></td>
<td>31.26</td>
</tr>
<tr>
<td><strong>31.5. PUP Level One Functions</strong></td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.1. Creating and Managing Pups</td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.2. Sockets</td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.3. Sending and Receiving Pups</td>
<td>31.29</td>
</tr>
<tr>
<td>31.5.4. Pup Routing Information</td>
<td>31.30</td>
</tr>
<tr>
<td>31.5.5. Miscellaneous PUP Utilities</td>
<td>31.31</td>
</tr>
<tr>
<td>31.5.6. PUP Debugging Aids</td>
<td>31.32</td>
</tr>
<tr>
<td><strong>31.6. NS Level One Functions</strong></td>
<td>31.36</td>
</tr>
<tr>
<td>31.6.1. Creating and Managing XIPs</td>
<td>31.36</td>
</tr>
<tr>
<td>31.6.2. NS Sockets</td>
<td>31.37</td>
</tr>
<tr>
<td>31.6.3. Sending and Receiving XIPs</td>
<td>31.37</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>31.6.4. NS Debugging Aids</td>
<td>31.38</td>
</tr>
<tr>
<td>31.7. Support for Other Level One Protocols</td>
<td>31.38</td>
</tr>
<tr>
<td>31.8. The SYSQUEUE mechanism</td>
<td>31.41</td>
</tr>
</tbody>
</table>
[This page intentionally left blank]
Interlisp-D can perform input/output operations on a large variety of physical devices, including local disk drives, floppy disk drives, the keyboard and display screen, and remote file server computers accessed over a network. While the low-level details of how all these devices perform input/output vary considerably, the Interlisp-D language provides the programmer a small, common set of abstract operations whose use is largely independent of the physical input/output medium involved—operations such as read, print, change font, or go to a new line. By merely changing the targeted I/O device, a single program can be used to produce output on the display, a file, or a printer.

The underlying data abstraction that permits this flexibility is the stream. A stream is a data object (an instance of the data type STREAM) that encapsulates all of the information about an input/output connection to a particular I/O device. Each of Interlisp-D's general-purpose I/O functions takes a stream as one of its arguments. The general-purpose function then performs action specific to the stream's device to carry out the requested operation. Not every device is capable of implementing every I/O operation, while some devices offer additional functionality by way of special functions for that device alone. Such restrictions and extensions are noted in the documentation of each device.

The vast majority of the streams commonly used in Interlisp-D fall into two interesting categories: the file stream and the image stream.

A file is an ordered collection of data, usually a sequence of characters or bytes, stored on a file device in a manner that allows the data to be retrieved at a later time. Floppy disks, hard disks, and remote file servers are among the devices used to store files. Files are identified by a "file name", which specifies the device on which the file resides and a name unique to a specific file on that device. Input or output to a file is performed by obtaining a stream to the file, using OPENSTREAM (page 24.2). In addition, there are functions that manipulate the files themselves, rather than their data content.

An image stream is an output stream to a display device, such as the display screen or a printer. In addition to the standard output operations, such as print, an image stream implements a variety of graphics operations, such as drawing lines and displaying characters in multiple fonts. Unlike a file, the
"content" of an image stream cannot be retrieved. Image streams are described on page 27.8.

The creation of other kinds of streams, such as network byte-stream connections, is described in the chapters peculiar to those kinds of streams. The operations common to streams in general are described on page 25.1. This chapter describes operations specific to file devices: how to name files, how to open streams to files, and how to manipulate files on their devices.

24.1 Opening and Closing File Streams

In order to perform input from or output to a file, it is necessary to create a stream to the file, using OPENSTREAM:

```
(OPENSTREAM FILE ACCESS RECOG PARAMETERS —)  [Function]
```

Opens and returns a stream for the file specified by FILE, a file name. FILE can be either a string or a litatom. The syntax and manipulation of file names is described at length on page 24.5. Incomplete file names are interpreted with respect to the connected directory (page 24.10).

RECOG specifies the recognition mode of FILE, as described on page 24.12. If RECOG = NIL, it defaults according to the value of ACCESS.

ACCESS specifies the "access rights" to be used when opening the file, one of the following:

**INPUT**

Only input operations are permitted on the file. The file must already exist. Starts reading at the beginning of the file. RECOG defaults to OLD.

**OUTPUT**

Only output operations are permitted on the file. Starts writing at the beginning of the file, which is initially empty. While the file is open, other users or processes are unable to open the file for either input or output. RECOG defaults to NEW.

**BOTH**

Both input and output operations are permitted on the file. Starts reading or writing at the beginning of the file. RECOG defaults to OLD/NEW. ACCESS = BOTH implies random accessibility (page 25.18), and thus may not be possible for files on some devices.

**APPEND**

Only sequential output operations are permitted on the file. Starts writing at the end of the file. RECOG defaults to OLD/NEW. ACCESS = APPEND may not be allowed for files on some devices.
Note: ACCESS = OUTPUT implies that one intends to write a new or different file, even if a version number was specified and the corresponding file already exists. Thus any previous contents of the file are discarded, and the file is empty immediately after the OPENSTREAM. If it is desired to write on an already existing file while preserving the old contents, the file must be opened for access BOTH or APPEND.

PARAMETERS is a list of pairs (ATTRIB VALUE), where ATTRIB is any file attribute that the file system is willing to allow the user to set (see SETFILEINFO, page 24.17). A non-list ATTRIB in PARAMETERS is treated as the pair (ATTRIB T). Generally speaking, attributes that belong to the permanent file (e.g., TYPE) can only be set when creating a new file, while attributes that belong only to a particular opening of a file (e.g., ENDOFSTREAMOP) can be set on any call to OPENSTREAM. Not all devices honor all attributes; those not recognized by a particular device are simply ignored.

In addition to the attributes permitted by SETFILEINFO, the following tokens are accepted by OPENSTREAM as values of ATTRIB in its PARAMETERS argument:

**DON'T.CHANGE.DATE**
If VALUE is non-NIL, the file's creation date (page 24.17) is not changed when the file is opened. This option is meaningful only for old files being opened for access BOTH. This should be used only for specialized applications in which the caller does not want the file system to believe the file's content has been changed.

**SEQUENTIAL**
If VALUE is non-NIL, this opening of the file need support only sequential access; i.e., the caller intends never to use SETFILEPTR. For some devices, sequential access to files is much more efficient than random access. Note that the device may choose to ignore this attribute and still open the file in a manner that permits random access. Also note that this attribute does not make sense with ACCESS = BOTH.

If FILE is not recognized by the file system, OPENSTREAM causes the error FILE NOT FOUND. Ordinarily, this error is intercepted via an entry on ERRORTYPELST (page 14.22), which causes SPELLFILE (page 24.32) to be called. SPELLFILE searches alternate directories and possibly attempts spelling correction on the file name. Only if SPELLFILE is unsuccessful will the FILE NOT FOUND error actually occur.

If FILE exists but cannot be opened, OPENSTREAM causes one of several other errors: FILE WON'T OPEN if the file is already opened for conflicting access by someone else; PROTECTION VIOLATION if the file is protected against the operation; FILE SYSTEM RESOURCES EXCEEDED if there is no more room in the file system.
OPENING AND CLOSING FILE STREAMS

(CLOSEF FILE) [Function]
Closes FILE, and returns its full file name. Generates an error, FILE NOT OPEN, if FILE does not designate an open stream. After closing a stream, no further input/output operations are permitted on it.

If FILE is NIL, it is defaulted to the primary input stream if that is not the terminal stream, or else the primary output stream if that is not the terminal stream. If both primary input and output streams are the terminal input/output streams, CLOSEF returns NIL. If CLOSEF closes either the primary input stream or the primary output stream (either explicitly or in the FILE = NIL case), it resets the primary stream for that direction to be the corresponding terminal stream. See page 25.3 for information on the primary input/output streams.

WHENCLOSE (page 24.20) allows the user to "advise" CLOSEF to perform various operations when a file is closed.

Because of buffering, the contents of a file open for output are not guaranteed to be written to the actual physical file device until CLOSEF is called. Buffered data can be forced out to a file without closing the file by using the function FORCEOUTPUT (page 25.10).

Some network file devices perform their transactions in the background. As a result, it is possible for a file to be closed by CLOSEF and yet not be "fully" closed for some small period of time afterward, during which time the file appears to still be busy, and cannot be opened for conflicting access by other users.

(CLOSEF? FILE) [Function]
Closes FILE if it is open, returning the value of CLOSEF; otherwise does nothing and returns NIL.

In the present implementation of Interlisp-D, all streams to files are kept, while open, in a registry of "open files". This registry does not include nameless streams, such as string streams (page 24.28), display streams (page 28.29), and the terminal input and output streams; nor streams explicitly hidden from the user, such as dribble streams (page 30.12). This registry may not persist in future implementations of Interlisp-D, but at the present time it is accessible by the following two functions:

(OPENP FILE ACCESS) [Function]
ACCESS is an access mode for a stream opening (one of INPUT, OUTPUT, BOTH, or APPEND), or NIL, meaning any access.

If FILE is a stream, returns its full name if it is open for the specified access, else NIL.
If \textit{FILE} is a file name (a litatom), \textit{FILE} is processed according to the rules of file recognition (page 24.12). If a stream open to a file by that name is registered and open for the specified access, then the file's full name is returned. If the file name is not recognized, or no stream is open to the file with the specified access, \textbf{NIL} is returned.

If \textit{FILE} is \textbf{NIL}, returns a list of the full names of all registered streams that are open for the specified access.

\textbf{(CLOSEALL \textit{ALLFLG})} [Function]

Closes all streams in the value of (OPENP). Returns a list of the files closed.

\textit{WHENCLOSE} (page 24.20) allows certain files to be "protected" from \textbf{CLOSEALL}. If \textit{ALLFLG} is \textbf{T}, all files, including those protected by \textit{WHENCLOSE}, are closed.

\section*{24.2 File Names}

A file name in Interlisp-D is a string or litatom whose characters specify a "path" to the actual file: on what host or device the file resides, in which directory, and so forth. Because Interlisp-D supports a variety of non-local file devices, parts of the path could be very device-dependent. However, it is desirable for programs to be able to manipulate file names in a device-independent manner. To this end, Interlisp-D specifies a uniform file name syntax over all devices; the functions that perform the actual file manipulation for a particular device are responsible for any translation to that device's naming conventions.

A file name is composed of a collection of \textit{fields}, some of which have specific semantic interpretations. The functions described below refer to each field by a \textit{field name}, a literal atom from among the following: \textit{HOST}, \textit{DEVICE}, \textit{DIRECTORY}, \textit{NAME}, \textit{EXTENSION}, and \textit{VERSION}. The standard syntax for a file name that contains all of those fields is \{\textit{HOST)}\textit{DEVICE:<DIRECTORY>}\textit{NAME.EXTENSION};\textit{VERSION}.

Some host's file systems do not use all of those fields in their file names.

- \textbf{HOST} Specifies the host whose file system contains the file. In the case of local file devices, the "host" is the name of the device, e.g., \textit{DSK} or \textit{FLOPPY}.
- \textbf{DEVICE} Specifies, for those hosts that divide their file system's name space among multiple physical devices, the device or logical structure on which the file resides. This should not be confused
with Interlisp-D's abstract "file device", which denotes either a host or a local physical device and is specified by the HOST field.

DIRECTORY

Specifies the "directory" containing the file. A directory usually is a grouping of a possibly large set of loosely related files, e.g., the personal files of a particular user, or the files belonging to some project. The DIRECTORY field usually consists of a principal directory and zero or more subdirectories that together describe a path through a file system's hierarchy. Each subdirectory name is set off from the previous directory or subdirectory by the character " "; e.g., "LISP>LIBRARY>NEW".

NAME

This field carries no specific meaning, but generally names a set of files thought of as being different renditions of the "same" abstract file.

EXTENSION

This field also carries no specific meaning, but generally distinguishes the form of files having the same name. Most files systems have some "conventional" extensions that denote something about the content of the file. E.g., in Interlisp-D, the extension DCOM standardly denotes a file containing compiled function definitions.

VERSION

A number used to distinguish the versions or "generations" of the files having a common name and extension. The version number is incremented each time a new file by the same name is created.

Most functions that take as input "a directory" accept either a directory name (the contents of the DIRECTORY field of a file name) or a "full" directory specification—a file name fragment consisting of only the fields HOST, DEVICE, and DIRECTORY. In particular, the "connected directory" (page 24.10) consists, in general, of all three fields.

For convenience in dealing with certain operating systems, Interlisp-D also recognizes [] and () as host delimiters (synonymous with {}), and / as a directory delimiter (synonymous with < at the beginning of a directory specification and > to terminate directory or subdirectory specification). For example, a file on a Unix file server UNX with the name /usr/foo/bar/stuff.tedit, whose DIRECTORY field is thus usr/foo/bar, could be specified as (UNX)/usr/foo/bar/stuff.tedit, or (UNX)<usr/foo/bar>stuff.tedit, or several other variations. Note that when using [] or () as host delimiters, they usually must be escaped with the reader's % escape character if the file name is expressed as a litatom rather than a string.

Different hosts have different requirements regarding which characters are valid in file names. From Interlisp-D's point of view, any characters are valid. However, in order to be able to parse a file name into its component fields, it is necessary that those characters that are conventionally used as file name delimiters be quoted when they appear inside of fields where
there could be ambiguity. The file name quoting character is "" (single quote). Thus, the following characters must be quoted when not used as delimiters: :, >, ;, /, and ' itself. The character . (period) need only be quoted if it is to be considered a part of the EXTENSION field. The characters }, [, and ) need only be quoted in a file name when the host field of the name is introduced by {, [ , and (, respectively. The characters {, [ , (, and < need only be quoted if they appear as the first character of a file name fragment, where they would otherwise be assumed to introduce the HOST or DIRECTORY fields.

The following functions are the standard way to manipulate file names in Interlisp. Their operation is purely syntactic—they perform no file system operations themselves.

(UNPACKFILENAME.STRING FILENAME) [Function]
Parses FILENAME, returning a list in property list format of alternating field names and field contents. The field contents are returned as strings. If FILENAME is a stream, its full name is used.

Only those fields actually present in FILENAME are returned. A field is considered present if its delimiting punctuation (in the case of EXTENSION and VERSION, the preceding period or semicolon, respectively) is present, even if the field itself is empty. Empty fields are denoted by "" (the empty string).

Examples:

(UNPACKFILENAME.STRING "FOO.BAR") = >
(NAME "FOO" EXTENSION "BAR")

(UNPACKFILENAME.STRING "FOO;2") = >
(NAME "FOO" EXTENSION "" VERSION "2")

(UNPACKFILENAME.STRING "FOO;") = >
(NAME "FOO" VERSION "")

(UNPACKFILENAME.STRING
 "[ERIS]<LISP>CURRENT>IMTRAN.DCOM;21")
= > (HOST "ERIS" DIRECTORY "LISP>CURRENT"
 NAME "IMTRAN" EXTENSION "DCOM"
 VERSION "21")

(UNPACKFILENAME FILE) [Function]
Old version of UNPACKFILENAME.STRING that returns the field values as atoms, rather than as strings. UNPACKFILENAME.STRING is now considered the "correct" way of unpacking file names, because it does not lose information when the contents of a field are numeric. For example,

(UNPACKFILENAME 'STUFF.TXT) = >
(NAME STUFF EXTENSION TEX)

STREAMS AND FILES
but

```lisp
(PACKFILENAME 'STUFF.029) =>
(NAME STUFF EXTENSION 29)
```

Explicitly omitted fields are denoted by the atom `NIL`, rather than the empty string.

Note: Both `UNPACKFILENAME` and `UNPACKFILENAME.STRING` leave the trailing colon on the device field, so that the Tenex device `NIL:` can be distinguished from the absence of a device. Although `UNPACKFILENAME.STRING` is capable of making the distinction, it retains this behavior for backward compatibility. Thus,

```lisp
(PACKFILENAME 'TOAST) DSK:FOO) =>
(HOST "TOAST" DEVICE "DSK:" NAME "FOO")
```

### FILENAMEFIELD FILENAME FIELDNAME

[Function]

Returns, as an atom, the contents of the `FIELDNAME` field of `FILENAME`. If `FILENAME` is a stream, its full name is used.

### PACKFILENAME.STRING FIELD$_1$ CONTENTS$_1$ ... FIELD$_N$ CONTENTS$_N$

[NoSpread Function]

Takes a sequence of alternating field names and field contents (atoms or strings), and returns the corresponding file name, as a string.

If `PACKFILENAME.STRING` is given a single argument, it is interpreted as a list of alternating field names and field contents. Thus `PACKFILENAME.STRING` and `UNPACKFILENAME.STRING` operate as inverses.

If the same field name is given twice, the first occurrence is used.

The contents of the field name `DIRECTORY` may be either a directory name or a full directory specification as described above.

`PACKFILENAME.STRING` also accepts the "field name" `BODY` to mean that its contents should itself be unpacked and spliced into the argument list at that point. This feature, in conjunction with the rule that fields early in the argument list override later duplicates, is useful for altering existing file names. For example, to provide a default field, place `BODY` first in the argument list, then the default fields. To override a field, place the new fields first and `BODY` last.

If the value of the `BODY` field is a stream, its full name is used.

Examples:

```lisp
(PACKFILENAME.STRING 'DIRECTORY "LISP"
 'NAME "NET")
 => "<LISP>NET"
```
(PACKFILENAME STRING 'NAME "NET"
 'DIRECTORY "[DSK] <LISPFILES >")
 => "[DSK] <LISPFILES> NET"

(PACKFILENAME STRING 'DIRECTORY "[DSK]"
 'BODY "(TOAST) < FOO > BAR")
 => "(DSK) BAR"

(PACKFILENAME STRING 'DIRECTORY "FRED"
 'BODY "(TOAST) < FOO > BAR")
 => "(TOAST) < FRED > BAR"

(PACKFILENAME STRING 'BODY "(TOAST) < FOO > BAR"
 'DIRECTORY "FRED")
 => "(TOAST) < FOO > BAR"

(PACKFILENAME STRING 'VERSION NIL
 'BODY "(TOAST) < FOO > BAR.DCOM; 2")
 => "(TOAST) < FOO > BAR.DCOM"

(PACKFILENAME STRING 'BODY "(TOAST) < FOO > BAR.DCOM"
 'VERSION 1)
 => "(TOAST) < FOO > BAR.DCOM; 1"

(PACKFILENAME STRING 'BODY "(TOAST) < FOO > BAR.DCOM;"
 'VERSION 1)
 => "(TOAST) < FOO > BAR.DCOM;"

(PACKFILENAME STRING 'BODY "BAR.; 1"
 'EXTENSION "DCOM")
 => "BAR.; 1"

(PACKFILENAME STRING 'BODY "BAR; 1"
 'EXTENSION "DCOM")
 => "BAR.DCOM; 1"

In the last two examples, note that in one case the extension is explicitly present in the body (as indicated by the preceding period), while in the other there is no indication of an extension, so the default is used.

(PACKFILENAME FIELD_1 CONTENTS_1 ... FIELD_N CONTENTS_N) [NoSpread Function]

The same as PACKFILENAME STRING, except that it returns the file name as a litatom, instead of a string.

24.3 Incomplete File Names

In general, it is not necessary to pass a complete file name (one containing all the fields listed above) to functions that take a file name as argument. Interlisp supplies suitable defaults for...
certain fields, as described below. Functions that return names of actual files, however, always return the fully specified name.

If the version field is omitted from a file name, Interlisp performs version recognition, as described on page 24.11.

If the host, device and/or directory field are omitted from a file name, Interlisp defaults them with respect to the currently connected directory. The connected directory is changed by calling the function CNDIR or using the programmer's assistant command CONN.

Defaults are added to the partially specified name “left to right” until a host, device or directory field is encountered. Thus, if the connected directory is `{TWENTY}PS: < FRED>, then

BAR.DCOM means
{TWENTY}PS: < FRED> BAR.DCOM

<GRANOLA> BAR.DCOM means
{TWENTY}PS: < GRANOLA> BAR.DCOM

MTA0: < GRANOLA> BAR.DCOM means
{TWENTY}MTA0: < GRANOLA> BAR.DCOM

{THIRTY} <GRANOLA> BAR.DCOM means
{THIRTY} <GRANOLA> BAR.DCOM

In addition, if the partially specified name contains a subdirectory, but no principal directory, then the subdirectory is appended to the connected directory. For example,

ISO> BAR.DCOM means
{TWENTY}PS: < FRED> ISO> BAR.DCOM

Or, if the connected directory is the Unix directory `{UNX}/usr/fred/, then iso/bar.dcom means
{UNX}/usr/fred/iso/bar.dcom, but /other/bar.dcom means
{UNX}/other/bar.dcom.

(CNDIR HOST/DIR)

Connects to the directory HOST/DIR, which can either be a directory name or a full directory specification including host and/or device. If the specification includes just a host, and the host supports directories, the directory is defaulted to the value of (USERNAME); if the host is omitted, connection is made to another directory on the same host as before. If HOST/DIR is NIL, connects to the value of LOGINHOST/DIR.

CNDIR returns the full name of the now-connected directory. Causes an error, Non-existent directory, if HOST/DIR is not recognized as a valid directory.

Note that CNDIR does not necessarily require or provide any directory access privileges. Access privileges are checked when a file is opened.
CONN HOST/DIR: [Prog. Asst. Command]

Convenient command form of CNDIR for use at the executive. Connects to HOST/DIR, or to the value of LOGINHOST/DIR if HOST/DIR is omitted. This command is undoable—undoing it causes the system to connect to the previously connected directory.

LOGINHOST/DIR: [Variable]

CONN with no argument connects to the value of the variable LOGINHOST/DIR, initially {DSK}, but usually reset in the user’s greeting file (page 12.1).

(DIRECTORYNAME DIRNAME STRPTR —): [Function]

If DIRNAME is T, returns the full specification of the currently connected directory. If DIRNAME is NIL, returns the “login” directory specification (the value of LOGINHOST/DIR). For any other value of DIRNAME, returns a full directory specification if DIRNAME designates an existing directory (satisfies DIRECTORYNAMEP), otherwise NIL.

If STRPTR is T, the value is returned as an atom, otherwise it is returned as a string.

(DIRECTORYNAMEP DIRNAME HOSTNAME): [Function]

Returns T if DIRNAME is recognized as a valid directory on host HOSTNAME, or on the host of the currently connected directory if HOSTNAME is NIL. DIRNAME may be either a directory name or a full directory specification containing host and/or device as well.

If DIRNAME includes subdirectories, this function may or may not pass judgment on their validity. Some hosts support “true” subdirectories, distinct entities manipulable by the file system, while others only provide them as a syntactic convenience.

(HOSTNAMEP NAME): [Function]

Returns T if NAME is recognized as a valid host or file device name at the moment HOSTNAMEP is called.

24.4 Version Recognition

Most of the file devices in Interlisp support file version numbers. That is, it is possible to have several files of the exact same name, differing only in their VERSION field, which is incremented for each new “version” of the file that is created. When a file name lacking a version number is presented to the file system, it is
necessary to determine which version number is intended. This process is known as version recognition.

When OPENSTREAM opens a file for input and no version number is given, the highest existing version number is used. Similarly, when a file is opened for output and no version number is given, a new file is created with a version number one higher than the highest one currently in use with that file name. The version number defaulting for OPENSTREAM can be changed by specifying a different value for its RECOG argument, as described under FULLNAME, below.

Other functions that accept file names as arguments generally perform the default version recognition, which is newest version for existing files, or a new version if using the file name to create a new file. The one exception is DELFILE, which defaults to the oldest existing version of the file.

The functions below can be used to perform version recognition without actually calling OPENSTREAM to open the file. Note that these functions only tell the truth about the moment at which they are called, and thus cannot in general be used to anticipate the name of the file opened by a comparable OPENSTREAM. They are sometimes, however, helpful hints.

\[
(FULLNAME \ X \ RECOG) \quad \text{[Function]}
\]

If \( X \) is an open stream, simply returns the full file name of the stream. Otherwise, if \( X \) is a file name given as a string or litatom, performs version recognition, as follows:

If \( X \) is recognized in the recognition mode specified by \( RECOG \) as an abbreviation for some file, returns the file's full name, otherwise NIL. \( RECOG \) is one of the following:

**OLD** Choose the newest existing version of the file. Return NIL if no file named \( X \) exists.

**OLDEST** Choose the oldest existing version of the file. Return NIL if no file named \( X \) exists.

**NEW** Choose a new (not yet existing) version of the file. That is, if versions of \( X \) already exist, then choose a version number one higher than highest existing version; else choose version 1. For some file systems, \( FULLNAME \) returns NIL if the user does not have the access rights necessary for creating a new file named \( X \).

**OLD/NEW** Try OLD, then NEW. That is, choose the newest existing version of the file, if any; else choose version 1. This usually only makes sense if you are intending to open \( X \) for access BOTH.

\( RECOG = \text{NIL} \) defaults to OLD. For all other values of \( RECOG \), generates an error \( \text{ILLEGAL ARG} \).

If \( X \) already contains a version number, the \( RECOG \) argument will never change it. In particular, \( RECOG = \text{NEW} \) does not require
that the file actually be new. For example, (FULLNAME 'FOO.;2 'NEW) may return (ERIS)<LISP>FOO.;2 if that file already exists, even though (FULLNAME 'FOO 'NEW) would default the version to a new number, perhaps returning (ERIS)<LISP>FOO.;5.

(INFILEP FILE) [Function]
Equivalent to (FULLNAME FILE 'OLD). That is, returns the full file name of the newest version of FILE if FILE is recognized as specifying the name of an existing file that could potentially be opened for input, NIL otherwise.

(OUTFILEP FILE) [Function]
Equivalent to (FULLNAME FILE 'NEW).

Note that INFILEP, OUTFILEP and FULLNAME do not open any files; they are pure predicates. In general they are also only hints, as they do not necessarily imply that the caller has access rights to the file. For example, INFILEP might return non-NIL, but OPENSTREAM might fail for the same file because the file is read-protected against the user, or the file happens to be open for output by another user at the time. Similarly, OUTFILEP could return non-NIL, but OPENSTREAM could fail with a FILE SYSTEM RESOURCES EXCEEDED error.

Note also that in a shared file system, such as a remote file server, intervening file operations by another user could contradict the information returned by recognition. For example, a file that was INFILEP might be deleted, or between an OUTFILEP and the subsequent OPENSTREAM, another user might create a new version or delete the highest version, causing OPENSTREAM to open a different version of the file than the one returned by OUTFILEP. In addition, some file servers do not well support recognition of files in output context. Thus, in general, the "truth" about a file can only be obtained by actually opening the file; creators of files should rely on the name of the stream opened by OPENSTREAM, not the value returned from these recognition functions. In particular, for the reasons described earlier, programmers are discouraged from using OUTFILEP or (FULLNAME NAME 'NEW).

24.5 Using File Names Instead of Streams

In earlier implementations of Interlisp, from the days of Interlisp-10 onward, the "handle" used to refer to an open file was not a stream, but rather the file's full name, represented as a
litatom. When the file name was passed to any I/O function, it was mapped to a stream by looking it up in a list of open files. This scheme was sometimes convenient for typing in file commands at the executive, but was very poor for serious programming in two major ways. First, the mapping from file name to stream on every input/output operation is inefficient. Second, and more importantly, using the file name as the handle on an open stream means that it is not possible to have more than one stream open on a given file at once.

As of this writing, Interlisp-D is in a transition period, where it still supports the use of litatom file names as synonymous with open streams, but this use is not recommended. The remainder of this section discusses this usage of file names for the benefit of those reading older programs and wishing to convert them as necessary to work properly when this compatibility feature is removed.

24.5.1 File Name Efficiency Considerations

It is possible for a program to be seriously inefficient using a file name as a stream if the program is not using the file's full name, the name returned by OPENFILE (below). Any time that an input/output function is called with a file name other than the full file name, Interlisp must perform recognition on the partial file name in order to determine which open file is intended. Thus if repeated operations are to be performed, it is considerably more efficient to use the full file name returned from OPENFILE than to repeatedly use the possibly incomplete name that was used to open the file.

There is a more subtle problem with partial file names, in that recognition is performed on the user's entire directory, not just the open files. It is possible for a file name that was previously recognized to denote one file to suddenly denote a different file. For example, suppose a program performs (INFILE 'FOO), opening FOO.;1, and reads several expressions from FOO. Then the user interrupts the program, creates a FOO.;2 and resumes the program (or a user at another workstation creates a FOO.;2). Now a call to READ giving it FOO as its FILE argument will generate a FILE NOT OPEN error, because FOO will be recognized as FOO.;2.

24.5.2 Obsolete File Opening Functions

The following functions are now considered obsolete, but are provided for backwards compatibility:
24.5.3 Converting Old Programs

At some point in the future, the Interlisp-D file system will change so that each call to OPENSTREAM returns a distinct stream, even if a stream is already open to the specified file. This change is required in order to deal rationally with files in a multiprocessing environment.

This change will of necessity produce the following incompatibilities:

1) The functions OPENFILE, INPUT, and OUTPUT will return a STREAM, not a full file name. To make this less confusing in interactive situations, STREAMS will have a print format that reveals the underlying file's actual name.

2) A greater penalty will ensue for passing as the FILE argument to i/o operations anything other than the object returned from OPENFILE. Passing the file's name will be significantly slower than passing the stream (even when passing the "full" file name), and in the case where there is more than one stream open on the file it might even act on the wrong one.

3) OPENP will return NIL when passed the name of a file rather than a stream (the value of OPENFILE or OPENSTREAM).

Users should consider the following advice when writing new programs and editing existing programs, in order that they will continue to operate well when this change is made:

Because of the efficiency and ambiguity considerations described earlier, users have long been encouraged to use only full file
USING FILE NAMES INSTEAD OF STREAMS

names as FILE arguments to i/o operations. The "proper" way to have done this was to bind a variable to the value returned from OPENFILE and pass that variable to all i/o operations; such code will continue to work. A less proper way to obtain the full file name, but one which has to date not incurred any obvious penalty, is that which binds a variable to the result of an INFILEP and passes that to OPENFILE and all i/o operations. This has worked because INFILEP and OPENFILE both return a full file name, an invalid assumption in this future world. Such code should be changed to pass around the value of the OPENFILE, not the INFILEP.

Code that calls OPENP to test whether a possibly incomplete file name is already open should be recoded to pass to OPENP only the value returned from OPENFILE or OPENSTREAM.

Code that uses ordinary string functions to manipulate file names, and in particular the value returned from OPENFILE, should be changed to use the functions UNPACKFILENAME.STRING and PACKFILENAME.STRING. Those functions work both on file names (strings) and streams (coercing the stream to the name of its file).

Code that tests the value of OUTPUT for equality to some known file name or T should be examined carefully and, if possible, recoded.

To see more directly the effects of passing around STREAMs instead of file names, replace your calls to OPENFILE with calls to OPENSTREAM. OPENSTREAM is called in exactly the same way, but returns a STREAM. Streams can be passed to READ, PRINT, CLOSEF, etc just as the file's full name can be currently, but using them is more efficient. The function FULLNAME, when applied to a stream, returns its full file name.

24.6 Using Files with Processes

Because Interlisp-D does not yet support multiple streams per file, problems can arise if different processes attempt to access the same file. The user has to be careful not to have two processes manipulating the same file at the same time, since the two processes will be sharing a single input stream and file pointer. For example, it will not work to have one process TCOMPL a file while another process is running LISTFILES on it.
24.7 File Attributes

Any file has a number of "file attributes", such as the read date, protection, and bytesize. The exact attributes that a file can have is dependent on the file device. The functions GETFILEINFO and SETFILEINFO allow the user to conveniently access file attributes:

(GETFILEINFO FILE ATTRIB) [Function]

Returns the current setting of the ATTRIB attribute of FILE.

(SETFILEINFO FILE ATTRIB VALUE) [Function]

Sets the attribute ATTRIB of FILE to be VALUE. SETFILEINFO returns T if it is able to change the attribute ATTRIB, and NIL if unsuccessful, either because the file device does not recognize ATTRIB or because the file device does not permit the attribute to be modified.

The FILE argument to GETFILEINFO and SETFILEINFO can be an open stream (or an argument designating an open stream, see page 25.2), or the name of a closed file. SETFILEINFO in general requires write access to the file.

The attributes recognized by GETFILEINFO and SETFILEINFO fall into two categories: permanent attributes, which are properties of the file, and temporary attributes, which are properties only of an open stream to the file. The temporary attributes are only recognized when FILE designates an open stream; the permanent attributes are usually equally accessible for open and closed files. However, some devices are willing to change the value of certain attributes of an open stream only when specified in the PARAMETERS argument to OPENSTREAM (page 24.2), not on a later call to SETFILEINFO.

The following are currently recognized as permanent attributes of a file:

- BYTESIZE: The byte size of the file. Interlisp-D currently only supports byte size 8.
- LENGTH: The number of bytes in the file. Alternatively, the byte position of the end-of-file. Like (GETEOFPTR FILE), but FILE does not have to be open.
- SIZE: The size of FILE in pages.
- CREATIONDATE: The date and time, as a string, that the content of FILE was "created". The creation date changes whenever the content of the file is modified, but remains unchanged when a file is transported, unmodified, across file systems. Specifically, COPYFILE and RENAMEFILE (page 24.31) preserve the file's creation date. Note that this is different from the concept of "creation date" used by some operating systems (e.g., Tops20).
WRITE DATE The date and time, as a string, that the content of FILE was last written to this particular file system. When a file is copied, its creation date does not change, but its write date becomes the time at which the copy is made.

READDATE The date and time, as a string, that FILE was last read, or NIL if it has never been read.

ICREATIONDATE IWRITEDATE IREADDATE The CREATIONDATE, WRITEDATE and READDATE, respectively, in integer form, as IDATE (page 12.14) would return. This form is useful for comparing dates.

AUTHOR The name of the user who last wrote the file.

TYPE The "type" of the file, some indication of the nature of the file's content. The "types" of files allowed depends on the file device. Most devices recognize the litatom TEXT to mean that the file contains just characters, or BINARY to mean that the file contains arbitrary data.

Some devices support a wider range of file types that distinguish among the various sorts of files one might create whose content is "binary". All devices interpret any value of TYPE that they do not support to be BINARY. Thus, GETFILEINFO may return the more general value BINARY instead of the original type that was passed to SETFILEINFO or OPENSTREAM. Similarly, COPYFILE, while attempting to preserve the TYPE of the file it is copying, may turn, say, an INTERPRESS file into a mere BINARY file.

The way in which some file devices (e.g., Xerox file servers) support a wide range of file types is by representing the type as an integer, whose interpretation is known by the client. The variable FILING.TYPES is used to associate symbolic types with numbers for these devices. This list initially contains some of the well-known assignments of type name to number; the user can add additional elements to handle any private file types. For example, suppose there existed an NS file type MAZEFILE with numeric value 5678. You could add the element (MAZEFILE 5678) to FILING.TYPES and then use MAZEFILE as a value for the TYPE attribute to SETFILEINFO or OPENSTREAM. Other devices are, of course, free to store TYPE attributes in whatever manner they wish, be it numeric or symbolic. FILING.TYPES is merely considered the official registry for Xerox file types.

For most file devices, the TYPE of a newly created file, if not specified in the PARAMETERS argument to OPENSTREAM, defaults to the value of DEFAULTFILETYPE, initially TEXT.

The following are currently recognized as temporary attributes of an open stream:
STREAMS AND FILES

FILE ATTRIBUTES

ACCESS
The current access rights of the stream (see page 24.2). Can be one of INPUT, OUTPUT, BOTH, APPEND; or NIL if the stream is not open.

ENDOFSTREAMOP
The action to be taken when a stream is at "end of file" and an attempt is made to take input from it. The value of this attribute is a function of one argument, the stream. The function can examine the stream and its calling context and take any action it wishes. If the function returns normally, its should return either T, meaning to try the input operation again, or the byte that BIN would have returned had there been more bytes to read. Ordinarily, one should not let the ENDOFSTREAMOP function return unless one is only performing binary input from the file, since there is no way in general of knowing in what state the reader was at the time the end of file occurred, and hence how it will interpret a single byte returned to it.

The default ENDOFSTREAMOP is a system function that causes the error END OF FILE. The behavior of that error can be further modified for a particular stream by using the EOF option of WHENCLOSE (page 24.20).

EOL
The end-of-line convention for the stream. This can be CR, LF, or CRLF, indicating with what byte or sequence of bytes the "End Of Line" character is represented on the stream. On input, that sequence of bytes on the stream is read as (CHARCODE EOL) by READCCODE or the string reader. On output, (TERPRI) and (PRINTCCODE (CHARCODE EOL)) cause that sequence of bytes to be placed on the stream.

The end of line convention is usually not apparent to the user. The file system is usually aware of the convention used by a particular remote operating system, and sets this attribute accordingly. If you believe a file actually is stored with a different convention than the default, it is possible to modify the default behavior by including the EOL attribute in the PARAMETERS argument to OPENSTREAM.

BUFFERS
Value is the number of 512-byte buffers that the stream maintains at one time. This attribute is only used by certain random-access devices (currently, the local disk, floppy, and Leaf servers); all others ignore it.

Streams open to files generally maintain some portion of the file buffered in memory, so that each call to an I/O function does not require accessing the actual file on disk or a file server. For files being read or written sequentially, not much buffer space is needed, since once a byte is read or written, it will never need to be seen again. In the case of random access streams, buffering is more complicated, since a program may jump around in the file, using SETFILEPTR (page 25.19). In this case, the more buffer space the stream has, the more likely it is that after a SETFILEPTR to a place in the file that has already been accessed, the stream
still has that part of the file buffered and need not go out to the
device again. This benefit must, of course, be traded off against
the amount of memory consumed by the buffers.

### 24.8 Closing and Reopening Files

The function `WHENCLOSE` permits the user to associate certain
operations with open streams that govern how and when the
stream will be closed. The user can specify that certain functions
will be executed before `CLOSEF` closes the stream and/or after
`CLOSEF` closes the stream. The user can make a particular stream
be invisible to `CLOSEALL`, so that it will remain open across user
invocations of `CLOSEALL`.

```
(WHENCLOSE FILE PROP1 VAL1 ... PROPN VALN)
```

*FILE* must designate an open stream other than T (NIL defaults to
the primary input stream, if other than T, or primary output
stream if other than T). The remaining arguments specify
properties to be associated with the full name of *FILE*. 
`WHENCLOSE` returns the full name of *FILE* as its value.

`WHENCLOSE` recognizes the following property names:

- **BEFORE** `VAL` is a function that `CLOSEF` will apply to the stream just before
  it is closed. This might be used, for example, to copy information
  about the file from an in-core data structure to the file just
  before it is closed.

- **AFTER** `VAL` is a function that `CLOSEF` will apply to the stream just after it
  is closed. This capability permits in-core data structures that
  know about the stream to be cleaned up when the stream is
  closed.

- **CLOSEALL** `VAL` is either YES or NO and determines whether *FILE* will be
  closed by `CLOSEALL` (YES) or whether `CLOSEALL` will ignore it
  (NO). `CLOSEALL` uses `CLOSEF`, so that any AFTER functions will
  be executed if the stream is in fact closed. Files are initialized
  with `CLOSEALL` set to YES.

- **EOF** `VAL` is a function that will be applied to the stream when an
  end-of-file error occurs, and the ERROTYPEPEST entry for that
  error, if any, returns NIL. The function can examine the context
  of the error, and can decide whether to close the stream,
  RETFROM some function, or perform some other computation. If
  the function supplied returns normally (i.e., does not RETFROM
  some function), the normal error machinery will be invoked.

The default EOF behavior, unless overridden by this `WHENCLOSE`
option, is to call the value of DEFAULTEOFCLOSE (below).
For some applications, the ENDOFSTREAMOP attribute (page 24.19) is a more useful way to intercept the end-of-file error. The ENDOFSTREAMOP attribute comes into effect before the error machinery is ever activated.

Multiple AFTER and BEFORE functions may be associated with a file; they are executed in sequence with the most recently associated function executed first. The CLOSEALL and EOF values, however, will override earlier values, so only the last value specified will have an effect.

**DEFAULTEOFCLOSE [Variable]**

Value is the name of a function that is called by default when an end of file error occurs and no EOF option has been specified for the stream by WHENCLOSE. The initial value of DEFAULTEOFCLOSE is NILL, meaning take no special action (go ahead and cause the error). Setting it to CLOSEF would cause the stream to be closed before the rest of the error machinery is invoked.

---

### 24.9 Local Hard Disk Device

**Warning:** This section describes the Interlisp-D functions that control the local hard disk drive available on some computers. All of these functions may not work on all computers running Interlisp-D. For more information on using the local hard disk facilities, see the users guide for your computer.

This section describes the local file system currently supported on the Xerox 1108 and 1186 computers. The Xerox 1132 supports a simpler local file system. The functions below are no-ops on the Xerox 1132, except for DISKPARTITION (which returns a disk partition number), and DISKFREEPAGES. On the Xerox 1132, different numbered partitions are referenced by using devices such as {DSK1}, {DSK2}, etc. {DSK} always refers to the disk partition that Interlisp is running on. The 1132 local file system does not support the use of directories.

The hard disk used with the Xerox 1108 or 1186 may be partitioned into a number of named "logical volumes." Logical volumes may be used to hold the Interlisp virtual memory file (see page 12.6), or Interlisp files. For information on initializing and partitioning the hard disk, see the users guide for your computer. In order to store Interlisp files on a logical volume, it is necessary to create a lisp file directory on that volume (see CREATEDSKDIRECTORY, below).

So long as there exists a logical volume with a Lisp directory on it, files on this volume can be accessed by using the file device called {DSK}. Interlisp-D can be used to read, write, and otherwise
interact with files on local disk disks through standard Interlisp input/output functions. All I/O functions such as LOAD, OPENSTREAM, READ, PRINT, GETFILEINFO, COPYFILE, etc., work with files on the local disk.

If you do not have a logical volume with a Lisp directory on it, Interlisp emulates the {DSK} device by a core device, a file device whose backing store is entirely within the Lisp virtual memory. However, this is not recommended because the core device only provides limited scratch space, and since the core device is contained in virtual memory, it (and the files stored on it) will be erased when the virtual memory file is reloaded.

Each logical volume with a Lisp directory on it serves as a directory of the device {DSK}. Files are referred to by forms such as

{DSK}<{VOLUMENAME}>FILENAME

Thus, the file INIT.LISP on the volume LISPFILES would be called {DSK}<{LISPFILES}>INIT.LISP.

Subdirectories within a logical volume are supported, using the > character in file names to delimit subdirectory names. For example, the file name {DSK}<{LISPFILES}>DOC>DESIGN.TEDIT designates the file names DESIGN.TEDIT on the subdirectory DOC on the logical volume LISPFILES.

If a logical volume name is not specified, it defaults in an unusual but simple way: the logical volume defaults to the next logical volume that has a Lisp file directory on it including or after the volume containing the currently running virtual memory. For example, if the local disk has the logical volumes LISP, TEMP, and LISPFILES, the LISP volume contains the running virtual memory, and only the LISP volume has a Lisp file directory on it, then {DSK}INIT.LISP refers to the file {DSK}<{LISPFILES}>INIT.LISP. All the functions below default logical volume names in a similar way, except for those such as CREATEDSKDIRECTORY. To determine the current default Lisp file directory, evaluate (DIRECTORYNAME '(DSK)).

(CREATEDSKDIRECTORY VOLUMENAME —) [Function]

Creates a Lisp file directory on the logical volume VOLUMENAME, and returns the name of the directory created. It is only necessary to create a Lisp file directory the first time the logical volume is used. After that, the system automatically recognizes and opens access to the logical volumes that have Lisp file directories on them.

(PURGEDSKDIRECTORY VOLUMENAME —) [Function]

Erases all Lisp files on the volume VOLUMENAME, and deletes the Lisp file directory.
(LISPDIRECTORYP VOLUMENAME)  [Function]
Returns T if the logical volume VOLUMENAME has a lisp file directory on it.

(VOLUMES)  [Function]
Returns a list of the names of all of the logical volumes on the local hard disk (whether they have lisp file directories or not).

(VOLUMESIZE VOLUMENAME —)  [Function]
Returns the total size of the logical volume VOLUMENAME in disk pages.

(DISKFREEPAGES VOLUMENAME —)  [Function]
Returns the total number of free disk pages left on the logical volume VOLUMENAME.

(DISKPARTITION)  [Function]
Returns the name of the logical volume containing the virtual memory file that Interlisp is currently running in (see page 12.6).

(DSKDISPLAY NEWSTATE)  [Function]
Controls a display window that displays information about the logical volumes on the local hard disk (logical volume names, sizes, free pages, etc.). DSKDISPLAY opens or closes this display window depending on the value of NEWSTATE (one of ON, OFF, or CLOSED), and returns the previous state of the display window.

If NEWSTATE is ON, the display window is opened, and it is automatically updated whenever the file system state changes (this can slow file operations significantly). If NEWSTATE is OFF, the display window is opened, but it is not automatically updated. If NEWSTATE is CLOSED, the display window is closed. The display mode is initially set to CLOSED.

Once the display window is open, the user can update it or change its state with the mouse. Left-buttoning the display window updates it, and middle-buttoning the window brings up a menu that allows you to change the display state.

Note: DSKDISPLAY uses the value of the variable DSKDISPLAY.POSITION for the position of the lower-left corner of the disk display window when it is opened. This variable is changed if the disk display window is moved.

(SCAVENGEDSKDIRECTORY VOLUMENAME SILENT)  [Function]
Rebuilds the lisp file directory for the logical volume VOLUMENAME. This may repair damage in the unlikely event of
file system failure, signified by symptoms such as infinite looping or other strange behavior while the system is doing a directory search. Calling SCAVENGEDSKDIRECTORY will not harm an intact volume.

Normally, SCAVENGEDSKDIRECTORY prints out messages as it scavenges the directory. If SILENT is non-NIL, these messages are not printed.

Note: Some low-level disk failures may cause "HARD DISK ERROR" errors to occur. To fix such a failure, it may be necessary to log out of Interlisp, scavenge the logical volume in question using Pilot tools, and then call SCAVENGEDSKDIRECTORY from within Interlisp. See the users guide for your computer for more information.

### 24.10 Floppy Disk Device

**Warning:** This section describes the Interlisp-D functions that control the floppy disk drive available on some computers. All of these functions may not work on all computers running Interlisp-D. For more information on using the floppy disk facilities, see the users guide for your computer.

The floppy disk drive is accessed through the device {FLOPPY}. Interlisp-D can be used to read, write, and otherwise interact with files on floppy disks through standard Interlisp input/output functions. All I/O functions such as LOAD, OPENSTREAM, READ, PRINT, GETFILEINFO, COPYFILE, etc., work with files on floppies.

Note that floppy disks are a removable storage medium. Therefore, it is only meaningful to perform i/o operations to the floppy disk drive, rather than to a given floppy disk. In this section, the phrase "the floppy" is used to mean "the floppy that is currently in the floppy disk drive."

For example, the following sequence could be used to open a file XXX.TXT on the floppy, print "Hello" on it, and close it:

```
(SETQ XXX (OPENSTREAM '{FLOPPY}XXX.TXT 'OUTPUT 'NEW)
(PRINT "Hello" XXX)
(CLOSEF XXX)
```

**Function**

**FLOPPY.MODE MODE**

Interlisp-D can currently read and write files on floppies stored in a number of different formats. At any point, the floppy is considered to be in one of four "modes," which determines how it reads and writes files on the floppy. FLOPPY.MODE sets the floppy mode to the value of MODE, one of PILOT, HIPEPILOT, SYSOUT, or CPM, and returns the previous floppy mode. The floppy modes are interpreted as follows:
PILOT  This is the normal floppy mode, using floppies in the Xerox Pilot floppy disk format. This file format allows all of the normal Interlisp-D I/O operations. This format also supports file names with arbitrary levels of subdirectories. For example, it is possible to create a file named `FLOPPY\Lisp\Project\FOO.TXT`.

HUGEPILOT  This floppy mode is used to access files that are larger than a single floppy, stored on multiple floppies. There are some restrictions with using "huge" files. Some I/O operations are not meaningful for "huge" files. When a stream is created for output in this mode, the LENGTH file attribute (page 24.17) must be specified when the file is opened, so that it is known how many floppies will be needed. When an output file is created, the floppy (or floppies) are automatically erased and reformatted (after confirmation from the user).

HUGEPILOT mode is primarily useful for saving big files to and from floppies. For example, the following could be used to copy the file `{ERIS}\Lisp\Bigfile.txt` onto the huge Pilot file `{FLOPPY}\BigFile.save`:

```
(FLOPPY.MODE 'HUGEPILOT)
(COPYFILE '{ERIS}\Lisp\Bigfile.txt '{FLOPPY}\BigFile.save)
```

and the following would restore the file:

```
(FLOPPY.MODE 'HUGEPILOT)
(COPYFILE '{FLOPPY}\BigFile.save '{ERIS}\Lisp\Bigfile.txt)
```

During each copying operation, the user will be prompted to insert "the next floppy" if `{ERIS}\Lisp\Bigfile.txt` takes multiple floppies.

SYSOUT  Similar to HUGEPILOT mode, SYSOUT mode is used for storing sysout files (page 12.8) on multiple floppy disks. The user is prompted to insert new floppies as they are needed.

This mode is set automatically when SYSOUT or MAKESYS is done to the floppy device: `(SYSOUT '{FLOPPY})` or `(MAKESYS '{FLOPPY})`. Notice that the file name does not need to be specified in SYSOUT mode; unlike HUGEPILOT mode, the file name Lisp.sysout is always used.

Note: The procedure for loading sysout files from floppies depends on the particular computer being used. For information on loading sysout files from floppies, see the users guide for your computer.

Explicitly setting the mode to SYSOUT is useful when copying a sysout file to or from floppies. For example, the following can be used to copy the sysout file `{ERIS}\Lisp\Lisp.sysout` onto floppies (it is important to set the floppy mode back when done):

```
(FLOPPY.MODE 'SYSOUT)
(COPYFILE '{ERIS}\Lisp\Lisp.sysout '{FLOPPY})
```
(FLOPPY.MODE 'PILOT)

CPM Interlisp-D supports the single-density single-sided (SDSS) CPM floppy format (a standard used by many computers). CPM-formatted floppies are totally different than Pilot floppies, so the user should call FLOPPY.MODE to switch to CPM mode when planning to use CPM floppies. After switching to CPM mode, FLOPPY.FORMAT can be used to create CPM-formatted floppies, and the usual input/output operations work with CPM floppy files.

Note: There are a few limitations on CPM floppy format files: (1) CPM file names are limited to eight or fewer characters, with extensions of three or fewer characters; (2) CPM floppies do not have directories or version numbers; and (3) CPM files are padded out with blanks to make the file lengths multiples of 128.

(FLOPPY.FORMAT NAME AUTOCONFIRMFLG SLOWFLG) [Function]

FLOPPY.FORMAT erases and initializes the track information on a floppy disk. This must be done when new floppy disks are to be used for the first time. This can also be used to erase the information on used floppy disks.

NAME should be a string that is used as the name of the floppy (106 characters max). This name can be read and set using FLOPPY.NAME (below).

If AUTOCONFIRMFLG is NIL, the user will be prompted to confirm erasing the floppy, if it appears to contain valid information. If AUTOCONFIRMFLG is T, the user is not prompted to confirm.

If SLOWFLG is NIL, only the Pilot records needed to give your floppy an empty directory are written. If SLOWFLG is T, FLOPPY.FORMAT will completely erase the floppy, writing track information and critical Pilot records on it. SLOWFLG should be set to T when formatting a brand-new floppy.

Note: Formatting a floppy is a very compute-intensive operation for the I/O hardware. Therefore, the cursor may stop tracking the mouse and keystrokes may be lost while formatting a floppy. This behavior goes away when the formatting is finished.

Warning: The floppy mode set by FLOPPY.MODE (above) affects how FLOPPY.FORMAT formats the floppy. If the floppy is going to be used in Pilot mode, it should be formatted under (FLOPPY.MODE 'PILOT). If it is to be used as a CPM floppy, it should be formatted under (FLOPPY.MODE 'CPM). The two types of formatting are incompatible.
(FLOPPY.NAME NAME) [Function]
If NAME is NIL, returns the name stored on the floppy disk. If NAME is non-NIL, then the name of the floppy disk is set to NAME.

(FLOPPY.FREE.PAGES) [Function]
Returns the number of unallocated free pages on the floppy disk in the floppy disk drive.

Note: Pilot floppy files are represented by contiguous pages on a floppy disk. If the user is creating and deleting a lot of files on a floppy, it is advisable to keep such a floppy less than 75 percent full.

(FLOPPY.CAN.READP) [Function]
Returns non-NIL if there is a floppy in the floppy drive.

Note: FLOPPY.CAN.READP does not provide any debouncing (protection against not fully closing the floppy drive door). It may be more useful to use FLOPPY.WAIT.FOR.FLOPPY (below).

(FLOPPY.CAN.WRITEP) [Function]
Returns non-NIL if there is a floppy in the floppy drive and the floppy drive can write on this floppy.

It is not possible to write on a floppy disk if the "write-protect notch" on the floppy disk is punched out.

(FLOPPY.WAIT.FOR.FLOPPY NEWFLG) [Function]
If NEWFLG is NIL, waits until a floppy is in the floppy drive before returning.

If NEWFLG is T, waits until the existing floppy in the floppy drive, if any, is removed, then waits for a floppy to be inserted into the drive before returning.

(FLOPPY.SCAVENGE) [Function]
Attempts to repair a floppy whose critical records have become confused (causing errors when file operations are attempted). May also retrieve accidentally-deleted files, provided they haven't been overwritten by new files.

(FLOPPY.TO.FILE TOFILE) [Function]
Copies the entire contents of the floppy to the "floppy image" file TOFILE, which can be on a file server, local disk, etc. This can be used to create a centralized copy of a floppy, that different users can copy to their own floppy disks (using FLOPPY.FROM.FILE).
Note: A floppy image file for an 8-inch floppy is about 2500 pages long, regardless of the number of pages in use on the floppy.

**FLOPPY.FROM.FILE FROMFILE**  
[Function]  
Copies the "floppy image" file FROMFILE to the floppy. FROMFILE must be a file produced by FLOPPY.TO.FILE.

**FLOPPY.ARCHIVE FILES NAME**  
[Function]  
FLOPPY.ARCHIVE formats a floppy inserted into the floppy drive, giving the floppy the name NAME#1. FLOPPY.ARCHIVE then copies each file in FILES to the freshly formatted floppy. If the first floppy fills up, FLOPPY.ARCHIVE uses multiple floppies (named NAME#2, NAME#3, etc.), each time prompting the user to insert a new floppy.

The function DIRECTORY (page 24.33) is convenient for generating a list of files to archive. For example,

```
(FLOPPY.ARCHIVE
  (DIRECTORY '([ERIS]<Lisp>Project>)
    'Project)
```

will archive all files on the directory [ERIS]<Lisp>Project> to floppies (named Project#1, Project#2, etc.).

**FLOPPY.UNARCHIVE HOST/DIRECTORY**  
[Function]  
FLOPPY.UNARCHIVE copies all files on the current floppy to the directory HOST/DIRECTORY. For example, (FLOPPY.UNARCHIVE '([ERIS]<Lisp>Project>) will copy each file on the current floppy to the directory [ERIS]<Lisp>Project>. If there is more than one floppy to restore from archive, FLOPPY.UNARCHIVE should be called on each floppy disk.

### 24.11 I/O Operations to and from Strings

It is possible to treat a string as if it were the contents of a file by using the following function:

**OPENSTRINGSTREAM STR ACCESS**  
[Function]  
Returns a stream that can be used to access the characters of the string STR. ACCESS may be either INPUT, OUTPUT, or BOTH; NIL defaults to INPUT. The stream returned may be used exactly like a file opened with the same access, except that output operations may not extend past the end of the original string. Also, string streams do not appear in the value of (OPENP).
For example, after performing

\[
\text{(SETQ STRM (OPENSTRINGSTREAM "THIS 2 (IS A LIST)"))}
\]

the following succession of reads could occur:

\[
\begin{align*}
\text{(READ STRM)} & \Rightarrow \text{THIS} \\
\text{(RATOM STRM)} & \Rightarrow 2 \\
\text{(READ STRM)} & \Rightarrow \text{(IS A LIST)} \\
\text{(EOFP STRM)} & \Rightarrow T
\end{align*}
\]

Compatibility Note: In Interlisp-10 it was possible to take input from a string simply by passing the string as the \textit{FILE} argument to an input function. In order to maintain compatibility with this feature, Interlisp-D provides the same capability. This not terribly clean feature persists in the present implementation to give users time to convert old code. This means that strings are not equivalent to litatoms when specifying a file name as a stream argument (see page 24.13). In a future release, the old Interlisp-10 string-reading feature will be decommissioned, and \textit{OPENSTRINGSTREAM} will be the only way to perform I/O on a string.

---

\section{Temporary Files and the CORE Device}

Many operating systems have a notion of "scratch file", a file typically used as temporary storage for data most naturally maintained in the form of a file, rather than some other data structure. A scratch file can be used as a normal file in most respects, but is automatically deleted from the file system after its useful life is up, e.g., when the job terminates, or the user logs out. In normal operation, the user need never explicitly delete such files, since they are guaranteed to disappear soon.

A similar functionality is provided in Interlisp-D by core-resident files. Core-resident files are on the device CORE. The directory structure for this device and all files on it are represented completely within the user's virtual memory. These files are treated as ordinary files by all file operations; their only distinguishing feature is that all trace of them disappears when the virtual memory is abandoned.

Core files are opened and closed by name the same as any other file, e.g., \textit{(OPENSTREAM \{CORE\}<FOO> FILE.DCOM 'OUTPUT)}. Directory names are completely optional, so files can also have names of the form \{CORE\}NAME.EXT. Core files can be enumerated by \textit{DIRECTORY} (page 24.33). While open, they are registered in \textit{(OPENP)}. They do consume virtual memory space, which is only reclaimed when the file is deleted. Some caution
should thus be used when creating large CORE files. Since the virtual memory of an Interlisp-D workstation usually persists far longer than the typical process on a mainframe computer, it is still important to delete CORE files after they are no longer in use.

For many applications, the name of the scratch file is irrelevant, and there is no need for anyone to have access to the file independent of the program that created it. For such applications, NODIRCORE files are preferable. Files created on the device lisp NODIRCORE are core-resident files that have no name and are registered in no directory. These files "disappear", and the resources they consume are reclaimed, when all pointers to the file are dropped. Hence, such files need never be explicitly deleted or, for that matter, closed. The "name" of such a file is simply the stream object returned from (OPENSTREAM '{NODIRCORE} 'OUTPUT), and it is this stream object that must be passed to all input/output operations, including CLOSEF and any calls to OPENSTREAM to reopen the file.

(COREDEVICE NAME NODIRFLG) [Function]

Creates a new device for core-resident files and assigns NAME as its device name. Thus, after performing (COREDEVICE 'FOO), one can execute (OPENSTREAM '{FOO}BAR 'OUTPUT) to open a file on that device. Interlisp-D is initialized with the single core-resident device named CORE, but COREDEVICE may be used to create any number of logically distinct core devices.

If NODIRFLG is non-NIL, a core device that acts like {NODIRCORE} is created.

Compatibility note: In Interlisp-10, it was possible to create scratch files by using file names with suffixes ;S or ;T. In Interlisp-D, these suffixes in file names are simply ignored when output is directed to a particular host or device. However, the function PACKFILENAME.STRING is defined to default the device name to CORE if the file has the TEMPORARY attribute and no explicit host is provided.

24.13 NULL Device

The NULL device provides a source of content-free "files". (OPENSTREAM '{NULL} 'OUTPUT) creates a stream that discards all output directed at it. (OPENSTREAM '{NULL} 'INPUT) creates a stream that is perpetually at end-of-file (i.e., has no input).
24.15 Deleting, Copying, and Renaming Files

(DELFILE FILE) [Function]
Deletes FILE if possible. The file must be closed. Returns the full name of the file if deleted, else NIL. Recognition mode for FILE is OLDEST, i.e., if FILE does not have a version number specified, then DELFILE deletes the oldest version of the file.

(COPYFILE FROMFILE TOFILE) [Function]
Copies FROMFILE to a new file named TOFILE. The source and destination may be on any combination of hosts/devices. COPYFILE attempts to preserve the TYPE and CREATIONDATE where possible. If the original file's file type is unknown, COPYFILE attempts to infer the type (file type is BINARY if any of its 8-bit bytes have their high bit on).

COPYFILE uses COPYCHARS (page 25.20) if the source and destination hosts have different EOL conventions. Thus, it is possible for the source and destination files to be of different lengths.

(RENAMEFILE OLDFILE NEWFILE) [Function]
Renames OLDFILE to be NEWFILE. Causes an error, FILE NOT FOUND if FILE does not exist. Returns the full name of the new file, if successful, else NIL if the rename cannot be performed.

If OLDFILE and NEWFILE are on the same host/device, and the device implements a renaming primitive, RENAMEFILE can be very fast. However, if the device does not know how to rename files in place, or if OLDFILE and NEWFILE are on different devices, RENAMEFILE works by copying OLDFILE to NEWFILE and then deleting OLDFILE.

24.16 Searching File Directories

DIRECTORIES [Variable]
Global variable containing the list of directories searched (in order) by SPELLFILE and FINDFILE (below) when not given an explicit DIRLIST argument. In this list, the atom NIL stands for the login directory (the value of LOGINHOST/DIR), and the atom T stands for the currently connected directory. Other elements should be full directory specifications, e.g., {TWENTY}PS:<LISPUSERS>, not merely LISPUSERS.
**LISPU SERSDI RECTORI ES**

[Variable]

Global variable containing a list of directories to search for "library" package files. Used by the FILES file package command (page 17.39).

**SPELLFILE FILE NOPRINTFLG NSFLG DIRST**

[Function]

Searches for the file name FILE, possibly performing spelling correction (see page 20.15). Returns the corrected file name, if any, otherwise NIL.

If FILE has a directory field, SPELLFILE attempts spelling correction against the files in that particular directory. Otherwise, SPELLFILE searches for the file on the directory list DIRST before attempting any spelling correction.

If NOPRINTFLG is NIL, SPELLFILE asks the user to confirm any spelling correction done, and prints out any files found, even if spelling correction is not done. If NOPRINTFLG = T, SPELLFILE does not do any printing, nor ask for approval.

If NSFLG = T (or NOSPELLFLG = T, see page 20.13), no spelling correction is attempted, though searching through DIRST still occurs.

DIRST is the list of directories searched if FILE does not have a directory field. If DIRST is NIL, the value of the variable DIRECTORIES is used.

Note: If DIRST is NIL, and FILE is not found by searching the directories on DIRECTORIES, but the root name of FILE has a FILEDATES property (page 17.20) indicating that a file by that name has been loaded, then the directory indicated in the FILEDATES property is searched, too. This additional search is not done if DIRST is non-NIL.

**ERRORTYPELST** (page 14.22) initially contains the entry ((23 (SPELLFILE (CADR ERRORMESS) NIL NOFILESPELLFLG)), which causes SPELLFILE to be called in case of a FILE NOT FOUND error. If the variable NOFILESPELLFLG is T (its initial value), then spelling correction is not done on the file name, but DIRECTORIES is still searched. If SPELLFILE is successful, the operation will be reexecuted with the new (corrected) file name.

**FINDFILE FILE NSFLG DIRST**

[Function]

Uses SPELLFILE to search for a file named FILE. If it finds one, returns its full name, with no user interaction. Specifically, it calls (SPELLFILE FILE T NSFLG DIRST), after first performing two simple checks: If FILE has an explicit directory, it checks to see if a file so named exists, and if so returns that file. If DIRST is NIL, it looks for FILE on the connected directory before calling SPELLFILE.
24.17 Listing File Directories

The function DIRECTORY allows the user to conveniently specify and/or program a variety of directory operations:

\[
\text{(DIRECTORY FILES COMMANDS DEFAULTEXT DEFAULTVERS)} \quad \text{[Function]}
\]

Returns, lists, or performs arbitrary operations on all files specified by the "file group" FILES. A file group has the form of a regular file name, except that the character * can be used to match any number of characters, including zero, in the file name. For example, the file group A*B matches all file names beginning with the character A and ending with the character B. The file group *.DCOM matches all files with an extension of DCOM.

If FILES does not contain an explicit extension, it is defaulted to DEFAULTEXT; if FILES does not contain an explicit version, it is defaulted to DEFAULTVERS. DEFAULTEXT and DEFAULTVERS themselves default to *. If the period or semicolon preceding the omitted extension or version, respectively, is present, the field is explicitly empty and no default is used. All other unspecified fields default to *.

Null version is interpreted as "highest". Thus FILES = * or *. or *. or *;* enumerates all files on the connected directory; FILES = *. or *.;* enumerates all versions of files with null extension; FILES = *.;* enumerates the highest version of files with null extension; and FILES = *.;* enumerates the highest version of all files. If FILES is NIL, it defaults to *;*.

Note: Some hosts/devices are not capable of supporting "highest version" in enumeration. Such hosts instead enumerate all versions.

For each file that matches the file group FILES, the "file commands" in COMMANDS are executed in order. Some of the file commands allow aborting the command processing for a given file, effectively filtering the list of files. The interpretation of the different file commands is described below. If COMMANDS is NIL, it defaults to (COLLECT), which collects the matching file names in a list and returns it as the value of DIRECTORY.

The "file commands" in COMMANDS are interpreted as follows:

- **P** Prints the file's name. For readability, DIRECTORY strips the directory from the name, printing it once as a header in front of each set of consecutive files on the same directory.
- **PP** Prints the file's name without a version number.
- **a string** Prints the string.
READDATE, WRIITEDATE
CREATIONDATE, SIZE
LENGTH, BYTESIZE
PROTECTION, AUTHOR

TYPE Prints the appropriate information returned by GETFILEINFO
(page 24.17).

COLLECT Adds the full name of this file to an accumulating list, which will
be returned as the value of DIRECTORY.

COUNTSIZE Adds the size of this file to an accumulating sum, which will be
returned as the value of DIRECTORY.

DELETE Deletes the file.

DELIVER If this file is not the highest version of files by its name, delete it.

PAUSE Waits until the user types any char before proceeding with the
rest of the commands (good for display if you want to ponder).

The following commands are predicates to filter the list. If the
predicate is not satisfied, then processing for this file is aborted
and no further commands (such as those above) are executed for
this file.

Note: if the P and PP commands appear in COMMANDS ahead of
any of the filtering commands below except PROMPT, they are
postponed until after the filters. Thus, assuming the caller has
placed the attribute options after the filters as well, no printing
occurs for a file that is filtered out. This is principally so that
functions like DIR (below) can both request printing and pass
arbitrary commands through to DIRECTORY, and have the
printing happen in the appropriate place.

PROMPT MESS Prompts with the yes/no question MESS; if user responds with
No, abort command processing for this file.

OLDERTHAN N Continue command processing if the file hasn’t been referenced
(read or written) in N days. N can also be a string naming an
explicit date and time since which the file must not have been
referenced.

NEWERTHAN N Continue command processing if the file has been written within
the last N days. N can also be a string naming an explicit date
and time. Note that this is not quite the complement of
OLDERTHAN, since it ignores the read date.

BY USER Continue command processing if the file was last written by the
given user, i.e., its AUTHOR attribute matches (case insensitively)
USER.

@ X X is either a function of one argument (FILENAME), or an
arbitrary expression which uses the variable FILENAME freely. If
X returns NIL, abort command processing for this file.
The following two commands apply not to any particular file, but globally to the manner in which directory information is printed.

**OUT FILE**  Directs output to *FILE*.

**COLUMNS N**  Attempts to format output in *N* columns (rather than just 1).

**DIRECTORY** uses the variable **DIRCOMMANDS** as a spelling list to correct spelling and define abbreviations and synonyms (see page 20.15). Currently the following abbreviations are recognized:

- **AU**  =>  **AUTHOR**
- **-**  =>  **PAUSE**
- **COLLECT?**  =>  **PROMPT " ? " COLLECT**
- **DA**
- **DATE**  =>  **CREATIONDATE**
- **TI**  =>  **WRITEDATE**
- **DEL**  =>  **DELETE**
- **DEL?**
- **DELETE?**  =>  **PROMPT " delete? " DELETE**
- **OLD**  =>  **OLDERTHAN 90**
- **PR**  =>  **PROTECTION**
- **SI**  =>  **SIZE**
- **VERBOSE**  =>  **AUTHOR CREATIONDATE SIZE READDATE WRITEDATE**

**FILDIR FILEGROUP**  [Function]

Obsolete synonym of **DIRECTORY FILEGROUP**.

**DIR FILEGROUP COM₁ ... COMₙ**  [NLambda NoSpread Function]

Convenient form of **DIRECTORY** for use in type-in at the executive. Performs **(DIRECTORY 'FILEGROUP 'P COM₁ ... COMₙ)**.

**NDIR FILEGROUP COM₁ ... COMₙ**  [NLambda NoSpread Function]

Version of **DIR** that lists the file names in a multi-column format. Also, by default only lists the most recent version of files (unless **FILEGROUP** contains an explicit version).
24.18 File Servers

A file server is a shared resource on a local communications network which provides large amounts of file storage. Different file servers honor a variety of access protocols. Interlisp-D supports the following protocols: PUP-FTP, PUP-Leaf, and NS Filing. In addition, there are library packages available that support other communications protocols, such as TCP/IP and RS232.

With the exception of the RS232-based protocols, which exist only for file transfer, these network protocols are integrated into the Interlisp-D file system to allow files on a file server to be treated in much the same way files are accessed on local devices, such as the disk. Thus, it is possible to call OPENSTREAM on the file \{ERIS\}<LISP>FOO.DCOM;3 and read from it or write to it just as if the file had been on the local disk \{DSK\}<LISP>FOO.DCOM;3, rather than on a remote server named ERIS. However, the protocols vary in how much control they give the workstation over file system operations. Hence, some restrictions apply, as described in the following sections.

24.18.1 Pup File Server Protocols

There are two file server protocols in the family of Pup protocols: Leaf and FTP. Some servers support both, while others support only one of them. Interlisp-D uses whichever protocol is more appropriate for the requested operation.

Leaf is a random access protocol, so files opened using these protocols are RANDACCESSP (page 25.20), and thus most normal i/o operations can be performed. However, Leaf does not support directory enumeration. Hence, DIRECTORY cannot be used on a Leaf file server unless the server also supports FTP. In addition, Leaf does not supply easy access to a file’s attributes. INFILEP and GETFILEINFO have to open the file for input in order to obtain their information, and hence the file’s read date will change, even though the semantics of these functions do not imply it.

FTP is a file transfer protocol that only permits sequential access to files. However, most implementations of it are considerably more efficient than Leaf. Interlisp-D uses FTP in preference to Leaf whenever the call to OPENSTREAM requests sequential access only. In particular, the functions SYSOUT and COPYFILE open their files for sequential access. If a file server supports FTP but for some reason it is undesirable for Lisp to use it, one can set the internal variable FTPAVAILABLE to NIL.

The system normally maintains a Leaf connection to a host in the background. This connection can be broken by calling...
(BREAKCONNECTION HOST). Any subsequent reference to files on that host will reestablish the connection. The principal use for this function arises when the user interrupts a file operation in such a way that the file server thinks the file is open but Lisp thinks it is closed (or not yet open). As a result, the next time Lisp tries to open the file, it gets a file busy error.

24.18.2 Xerox NS File Server Protocols

Interlisp supports file access to Xerox 803x file servers, using the Filing Protocol built on Xerox Network Systems protocols. Interlisp-D determines that a host is an NS File Server by the presence of a colon in its name, e.g., {PHYLEX:]. The general format of NS filesystem device names is {SERVERNAME:DOMAIN:ORGANIZATION}; the device specification for an 8000-series product in general includes the ClearingHouse domain and organization. If domain and organization are not supplied directly, then they are obtained from the defaults, which themselves are found by consulting the nearest ClearingHouse if the user has not defined them in an init file (page 31.8). However, note that the server name must still have a colon in it to distinguish it from other types of host names (e.g., Pup server names).

NS file servers in general permit arbitrary characters in file names. The user should be cognizant of file name quoting conventions (page 24.6), and the fact that any file name presented as a litatom needs to have characters of significance to the reader, such as space, escaped with a %. Of course, one can always present the file name as a string, in which case only the quoting conventions are important.

NS file servers support a true hierarchical file system, where subdirectories are just another kind of file, which needs to be explicitly created. In Interlisp, subdirectories are created automatically as needed: A call to OPENFILE to create a file in a non-existent subdirectory automatically creates the subdirectory; CONN to a non-existent subdirectory asks the user whether to create the directory. For those using Star software, a directory corresponds to a "File Drawer", while a subdirectory corresponds to a "File Folder".

Because of their hierarchical structure, NS directories can be enumerated to arbitrary levels. The default is to enumerate all the files (the leaves of the tree), omitting the subdirectory nodes themselves. This default can be changed by the following variable:
FILE SERVERS

FILING.ENUMERATION.DEPTH

This variable is either a number, specifying the number of levels deep to enumerate, or T, meaning enumerate to all levels. In the former case, when the enumeration reaches the specified depth, only the subdirectory name rooted at that level is listed, and none of its descendants is listed. When FILING.ENUMERATION.DEPTH is T, all files are listed, and no subdirectory names are listed. FILING.ENUMERATION.DEPTH is initially T.

Independent of FILING.ENUMERATION.DEPTH, a request to enumerate the top-level of a file server's hierarchy lists only the top level, i.e., assumes a depth of 1. For example, (DIRECTORY '{PHYLEX:}) lists exactly the top-level directories of the server PHYLEX:

NS file servers do not currently support random access. Therefore, SETFILEPTR of an NS file generally causes an error. However, GETFILEPTR returns the correct character position for open files on NS file servers. In addition, SETFILEPTR works in the special case where the file is open for input, and the file pointer is being set forward. In this case, the intervening characters are automatically read.

Even while Interlisp has no file open on an NS Server, the system maintains a "session" with the server for a while in order to improve the speed of subsequent requests to the server. While this session is open, it is possible for some nodes of the server's file system to appear "busy" or inaccessible to certain clients on other workstations (such as Star). If this happens, the following function can be used to terminate any open sessions immediately:

(BREAK.NSFILING.CONNECTION HOST)    [Function]
Closes any open connections to NS file server HOST.

24.18.3 Operating System Designations

Some of the network server protocols are implemented on more than one kind of foreign host. Such hosts vary in their conventions for logging in, naming files, representing end-of-line, etc. In order for Interlisp to communicate gracefully with all these hosts, it is necessary that the variable NETWORKOSTYPES be correctly set.

NETWORKOSTYPES

An association-list that associates a host name with its operating system type. Elements in this list are of the form (HOSTNAME .

24.38 STREAMS AND FILES
FILE SERVERS

TYPE), for example, (MAXC2 . TENEX). The operating system types currently known to Lisp are TENEX, TOPS20, UNIX, and VMS. The host names in this list should be the "canonical" host name, represented as an uppercase atom. For Pup and NS hosts, the function CANONICAL.HOSTNAME (below) can be used to determine which of several aliases of a server is the canonical name.

(CANONICAL.HOSTNAME HOSTNAME) [Function]
Returns the "canonical" name of the server HOSTNAME, or NIL if HOSTNAME is not the name of a server.

24.18.4 Logging In

Most file servers require a user name and password for access. Interlisp-D maintains an ephemeral database of user names and passwords for each host accessed recently. The database vanishes when LOGOUT, SAVEVM, SYSOUT, or MAKESYS is executed, so that the passwords remain secure from any subsequent user of the same virtual memory image. Interlisp-D also maintains a notion of the "default" user name and password, which are generally those with which the user initially logs in (on the 1132, the default user name corresponds to that displayed in the Alto executive).

When a file server for which the system does not yet have an entry in its password database requests a name and password, the system first tries the default user name and password. If the file server doesn't recognize that name/password, the system prompts the user for a name and password to use for that host. It suggests a default name:

{ERIS} Login: Green

which the user can accept by typing a carriage return, or replace the name by typing a new name or backspacing over it. Following the name, the user is prompted for a password:

{ERIS} Login: Verdi (password)

which is not echoed, terminated by another carriage return. This information is stored in the password database so that the user is prompted only once, until the database is again cleared.

Interlisp-D also prompts for password information when a protection violation occurs on accessing a directory on certain kinds of servers that support password-protected directories. Some such servers allow one to protect a file in a way that it is inaccessible to even its owner until the file's protection is changed; in such case, no password would help, and the system causes the normal PROTECTION VIOLATION error.
The user can abort a password interaction by typing the **ERROR** interrupt, initially Control-E. This generally either causes a **PROTECTION VIOLATION** error, if the password was requested in order to gain access to a protected file on an otherwise accessible server; or to act as though the server did not exist, in the case where the password was needed in order to gain any access to the server.

The following functions are useful for altering the password database:

### (LOGIN HOSTNAME FLG DIRECTORY MSG) [Function]

Forces Interlisp-D to ask for the user name and password to be used when accessing host **HOSTNAME**. Any previous login information for **HOSTNAME** is overridden. If **HOSTNAME** is **NIL**, it overrides login information for all hosts and resets the default user name and password to be those typed in by the user. The special value **HOSTNAME = NS::** is used to obtain the default user name and password for all logins for NS Servers.

If **FLG** is the atom **QUIET**, only prompts the user if there is no cached information for **HOSTNAME**.

If **DIRECTORY** is specified, it is the name of a directory on **HOSTNAME**. In this case, the information requested is the "connect" password for that directory. Connect passwords for any number of different directories on a host can be maintained.

If **MSG** is non-**NIL**, it is a message (a string) to be printed before the name and password information is requested.

**LOGIN** returns the user name with which the user completed the login.

### (SETPASSWORD HOST USER PASSWORD DIRECTORY) [Function]

Sets the values in the internal password database, exactly as if the strings **USER** and **PASSWORD** were typed in via (**LOGIN** HOST **NIL** DIRECTORY).

### (SETUSERNAME NAME) [Function]

Sets the default user name to **NAME**.

### (USERNAME FLG STRPTR PRESERVECASE) [Function]

If **FLG** = **NIL**, returns the default user name. This is the only value of **FLG** that is meaningful in Interlisp-D.

**USERNAME** returns the value as a string, unless **STRPTR** is **T**, in which case **USERNAME** returns the value as an atom. The name is returned in upper case, unless **PRESERVECASE** is true.
24.18.5 Abnormal Conditions

If Interlisp-D tries to access a file and does not get a response from the file server in a reasonable period of time, it prints a message that the file server is not responding, and keeps trying. If the file server has actually crashed, this may continue indefinitely. A control-E or similar interrupt aborts out of this state.

If the file server crashes but is restarted before the user attempts to do anything, file operations will usually proceed normally, except for a brief pause while Interlisp-D tries to reestablish any connections it had open before the crash. However, this is not always possible. For example, when a file is open for sequential output and the server crashes, there is no way to recover the output already written, since it vanished with the crash. In such cases, the system will cause an error such as Connection Lost.

LOGOUT closes any file server connections that are currently open. On return, it attempts to reestablish connections for any files that were open before logging out. If a file has disappeared or been modified, Interlisp-D reports this fact. Files that were open for sequential access generally cannot be reopened after LOGOUT.

Interlisp supports simultaneous access to the same server from different processes and permits overlapping of Lisp computation with file server operations, allowing for improved performance. However, as a corollary of this, a file is not closed the instant that CLOSEF returns; Interlisp closes the file "in the background". It is therefore very important that the user exits Interlisp via (LOGOUT), or (LOGOUT T), rather than boot the machine.

On rare occasions, the Ethernet may appear completely unresponsive, due to Interlisp having gotten into a bad state. Typing (RESTART.ETHER) will reinitialize Lisp's Ethernet driver(s), just as when the Lisp system is started up following a LOGOUT, SYSOUT, etc (see page 31.38)
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25. Input/Output Functions</td>
<td></td>
</tr>
<tr>
<td>25.1. Specifying Streams for Input/Output Functions</td>
<td>25.1</td>
</tr>
<tr>
<td>25.2. Input Functions</td>
<td>25.2</td>
</tr>
<tr>
<td>25.3. Output Functions</td>
<td>25.7</td>
</tr>
<tr>
<td>25.3.1. PRINTLEVEL</td>
<td>25.11</td>
</tr>
<tr>
<td>25.3.2. Printing numbers</td>
<td>25.13</td>
</tr>
<tr>
<td>25.3.3. User Defined Printing</td>
<td>25.16</td>
</tr>
<tr>
<td>25.3.4. Printing Unusual Data Structures</td>
<td>25.17</td>
</tr>
<tr>
<td>25.4. Random Access File Operations</td>
<td>25.18</td>
</tr>
<tr>
<td>25.5. Input/Output Operations with Characters and Bytes</td>
<td>25.22</td>
</tr>
<tr>
<td>25.6. PRINTOUT</td>
<td>25.23</td>
</tr>
<tr>
<td>25.6.1. Horizontal Spacing Commands</td>
<td>25.25</td>
</tr>
<tr>
<td>25.6.2. Vertical Spacing Commands</td>
<td>25.26</td>
</tr>
<tr>
<td>25.6.3. Special Formatting Controls</td>
<td>25.27</td>
</tr>
<tr>
<td>25.6.4. Printing Specifications</td>
<td>25.27</td>
</tr>
<tr>
<td>25.6.4.1. Paragraph Format</td>
<td>25.28</td>
</tr>
<tr>
<td>25.6.4.2. Right-Flushing</td>
<td>25.29</td>
</tr>
<tr>
<td>25.6.4.3. Centering</td>
<td>25.29</td>
</tr>
<tr>
<td>25.6.4.4. Numbering</td>
<td>25.29</td>
</tr>
<tr>
<td>25.6.5. Escaping to Lisp</td>
<td>25.30</td>
</tr>
<tr>
<td>25.6.6. User-Defined Commands</td>
<td>25.31</td>
</tr>
<tr>
<td>25.6.7. Special Printing Functions</td>
<td>25.32</td>
</tr>
<tr>
<td>25.7. READFILE and WRITEFILE</td>
<td>25.33</td>
</tr>
<tr>
<td>25.8. Read Tables</td>
<td>25.33</td>
</tr>
<tr>
<td>25.8.1. Read Table Functions</td>
<td>25.34</td>
</tr>
<tr>
<td>25.8.2. Syntax Classes</td>
<td>25.35</td>
</tr>
<tr>
<td>25.8.3. Read Macros</td>
<td>25.39</td>
</tr>
</tbody>
</table>
This chapter describes the standard I/O functions used for reading and printing characters and Interlisp expressions on files and other streams. First, the primitive input functions are presented, then the output functions, then functions for random-access operations (such as searching a file for a given stream, or changing the "next-character" pointer to a position in a file). Next, the PRINTOUT statement is documented (page 25.23), which provides an easy way to write complex output operations. Finally, read tables, used to parse characters as Interlisp expressions, are documented.

25.1 Specifying Streams for Input/Output Functions

Most of the input/output functions in Interlisp-D have an argument named STREAM or FILE, specifying on which open stream the function's action should occur (the name FILE is used in older functions that predate the concept of stream; the two should, however, be treated synonymously). The value of this argument should be one of the following:

a stream An object of type STREAM, as returned by OPENSTREAM (page 24.2) or other stream-producing functions, is always the most precise and efficient way to designate a stream argument.

T The litatom T designates the terminal input or output stream of the currently running process, controlling input from the keyboard and output to the display screen. For functions where the direction (input or output) is ambiguous, T is taken to designate the terminal output stream. The T streams are always open; they cannot be closed.

The terminal output stream can be set to a given window or display stream by using TTYDISPLAYSTREAM (page 28.29). The terminal input stream cannot be changed. For more information on terminal I/O, see page 30.1.

NIL The litatom NIL designates the "primary" input or output stream. These streams are initially the same as the terminal
input/output streams, but they can be changed by using the functions INPUT (page 25.3) and OUTPUT (page 25.8).

For functions where the direction (input or output) is ambiguous, e.g., GETFILEPTR, the argument NIL is taken to mean the primary input stream, if that stream is not identical to the terminal input stream, else the primary output stream.

- a window: Uses the display stream of the window (page 28.34). Valid for output only.
- a file name: As of this writing, the name of an open file (as a litatom) can be used as a stream argument. However, there are inefficiencies and possible future incompatibilities associated with doing so. See page page 24.13 for details.

(GETSTREAM FILE ACCESS) [Function]
Coerces the argument FILE to a stream by the above rules. If ACCESS is INPUT, OUTPUT, or BOTH, produces the stream designated by FILE that is open for ACCESS. If ACCESS = NIL, returns a stream for FILE open for any kind of input/output (see the list above for the ambiguous cases). If FILE does not designate a stream open in the specified mode, causes an error, FILE NOT OPEN.

(STREAMP X) [Function]
Returns X if X is a STREAM, otherwise NIL.

25.2 Input Functions

While the functions described below can take input from any stream, some special actions occur when the input is from the terminal (the T input stream, see page 25.1). When reading from the terminal, the input is buffered a line at a time, unless buffering has been inhibited by CONTROL (page 30.10) or the input is being read by READC or PEEKC (page 25.5). Using specified editing characters, the user can erase a character at a time, a word at a time, or the whole line. The keys that perform these editing functions are assignable via SETSYNTAX (page 25.37), with the initial settings chosen to be those most natural for the given operating system. In Interlisp-D, the initial settings are as follows: characters are deleted one at a time by Backspace; words are erased by control-W; the whole line is erased by control-Q.

On the Interlisp-D display, deleting a character or a line causes the characters to be physically erased from the screen. In
Interlisp-10, the deleting action can be modified for various types of display terminals by using DELETECONTROL (page 30.8).

Unless otherwise indicated, when the end of file is encountered while reading from a file, all input functions generate an error, END OF FILE. Note that this does not close the input file. The ENDOFSTREAMOP stream attribute (page 24.19) is useful for changing the behavior at end of file.

Most input functions have a RDTBL argument, which specifies the read table to be used for input (see page 25.33). Unless otherwise specified, if RDTBL is NIL, the primary read table is used.

If the FILE or STREAM argument to an input function is NIL, the primary input stream is used (see page 25.1).

### (INPUT FILE)

Sets FILE as the primary input stream; returns the old primary input stream. FILE must be open for input.

(INPUT) returns the current primary input stream, which is not changed.

Note: If the primary input stream is set to a file, the file's full name, rather than the stream itself, is returned. See discussion on page 24.13.

### (READ FILE RDTBL FLG)

Reads one expression from FILE. Atoms are delimited by the break and separator characters as defined in RDTBL. To include a break or separator character in an atom, the character must be preceded by the character %, e.g., AB%(C is the atom AB(C, % is the atom %, %control-K is the atom control-K. For input from the terminal, an atom containing an interrupt character can be input by typing instead the corresponding alphabetic character preceded by control-V, e.g., ↑VD for control-D.

Strings are delimited by double quotes. To input a string containing a double quote or a %, precede it by %, e.g., "AB%"C is the string AB"C. Note that % can always be typed even if next character is not "special", e.g., %A%B%C is read as ABC.

If an atom is interpretable as a number, READ creates a number, e.g., 1E3 reads as a floating point number, 1D3 as a literal atom, 1.0 as a number, 1,0 as a literal atom, etc. An integer can be input in a non-decimal radix by using syntax such as 123Q, |b10101, |s1234 (see page 7.4). The function RADIX (page 25.13), sets the radix used to print integers.

When reading from the terminal, all input is line-buffered to enable the action of the backspacing control characters, unless inhibited by CONTROL (page 30.10). Thus no characters are
actually seen by the program until a carriage-return (actually the character with terminal syntax class EOL, see page 30.6), is typed. However, for reading by \texttt{READ}, when a matching right parenthesis is encountered, the effect is the same as though a carriage-return were typed, i.e., the characters are transmitted. To indicate this, Interlisp also prints a carriage-return line-feed on the terminal. The line buffer is also transmitted to \texttt{READ} whenever an \texttt{IMMEDIATE} read macro character is typed (page 25.41).

\( \texttt{FLG} = \texttt{T} \) suppresses the carriage-return normally typed by \texttt{READ} following a matching right parenthesis. (However, the characters are still given to \texttt{READ}; i.e., the user does not have to type the carriage-return.)

\textbf{(RATOM FILE RDTBL)}

\textbf{[Function]}

Reads in one atom from \texttt{FILE}. Separation of atoms is defined by \texttt{RDTBL}. \% is also defined for \texttt{RATOM}, and the remarks concerning line-buffering and editing control characters also apply.

If the characters comprising the atom would normally be interpreted as a number by \texttt{READ}, that number is returned by \texttt{RATOM}. Note however that \texttt{RATOM} takes no special action for " whether or not it is a break character, i.e., \texttt{RATOM} never makes a string.

\textbf{(RSTRING FILE RDTBL)}

\textbf{[Function]}

Reads characters from \texttt{FILE} up to, but not including, the next break or separator character, and returns them as a string. Backspace, control-W, control-Q, control-V, and \% have the same effect as with \texttt{READ}.

Note that the break or separator character that terminates a call to \texttt{RATOM} or \texttt{RSTRING} is not read by that call, but remains in the buffer to become the first character seen by the next reading function that is called. If that function is \texttt{RSTRING}, it will return the null string. This is a common source of program bugs.

\textbf{(RATOMS A FILE RDTBL)}

\textbf{[Function]}

Calls \texttt{RATOM} repeatedly until the atom \texttt{A} is read. Returns a list of the atoms read, not including \texttt{A}.

\textbf{(RATEST FLG)}

\textbf{[Function]}

If \( \texttt{FLG} = \texttt{T} \), \texttt{RATEST} returns \texttt{T} if a separator was encountered immediately prior to the atom returned by the last \texttt{RATOM} or \texttt{READ}, \texttt{NIL} otherwise.
If FLG = NIL, RATEST returns T if last atom read by RATOM or READ was a break character, NIL otherwise.

If FLG = 1, RATEST returns T if last atom read (by READ or RATOM) contained a % used to quote the next character (as in %[ or %A%B%C), NIL otherwise.

(READC FILE RDTBL) [Function]
Reads and returns the next character, including %, "", etc, i.e., is not affected by break or separator characters. The action of READC is subject to line-buffering, i.e., READC does not return a value until the line has been terminated even if a character has been typed. Thus, the editing control characters have their usual effect. RDTBL does not directly affect the value returned, but is used as usual in line-buffering, e.g., determining when input has been terminated. If (CONTROL T) has been executed (page 30.10), defeating line-buffering, the RDTBL argument is irrelevant, and READC returns a value as soon as a character is typed (even if the character typed is one of the editing characters, which ordinarily would never be seen in the input buffer).

(PEEKC FILE —) [Function]
Returns the next character, but does not actually read it and remove it from the buffer. If reading from the terminal, the character is echoed as soon as PEEKC reads it, even though it is then "put back" into the system buffer, where backspace, control-W, etc. could change it. Thus it is possible for the value returned by PEEKC to "disagree" in the first character with a subsequent READ.

-LASTC FILE) [Function]
Returns the last character read from FILE.

(READCODE FILE RDTBL) [Function]
Returns the next character code from STREAM; thus, this operation is equivalent to, but more efficient than, (CHCON1 (READC FILE RDTBL)).

(PEEKCODE FILE —) [Function]
Returns, without consuming, the next character code from STREAM; thus, this operation is equivalent to, but more efficient than, (CHCON1 (PEEKC FILE)).

(BIN STREAM) [Function]
Returns the next byte from STREAM. This operation is useful for reading streams of binary, rather than character, data.
Note: BIN is similar to READCODE, except that BIN always reads a single byte, whereas READCODE reads a "character" that can consist of more than one byte, depending on the character and its encoding (see page 25.22).

READ, RATOM, RATOMS, PEEK, READC all wait for input if there is none. The only way to test whether or not there is input is to use READP:

(READP FILE FLG) [Function]
Returns T if there is anything in the input buffer of FILE, NIL otherwise. This operation is only interesting for streams whose source of data is dynamic, e.g., the terminal or a byte stream over a network; for other streams, such as to files, (READP FILE) is equivalent to (NOT (EOFP FILE)).

Note that because of line-buffering, READP may return T, indicating there is input in the buffer, but READ may still have to wait.

Frequently, the terminal's input buffer contains a single EOL character left over from a previous input. For most applications, this situation wants to be treated as though the buffer were empty, and so READP returns NIL in this case. However, if FLG = T, READP returns T if there is any character in the input buffer, including a single EOL. FLG is ignored for streams other than the terminal.

(EOFP FILE) [Function]
Returns true if FILE is at "end of file", i.e., the next call to an input function would cause an END OF FILE error; NIL otherwise. For randomly accessible files (page 25.18), this can also be thought of as the file pointer pointing beyond the last byte of the file. FILE must be open for (at least) input, or an error is generated, FILE NOT OPEN.

Note that EOFP can return NIL and yet the next call to READ might still cause an END OF FILE error, because the only characters remaining in the input were separators or otherwise constituted an incomplete expression. The function SKIPSEPRS (page 25.7) is sometimes more useful as a way of detecting end of file when it is known that all the expressions in the file are well formed.

(WAITFORINPUT FILE) [Function]
Waits until input is available from FILE or from the terminal, i.e. from T. WAITFORINPUT is functionally equivalent to (until (OR (READP T) (READP FILE)) do NIL), except that it does not use up machine cycles while waiting. Returns the device for which input is now available, i.e. FILE or T.
FILE can also be an integer, in which case WAITFORINPUT waits until there is input available from the terminal, or until FILE milliseconds have elapsed. Value is T if input is now available, NIL in the case that WAITFORINPUT timed out.

(SKREAD FILE REREADSTRING RDTBL) [Function]

"Skip Read". SKREAD consumes characters from FILE as if one call to READ had been performed, without paying the storage and compute cost to really read in the structure. REREADSTRING is for the case where the caller has already performed some READC's and RATOM's before deciding to skip this expression. In this case, REREADSTRING should be the material already read (as a string), and SKREAD operates as though it had seen that material first, thus setting up its parenthesis count, double-quote count, etc.

The read table RDTBL is used for reading from FILE. If RDTBL is NIL, it defaults to the value of FILERDTBL. SKREAD may have difficulties if unusual read macros (page 25.39) are defined in RDTBL. SKREAD does not recognize read macro characters in REREADSTRING, nor SPICE or INFIX read macros. This is only a problem if the read macros are defined to parse subsequent input in the stream that does not follow the normal parenthesis and string-quote conventions.

SKREAD returns %) if the read terminated on an unbalanced closing parenthesis; %] if the read terminated on an unbalanced %], i.e., one which also would have closed any extant open left parentheses; otherwise NIL.

(SKIPSEPRS FILE RDTBL) [Function]

Consumes characters from FILE until it encounters a non-separator character (as defined by RDTBL). SKIPSEPRS returns, but does not consume, the terminating character, so that the next call to READC would return the same character. If no non-separator character is found before the end of file is reached, SKIPSEPRS returns NIL and leaves the stream at end of file. This function is useful for skipping over "white space" when scanning a stream character by character, or for detecting end of file when reading expressions from a stream with no pre-arranged terminating expression.

25.3 Output Functions

Unless otherwise specified by DEFPRT (page 25.16), pointers other than lists, strings, atoms, or numbers, are printed in the form {DATATYPE} followed by the octal representation of the
address of the pointer (regardless of radix). For example, an array pointer might print as \{ARRAYP\}#43,2760. This printed representation is for compactness of display on the user’s terminal, and will not read back in correctly; if the form above is read, it will produce the litatom \{ARRAYP\}#43,2760.

Note: the term "end-of-line" appearing in the description of an output function means the character or characters used to terminate a line in the file system being used by the given implementation of Interlisp. For example, in Interlisp-D end-of-line is indicated by the character carriage-return.

Some of the functions described below have a RDTBL argument, which specifies the read table to be used for output (see page 25.33). If RDTBL is NIL, the primary read table is used.

Most of the functions described below have an argument FILE, which specifies the stream on which the operation is to take place. If FILE is NIL, the primary output stream is used (see page 25.1).

### (OUTPUT FILE)
[Function]
Sets FILE as the primary output stream; returns the old primary output stream. FILE must be open for output.

(OUTPUT) returns the current primary output stream, which is not changed.

Note: if the primary output stream is set to a file, the file’s full name, rather than the stream itself, is returned. See discussion on page 24.13.

### (PRIN1 X FILE)
[Function]
Prints X on FILE.

### (PRIN2 X FILE RDTBL)
[Function]
Prints X on FILE with %’s and ”’s inserted where required for it to read back in properly by READ, using RDTBL.

Both PRIN1 and PRIN2 print any kind of Lisp expression, including lists, atoms, numbers, and strings. PRIN1 is generally used for printing expressions where human readability, rather than machine readability, is important, e.g., when printing text rather than program fragments. PRIN1 does not print double quotes around strings, or % in front of special characters. PRIN2 is used for printing Interlisp expressions which can then be read back into Interlisp with READ; i.e., break and separator characters in atoms will be preceded by %’s. For example, the atom "()" is printed as %() by PRIN2. If the integer output radix (as set by RADIX, page 25.13) is not 10, PRIN2 prints the integer using the
input syntax for non-decimal integers (see page 7.4) but PRIN1
does not (but both print the integer in the output radix).

(PRIN3 X FILE) [Function]

(PRIN4 X FILE RDTBL) [Function]
PRIN3 and PRIN4 are the same as PRIN1 and PRIN2 respectively,
except that they do not increment the horizontal position
counter nor perform any linelength checks. They are useful
primarily for printing control characters.

(PRINT X FILE RDTBL) [Function]
Prints the expression X using PRIN2 followed by an end-of-line.
Returns X.

(PRINTCCODE CHARCODE FILE) [Function]
Outputs a single character whose code is CHARCODE to FILE.
This is similar to (PRIN1 (CHARACTER CHARCODE)), except that
numeric characters are guaranteed to print "correctly"; e.g.,
(PRINTCCODE (CHARCODE 9)) always prints "9", independent of
the setting of RADIX.

Note that PRINTCCODE may actually print more than one byte on
FILE, due to character encoding and end of line conventions;
thus, no assumptions should be made about the relative motion
of the file pointer (see GETFILEPTR, page 25.19) during this
operation.

(BOUT STREAM BYTE) [Function]
Outputs a single 8-bit byte to STREAM. This is similar to
PRINTCCODE, but for binary streams the character position in
STREAM is not updated (as with PRIN3), and end of line
conventions are ignored.

Note: BOUT is similar to PRINTCCODE, except that BOUT always
writes a single byte, whereas PRINTCCODE writes a "character"
that can consist of more than one byte, depending on the
character and its encoding (see page 25.22).

(SPACES N FILE) [Function]
Prints N spaces. Returns NIL.

(TERPRI FILE) [Function]
Prints an end-of-line character. Returns NIL.
(FRESHLINE STREAM)  [Function]
Equivalent to TERPRI, except it does nothing if it is already at the
beginning of the line. Returns T if it prints an end-of-line, NIL
otherwise.

(TAB POS MINSPACES FILE)  [Function]
Prints the appropriate number of spaces to move to position
POS. MINSPACES indicates how many spaces must be printed (if
NIL, 1 is used). If the current position plus MINSPACES is greater
than POS, TAB does a TERPRI and then (SPACES POS). If
MINSPACES is T, and the current position is greater than POS,
then TAB does nothing.

Note: A sequence of PRINT, PRIN2, SPACES, and TERPRI
expressions can often be more conveniently coded with a single
PRINTOUT statement (page 25.23).

(SHOWPRIN2 X FILE RDTBL)  [Function]
Like PRIN2 except if SYSPRETTYFLG = T, prettyprints X instead.
Returns X.

(SHOWPRINT X FILE RDTBL)  [Function]
Like PRINT except if SYSPRETTYFLG = T, prettyprints X instead,
followed by an end-of-line. Returns X.

SHOWPRINT and SHOWPRIN2 are used by the programmer's
assistant (page 13.1) for printing the values of expressions and
for printing the history list, by various commands of the break
package (page 14.1), e.g. ?= and BT commands, and various
other system packages. The idea is that by simply settling or
binding SYSPRETTYFLG to T (initially NIL), the user instructs the
system when interacting with the user to PRETTYPRINT
expressions (page 26.40) instead of printing them.

(PRINTBELLS —)  [Function]
Used by DWIM (page 20.1) to print a sequence of bells to alert
the user to stop typing. Can be advised or redefined for special
applications, e.g., to flash the screen on a display terminal.

(FORCEOUTPUT STREAM WAITFORFINISH)  [Function]
Forces any buffered output data in STREAM to be transmitted.
If WAITFORFINISH is non-NIL, this doesn't return until the data
has been forced out.
25.3.1 PRINTLEVEL

When using Interlisp one often has to handle large, complicated lists, which are difficult to understand when printed out. PRINTLEVEL allows the user to specify in how much detail lists should be printed. The print functions PRINT, PRIN1, and PRIN2 are all affected by level parameters set by:

(PRINTLEVEL CARVAL CDRVAL) [Function]

Sets the CAR print level to CARVAL, and the CDR print level to CDRVAL. Returns a list cell whose CAR and CDR are the old settings. PRINTLEVEL is initialized with the value (1000 . -1).

In order that PRINTLEVEL can be used with RESETFORM or RESETSAVE, if CARVAL is a list cell it is equivalent to (PRINTLEVEL (CAR CARVAL) (CDR CARVAL)).

(PRINTLEVEL N NIL) changes the CAR printlevel without affecting the CDR printlevel. (PRINTLEVEL NIL N) changes the CDR
printlevel with affecting the CAR printlevel. (PRINTLEVEL) gives
the current setting without changing either.

Note: control-P (page 30.2) can be used to change the
PRINTLEVEL setting dynamically, even while Interlisp is printing.

The CAR printlevel specifies how "deep" to print a list.
Specifically, it is the number of unpaired left parentheses which
will be printed. Below that level, all lists will be printed as &. If
the CAR printlevel is negative, the action is similar except that an
end-of-line is inserted after each right parentheses that would
be immediately followed by a left parenthesis.

The CDR printlevel specifies how "long" to print a list. It is the
number of top level list elements that will be printed before the
printing is terminated with --. For example, if CDRVAL = 2, (A B C
D E) will print as (A B --). For sublists, the number of list elements
printed is also affected by the depth of printing in the CAR
direction: Whenever the sum of the depth of the sublist (i.e. the
number of unmatched left parentheses) and the number of
elements is greater than the CDR printlevel, -- is printed. This
gives a "triangular" effect in that less is printed the farther one
goes in either CAR or CDR direction. If the CDR printlevel is
negative, then it is the same as if the CDR printlevel were infinite.

Examples:

After: (A (B C (D (E F) G) H) K L) prints as:
(PRTLEVEL 3 -1) (A (B C (D (E F) G) H) K L)
(PRTLEVEL 2 -1) (A (B C & H) K L)
(PRTLEVEL 1 -1) (A & K L)
(PRTLEVEL 0 -1) &
(PRTLEVEL 1000 2) (A (B --) --)
(PRTLEVEL 1000 3) (A (B C --) K --)
(PRTLEVEL 1 3) (A & K --)

PLVLFILEFLG [Variable]

Normally, PRINTLEVEL only affects terminal output. Output to
all other files acts as though the print level is infinite. However,
if PLVLFILEFLG is T (initially NIL), then PRINTLEVEL affects output
to files as well.

The following three functions are useful for printing isolated
expressions at a specified print level without going to the
overhead of resetting the global print level.
OUTPUT FUNCTIONS

(LVLPRINT X FILE CARLVL CDRLVL TAIL) [Function]
Performs PRINT of X to FILE, using as CAR and CDR print levels the values CARLVL and CDRLVL, respectively. Uses the T read table. If TAIL is specified, and X is a tail of it, then begins its printing with "...", rather than on open parenthesis.

(LVLPRIN2 X FILE CARLVL CDRLVL TAIL) [Function]
Similar to LVLPRIN2, but performs a PRIN2.

(LVLPRIN1 X FILE CARLVL CDRLVL TAIL) [Function]
Similar to LVLPRIN1, but performs a PRIN.

25.3.2 Printing numbers

How the ordinary printing functions (PRIN1, PRIN2, etc.) print numbers can be affected in several ways. RADIX influences the printing of integers, and FLTFMT influences the printing of floating point numbers. The setting of the variable PRXFLG determines how the symbol-manipulation functions handle numbers. The PRINTNUM package permits greater controls on the printed appearance of numbers, allowing such things as left-justification, suppression of trailing decimals, etc.

(RADIX N) [Function]
Resets the output radix for integers to the absolute value of N. The value of RADIX is its previous setting. (RADIX) gives the current setting without changing it. The initial setting is 10.

Note that RADIX affects output only. There is no input radix; on input, numbers are interpreted as decimal unless they are entered in a non-decimal radix with syntax such as 123Q, |b10101, |s1234 (see page 7.4). RADIX does not affect the behavior of UNPACK, etc., unless the value of PRXFLG (below) is T. For example, if PRXFLG is NIL and the radix is set to 8 with (RADIX 8), the value of (UNPACK 9) is (9), not (1 1).

Using PRINTNUM (page 25.15) or the PRINTOUT command .J (page 25.30) is often a more convenient and appropriate way to print a single number in a specified radix than to globally change RADIX.

(FLTFMT FORMAT) [Function]
Resets the output format for floating point numbers to the FLOAT format FORMAT (see PRINTNUM below for a description of FLOAT formats). FORMAT=T specifies the default "free" formatting: some number of significant digits (a function of the implementation) are printed, with trailing zeros suppressed;
numbers with sufficiently large or small exponents are instead printed in exponent notation.

FLTFMT returns its current setting. (FLTFMT) returns the current setting without changing it. The initial setting is T.

Note: In Interlisp-D, FLTFMT ignores the WIDTH and PAD fields of the format (they are implemented only by PRINTNUM).

Whether print name manipulation functions (UNPACK, NCHARS, etc.) use the values of RADIX and FLTFMT is determined by the variable PRXFLG:

**PRXFLG**  

If PRXFLG = NIL (the initial setting), then the "PRIN1" name used by PACK, UNPACK, MKSTRING, etc., is computed using base 10 for integers and the system default floating format for floating point numbers, independent of the current setting of RADIX or FLTFMT. If PRXFLG = T, then RADIX and FLTFMT do dictate the "PRIN1" name of numbers. Note that in this case, PACK and UNPACK are not inverses.

Examples with (RADIX 8), (FLTFMT '(FLOAT 4 2)):

With PRXFLG = NIL,

- (UNPACK 13) => (1 3)
- (PACK '(A 9)) => A9
- (UNPACK 1.2345) => (1 %.2345)

With PRXFLG = T,

- (UNPACK 13) => (1 5)
- (PACK '(A 9)) => A11
- (UNPACK 1.2345) => (1 %.23)

Note that PRXFLG does not effect the radix of "PRIN2" names, so with (RADIX 8), (NCHARS 9 T), which uses PRIN2 names, would return 3, (since 9 would print as 11Q) for either setting of PRXFLG.

Warning: Some system functions will not work correctly if PRXFLG is not NIL. Therefore, resetting the global value of PRXFLG is not recommended. It is much better to rebind PRXFLG as a SPECVAR for that part of a program where it needs to be non-NIL.

The basic function for printing numbers under format control is PRINTNUM. Its utility is considerably enhanced when used in conjunction with the PRINTOUT package (page 25.23), which implements a compact language for specifying complicated
sequences of elementary printing operations, and makes fancy output formats easy to design and simple to program.

\[
\text{(PRINTNUM FORMAT NUMBER FILE)}
\]

[Function]

Prints \text{NUMBER} on \text{FILE} according to the format \text{FORMAT}. \text{FORMAT} is a list structure with one of the forms described below.

If \text{FORMAT} is a list of the form \( \text{(FIX WIDTH RADIX PAD0 LEFTFLUSH)} \), this specifies a \text{FIX} format. \text{NUMBER} is rounded to the nearest integer, and then printed in a field \text{WIDTH} characters long with radix set to \text{RADIX} (or 10 if \text{RADIX} = \text{NIL}; note that the setting from the function \text{RADIX} is not used as the default). If \text{PAD0} and \text{LEFTFLUSH} are both \text{NIL}, the number is right-justified in the field, and the padding characters to the left of the leading digit are spaces. If \text{PAD0} is \text{T}, the character "0" is used for padding. If \text{LEFTFLUSH} is \text{T}, then the number is left-justified in the field, with trailing spaces to fill out \text{WIDTH} characters.

The following examples illustrate the effects of the \text{FIX} format options on the number 9 (the vertical bars indicate the field width):

\[
\begin{align*}
\text{FORMAT:} & \quad \text{(PRINTNUM FORMAT 9) prints:} \\
\text{(FIX 2)} & \quad |9| \\
\text{(FIX 2 NIL T)} & \quad |09| \\
\text{(FIX 12 8 T)} & \quad |000000000011| \\
\text{(FIX 5 NIL NIL T)} & \quad |9| \\
\end{align*}
\]

If \text{FORMAT} is a list of the form \( \text{(FLOAT WIDTH DECPART EXPPART PAD0 ROUND)} \), this specifies a \text{FLOAT} format. \text{NUMBER} is printed as a decimal number in a field \text{WIDTH} characters wide, with \text{DECPART} digits to the right of the decimal point. If \text{EXPPART} is not 0 (or \text{NIL}), the number is printed in exponent notation, with the exponent occupying \text{EXPPART} characters in the field. \text{EXPPART} should allow for the character \text{E} and an optional sign to be printed before the exponent digits. As with \text{FIX} format, padding on the left is with spaces, unless \text{PAD0} is \text{T}. If \text{ROUND} is given, it indicates the digit position at which rounding is to take place, counting from the leading digit of the number.

Interlisp-D interprets \text{WIDTH} = \text{NIL} to mean no padding, i.e., to use however much space the number needs, and interprets \text{DECPART} = \text{NIL} to mean as many decimal places as needed.

The following examples illustrate the effects of the \text{FLOAT} format options on the number 27.689 (the vertical bars indicate the field width):

\[
\begin{align*}
\text{FORMAT:} & \quad \text{(PRINTNUM FORMAT 27.689) prints:} \\
\text{(FLOAT 7 2)} & \quad |27.69|
\end{align*}
\]
### 25.3.3 User Defined Printing

Initially, Interlisp only knows how to print in an interesting way objects of type `litatom`, `number`, `string`, `list` and `stackp`. All other types of objects are printed in the form `{datatype}` followed by the octal representation of the address of the pointer, a format that cannot be read back in to produce an equivalent object.

When defining user data types (using the `DATATYPE` record type, page 8.9), it is often desirable to specify as well how objects of that type should be printed, so as to make their contents readable, or at least more informative to the viewer. The function `DEFPRINT` is used to specify the printing format of a data type.

#### (DEFPRINT TYPE FN)

`TYPE` is a type name. Whenever a printing function (`PRINT`, `PRIN1`, `PRIN2`, etc.) or a function requiring a print name (`CHCON`, `NCHARS`, etc.) encounters an object of the indicated type, `FN` is called with two arguments: the item to be printed and the name of the stream, if any, to which the object is to be printed. The second argument is `NIL` on calls that request the print name of an object without actually printing it.

If `FN` returns a list of the form `(ITEM1 . ITEM2)`, `ITEM1` is printed using `PRIN1` (unless it is `NIL`), and then `ITEM2` is printed using `PRIN2` (unless it is `NIL`). No spaces are printed between the two items. Typically, `ITEM1` is a read macro character.

If `FN` returns `NIL`, the datum is printed in the system default manner.
If \( FN \) returns \( T \), nothing further is printed; \( FN \) is assumed to have printed the object to the stream itself. Note that this case is permitted only when the second argument passed to \( FN \) is non-NIL; otherwise, there is no destination for \( FN \) to do its printing, so it must return as in one of the other two cases.

### 25.3.4 Printing Unusual Data Structures

\( \text{HPRINT} \) (for "Horrible Print") and \( \text{HREAD} \) provide a mechanism for printing and reading back in general data structures that cannot normally be dumped and loaded easily, such as (possibly re-entrant or circular) structures containing user datatypes, arrays, hash tables, as well as list structures. \( \text{HPRINT} \) will correctly print and read back in any structure containing any or all of the above, chasing all pointers down to the level of literal atoms, numbers or strings. \( \text{HPRINT} \) currently cannot handle compiled code arrays, stack positions, or arbitrary unboxed numbers.

\( \text{HPRINT} \) operates by simulating the Interlisp PRINT routine for normal list structures. When it encounters a user datatype (see page 8.20), or an array or hash array, it prints the data contained therein, surrounded by special characters defined as read macro characters (see page 25.39). While chasing the pointers of a structure, it also keeps a hash table of those items it encounters, and if any item is encountered a second time, another read macro character is inserted before the first occurrence (by resetting the file pointer with \( \text{SETFILEPTR} \)) and all subsequent occurrences are printed as a back reference using an appropriate macro character. Thus the inverse function, \( \text{HREAD} \) merely calls the Interlisp READ routine with the appropriate read table.

\[
\text{(HPRINT EXPR FILE UNCIRCULAR DATATYPESEEN)}
\]

\[\text{[Function]}\]

Prints \( EXPR \) on \( \text{FILE} \). If \( \text{UNCIRCULAR} \) is non-NIL, \( \text{HPRINT} \) does no checking for any circularities in \( EXPR \) (but is still useful for dumping arbitrary structures of arrays, hash arrays, lists, user data types, etc., that do not contain circularities). Specifying \( \text{UNCIRCULAR} \) as non-NIL results in a large, speed and internal-storage advantage.

Normally, when \( \text{HPRINT} \) encounters a user data type for the first time, it outputs a summary of the data type's declaration. When this is read in, the data type is redeclared. If \( \text{DATATYPESEEN} \) is non-NIL, \( \text{HPRINT} \) assumes that the same data type declarations will be in force at read time as were at \( \text{HPRINT} \) time, and not output declarations.

\( \text{HPRINT} \) is intended primarily for output to random access files, since the algorithm depends on being able to reset the file pointer. If \( \text{FILE} \) is not a random access file (and \( \text{UNCIRCULAR} = \text{NIL} \)), a temporary file, \( \text{HPRINT.SCRATCH} \), is opened, \( EXPR \) is
**OUTPUT FUNCTIONS**

HPRINTed on it, and then that file is copied to the final output file and the temporary file is deleted.

(HREAD FILE)  
**[Function]**  
Reads and returns an HPRINT-ed expression from FILE.

(HCOPYALL X)  
**[Function]**  
Copies data structure X. X may contain circular pointers as well as arbitrary structures.

Note: HORRIBLEVARS and UGLYVARS (page 17.36) are two file package commands for dumping and reloading circular and re-entrant data structures. They provide a convenient interface to HPRINT and HREAD.

When HPRINT is dumping a data structure that contains an instance of an Interlisp datatype, the datatype declaration is also printed onto the file. Reading such a data structure with HREAD can cause problems if it redefines a system datatype. Redefining a system datatype will almost definitely cause serious errors. The Interlisp system datatypes do not change very often, but there is always a possibility when loading in old files created under an old Interlisp release.

To prevent accidental system crashes, HREAD will not redefine datatypes. Instead, it will cause an error "attempt to read DATATYPE with different field specification than currently defined". Continuing from this error will redefine the datatype.

---

### 25.4 Random Access File Operations

For most applications, files are read starting at their beginning and proceeding sequentially, i.e., the next character read is the one immediately following the last character read. Similarly, files are written sequentially. However, for files on some devices, it is also possible to read/write characters at arbitrary positions in a file, essentially treating the file as a large block of auxiliary storage. For example, one application might involve writing an expression at the beginning of the file, and then reading an expression from a specified point in its middle. This particular example requires the file be open for both input and output. However, random file input or output can also be performed on files that have been opened for only input or only output.

Associated with each file is a "file pointer" that points to the location where the next character is to be read from or written to. The file position of a byte is the number of bytes that precede...
it in the file, i.e., 0 is the position of the beginning of the file. The file pointer to a file is automatically advanced after each input or output operation. This section describes functions which can be used to reposition the file pointer on those files that can be randomly accessed. A file used in this fashion is much like an array in that it has a certain number of addressable locations that characters can be put into or taken from. However, unlike arrays, files can be enlarged. For example, if the file pointer is positioned at the end of a file and anything is written, the file "grows." It is also possible to position the file pointer beyond the end of file and then to write. (If the program attempts to read beyond the end of file, an END OF FILE error occurs.) In this case, the file is enlarged, and a "hole" is created, which can later be written into. Note that this enlargement only takes place at the end of a file; it is not possible to make more room in the middle of a file. In other words, if expression A begins at position 1000, and expression B at 1100, and the program attempts to overwrite A with expression C, whose printed representation is 200 bytes long, part of B will be altered.

Warning: File positions are always in terms of bytes, not characters. The user should thus be very careful about computing the space needed for an expression. In particular, NS characters may take multiple bytes (see page 25.22). Also, the end-of-line character (see page 24.19) may be represented by a different number of characters in different implementations. Output functions may also introduce end-of-line's as a result of LINELENGTH considerations. Therefore NCHARS (page 2.9) does not specify how many bytes an expression takes to print, even ignoring line length considerations.

(GETFILEPTR FILE) [Function]
Returns the current position of the file pointer for FILE, i.e., the byte address at which the next input/output operation will commence.

(SETFILEPTR FILE ADR) [Function]
Sets the file pointer for FILE to the position ADR; returns ADR. The special value ADR = -1 is interpreted to mean the address of the end of file.

Note: If a file is opened for output only, the end of file is initially zero, even if an old file by the same name had existed (see OPENSTREAM, page 24.2). If a file is opened for both input and output, the initial file pointer is the beginning of the file, but (SETFILEPTR FILE -1) sets it to the end of the file. If the file had been opened in append mode by (OPENSTREAM FILE 'APPEND), the file pointer right after opening would be set to the end of the existing file, in which case a SETFILEPTR to position the file at the end would be unnecessary.
### GETEOFPTR FILE

Returns the byte address of the end of file, i.e., the number of bytes in the file. Equivalent to performing (SETFILEPTR FILE -1) and returning (GETFILEPTR FILE) except that it does not change the current file pointer.

### RANDACCESSP FILE

Returns FILE if FILE is randomly accessible, NIL otherwise. The file T is not randomly accessible, nor are certain network file connections in Interlisp-D. FILE must be open or an error is generated, FILE NOT OPEN.

### COPYBYTES SRCFIL DSTFIL START END

Copies bytes from SRCFIL to DSTFIL, starting from position START and up to but not including position END. Both SRCFIL and DSTFIL must be open. Returns T.

If END = NIL, START is interpreted as the number of bytes to copy (starting at the current position). If START is also NIL, bytes are copied until the end of the file is reached.

Warning: COPYBYTES does not take any account of multi-byte NS characters (page 2.12). COPYCHARS (below) should be used whenever copying information that might include NS characters.

### COPYCHARS SRCFIL DSTFIL START END

Like COPYBYTES except that it copies NS characters (page 2.12), and performs the proper conversion if the end-of-line conventions of SRCFIL and DSTFIL are not the same (see page 24.19). START and END are interpreted the same as with COPYBYTES, i.e., as byte (not character) specifications in SRCFIL. The number of bytes actually output to DSTFIL might be more or less than the number of bytes specified by START and END, depending on what the end-of-line conventions are. In the case where the end-of-line conventions happen to be the same, COPYCHARS simply calls COPYBYTES.

### FILEPOS PATTERN FILE START END SKIP TAIL CASEARRAY

Analogous to STRPOS (page 4.5), but searches a file rather than a string. FILEPOS searches FILE for the string PATTERN. Search begins at START (or the current position of the file pointer, if START = NIL), and goes to END (or the end of FILE, if END = NIL). Returns the address of the start of the match, or NIL if not found.

SKIP can be used to specify a character which matches any character in the file. If TAIL is T, and the search is successful, the value is the address of the first character after the sequence of characters corresponding to PATTERN, instead of the starting address of the sequence. In either case, the file is left so that the
next i/o operation begins at the address returned as the value of FILEPOS.

CASEARRAY should be a "case array" that specifies that certain characters should be transformed to other characters before matching. Case arrays are returned by CASEARRAY or SEPRCASE below. CASEARRAY = NIL means no transformation will be performed.

A case array is an implementation-dependent object that is logically an array of character codes with one entry for each possible character. FILEPOS maps each character in the file "through" CASEARRAY in the sense that each character code is transformed into the corresponding character code from CASEARRAY before matching. Thus if two characters map into the same value, they are treated as equivalent by FILEPOS. CASEARRAY and SETCASEARRAY provide an implementation-independent interface to case arrays.

For example, to search without regard to upper and lower case differences, CASEARRAY would be a case array where all characters map to themselves, except for lower case characters, whose corresponding elements would be the upper case characters. To search for a delimited atom, one could use "ATOM " as the pattern, and specify a case array in which all of the break and separator characters mapped into the same code as space.

For applications calling for extensive file searches, the function FFILEPOS is often faster than FILEPOS.

(FFILEPOS PATTERN FILE START END SKIP TAIL CASEARRAY) [Function]
Like FILEPOS, except much faster in most applications. FFILEPOS is an implementation of the Boyer-Moore fast string searching algorithm. This algorithm preprocesses the string being searched for and then scans through the file in steps usually equal to the length of the string. Thus, FFILEPOS speeds up roughly in proportion to the length of the string, e.g., a string of length 10 will be found twice as fast as a string of length 5 in the same position.

Because of certain fixed overheads, it is generally better to use FILEPOS for short searches or short strings.

(CASEARRAY OLDARRAY) [Function]
Creates and returns a new case array, with all elements set to themselves, to indicate the identity mapping. If OLDARRAY is given, it is reused.
**RANDOM ACCESS FILE OPERATIONS**

**FUNCTIONS**

**SETCASEARRAY CASEARRAY FROMCODE TOCODE**

Modifies the case array `CASEARRAY` so that character code `FROMCODE` is mapped to character code `TOCODE`.

**GETCASEARRAY CASEARRAY FROMCODE**

Returns the character code that `FROMCODE` is mapped to in `CASEARRAY`.

**SEPRCASE CLFLG**

Returns a new case array suitable for use by FILEPOS or FFILEPOS in which all of the break/separators of FILERDTBL are mapped into character code zero. If `CLFLG` is non-NIL, then all CLISP characters are mapped into this character as well. This is useful for finding a delimited atom in a file. For example, if PATTERN is "FOO", and (SEPRCASE T) is used for CASEARRAY, then FILEPOS will find "(FOO←".

**UPPERCASEARRAY**

Value is a case array in which every lowercase character is mapped into the corresponding uppercase character. Useful for searching text files.

**25.5 Input/Output Operations with Characters and Bytes**

Interlisp-D supports the 16-bit NS character set (see page 2.12). All of the standard string and print name functions accept litatoms and strings containing NS characters. In almost all cases, a program does not have to distinguish between NS characters or 8-bit characters. The exception to this rule is the handling of input/output operations. Interlisp-D uses two ways of writing 16-bit NS characters on files. One way is to write the full 16-bits (two bytes) every time a character is output. The other way is to use "run-encoding." Each 16 NS character can be decoded into a character set (an integer from 0 to 254 inclusive) and a character number (also an integer from 0 to 254 inclusive). In run-encoding, the byte 255 (illegal as either a character set number or a character number) is used to signal a change to a given character set, and the following bytes are all assumed to come from the same character set (until the next change-character set sequence). Run-encoding can reduce the number of bytes required to encode a string of NS characters, as long as there are long sequences of characters from the same character set (usually the case).
Note that characters are not the same as bytes. A single character can take anywhere from one to four bytes, depending on whether it is in the same character set as the preceding character, and whether run-encoding is enabled. Programs which assume that characters are equal to bytes must be changed to work with NS characters.

The functions BIN (page 25.5) and BOUT (page 25.9) should only be used to read and write single eight-bit bytes. The functions READCCODE (page 25.5) and PRINTCCODE (page 25.9) should be used to read and write single character codes, interpreting run-encoded NS characters. COPYBYTES (page 25.20) should only be used to copy blocks of 8-bit data; COPYCHARS should be used to copy characters. Most I/O functions (READC, PRIN1, etc.) read or write 16-bit NS characters.

The use of NS characters has serious consequences for any program that uses file pointers to access a file in a random access manner. At any point when a file is being read or written, it has a "current character set." If the file pointer is changed with SETFILEPTR (page 25.19) to a part of the file with a different character set, any characters read or written may have the wrong character set. The current character set can be accessed with the following function:

(CHARSET STREAM CHARACTERSET)  [Function]

Returns the current character set of the stream STREAM. If CHARACTERSET is non-NIL, the current character set for STREAM is set. Note that for output streams this may cause bytes to be written to the stream.

If CHARACTERSET is T, run encoding for STREAM is disabled: both the character set and the character number (two bytes total) will be written to the stream for each character printed.

25.6 PRINTOUT

Interlisp provides many facilities for controlling the format of printed output. By executing various sequences of PRIN1, PRIN2, TAB, TERPRI, SPACES, PRINTNUM, and PRINTDEF, almost any effect can be achieved. PRINTOUT implements a compact language for specifying complicated sequences of these elementary printing functions. It makes fancy output formats easy to design and simple to program.

PRINTOUT is a CLISP word (like FOR and IF) for interpreting a special printing language in which the user can describe the kinds of printing desired. The description is translated by DWIMIFY to the appropriate sequence of PRIN1, TAB, etc.,
before it is evaluated or compiled. PRINTOUT printing descriptions have the following general form:

\[ \text{PRINTOUT STREAM PRINTCOM}_1 \ldots \text{PRINTCOM}_N \]

STREAM is evaluated to obtain the stream to which the output from this specification is directed. The PRINTOUT commands are strung together, one after the other without punctuation, after STREAM. Some commands occupy a single position in this list, but many commands expect to find arguments following the command name in the list. The commands fall into several logical groups: one set deals with horizontal and vertical spacing, another group provides controls for certain formatting capabilities (font changes and subscripting), while a third set is concerned with various ways of actually printing items. Finally, there is a command that permits escaping to a simple Lisp evaluation in the middle of a PRINTOUT form. The various commands are described below. The following examples give a general flavor of how PRINTOUT is used:

Example 1: Suppose the user wanted to print out on the terminal the values of three variables, X, Y, and Z, separated by spaces and followed by a carriage return. This could be done by:

\[
(\text{PRIN1} \ X \ T) \\
(\text{SPACES} \ 1 \ T) \\
(\text{PRIN1} \ Y \ T) \\
(\text{SPACES} \ 1 \ T) \\
(\text{PRIN1} \ Z \ T) \\
(\text{TERPRI} \ T)
\]

or by the more concise PRINTOUT form:

\[
(\text{PRINTOUT} \ \text{NIL} \ \text{"X = "} \ 10 \ \text{.PPV} \ X \ T)
\]

Example 2: Suppose the values of X and Y are to be pretty-printed lined up at position 10, preceded by identifying strings. If the output is to go to the primary output stream, the user could write either:

\[
(\text{PRIN1} \ \text{"X = "}) \\
(\text{PRINTDEF} \ X \ 10 \ T) \\
(\text{TERPRI}) \\
(\text{PRIN1} \ \text{"Y = "}) \\
(\text{PRINTDEF} \ Y \ 10 \ T) \\
(\text{TERPRI})
\]

or the equivalent:

\[
(\text{PRINTOUT} \ \text{NIL} \ \text{"X = "} \ 10 \ \text{.PPV} \ X \ T)
\]
"Y = " 10 .PPV Y T)

Since strings are not recognized as special commands, "X =" is also printed with PRIN1 by default. The positive integer means TAB to position 10, where the .PPV command causes the value of X to be prettyprinted as a variable. By convention, special atoms used as PRINTOUT commands are prefixed with a period. The T causes a carriage return, so the Y information is printed on the next line.

Example 3. As a final example, suppose that the value of X is an integer and the value of Y is a floating-point number. X is to be printed right-flushed in a field of width 5 beginning at position 15, and Y is to be printed in a field of width 10 also starting at position 15 with 2 places to the right of the decimal point. Furthermore, suppose that the variable names are to appear in the font class named BOLDFONT and the values in font class SMALLFONT. The program in ordinary Interlisp that would accomplish these effects is too complicated to include here. With PRINTOUT, one could write:

(PRINTOUT NIL
.FONT BOLDFONT "X •" 15
.FONT SMALLFONT .IS X T
.FONT BOLDFONT "Y •" 15
.FONT SMALLFONT .F10.2 Y T
.FONT BOLDFONT)

The .FONT commands do whatever is necessary to change the font on a multi-font output device. The .IS command sets up a FIX format for a call to the function PRINTNUM (page 25.15) to print X in the desired format. The .F10.2 specifies a FLOAT format for PRINTNUM.

25.6.1 Horizontal Spacing Commands

The horizontal spacing commands provide convenient ways of calling TAB and SPACES. In the following descriptions, N stands for a literal positive integer (not for a variable or expression whose value is an integer).

N(N a number) [PRINTOUT command] Used for absolute spacing. It results in a TAB to position N (literally, a (TAB N)). If the line is currently at position N or beyond, the file will be positioned at position N on the next line.

.TAB POS [PRINTOUT command] Specifies TAB to position (the value of) POS. This is one of several commands whose effect could be achieved by simply escaping to Lisp, and executing the corresponding form. It is provided as a
separate command so that the PRINTOUT form is more concise and is prettyprinted more compactly. Note that .TAB N and \( N' \), where \( N \) is an integer, are equivalent.

\textbf{.TAB0 POS} \hfill [PRINTOUT command]

Like .TAB except that it can result in zero spaces (i.e. the call to TAB specifies MINSPACES = 0).

\textbf{-N (N a number)} \hfill [PRINTOUT command]

Negative integers indicate relative (as opposed to absolute) spacing. Translates as (SPACES \( |N| \)).

\textbf{,} \hfill [PRINTOUT command]

\textbf{"} \hfill [PRINTOUT command]

\textbf{""} \hfill [PRINTOUT command]

\textbf{"""} \hfill [PRINTOUT command]

(1, 2 or 3 commas) Provides a short-hand way of specifying 1, 2 or 3 spaces, i.e., these commands are equivalent to -1, -2, and -3, respectively.

\textbf{.SP DISTANCE} \hfill [PRINTOUT command]

Translates as (SPACES DISTANCE). Note that .SP \( N \) and -\( N \), where \( N \) is an integer, are equivalent.

\textbf{25.6.2 Vertical Spacing Commands}

Vertical spacing is obtained by calling TERPRI or printing form-feeds. The relevant commands are:

\textbf{T} \hfill [PRINTOUT command]

Translates as (TERPRI), i.e., move to position 0 (the first column) of the next line. To print the letter T, use the string "T".

\textbf{.SKIP LINES} \hfill [PRINTOUT command]

Equivalent to a sequence of LINES (TERPRI)'s. The .SKIP command allows for skipping large constant distances and for computing the distance to be skipped.

\textbf{.PAGE} \hfill [PRINTOUT command]

Puts a form-feed (control-L) out on the file. Care is taken to make sure that Interlisp's view of the current line position is correctly updated.
25.6.3 Special Formatting Controls

There are a small number of commands for invoking some of the formatting capabilities of multi-font output devices. The available commands are:

\texttt{.FONT FONTSPEC} \hspace{1cm} [PRINTOUT command]

Changes printing to the font \texttt{FONTSPEC}, which can be a font descriptor, a "font list" such as '\texttt{(MODERN 10)}, an image stream (coerced to its current font), or a windows (coerced to the current font of its display stream). See fonts (page 27.25) for more information.

\texttt{FONTSPEC} may also be a positive integer \texttt{N}, which is taken as an abbreviated reference to the font class named \texttt{FONTN} (e.g. \texttt{1 = > FONT1}).

\texttt{.SUP} \hspace{1cm} [PRINTOUT command]

Specifies superscripting. All subsequent characters are printed above the base of the current line. Note that this is absolute, not relative: a \texttt{.SUP} following a \texttt{.SUP} is a no-op.

\texttt{.SUB} \hspace{1cm} [PRINTOUT command]

Specifies subscripting. Subsequent printing is below the base of the current line. As with superscripting, the effect is absolute.

\texttt{.BASE} \hspace{1cm} [PRINTOUT command]

Moves printing back to the base of the current line. Un-does a previous \texttt{.SUP} or \texttt{.SUB}; a no-op, if printing is currently at the base.

25.6.4 Printing Specifications

The value of any expression in a PRINTOUT form that is not recognized as a command itself or as a command argument is printed using PRIN1 by default. For example, title strings can be printed by simply including the string as a separate PRINTOUT command, and the values of variables and forms can be printed in much the same way. Note that a literal integer, say \texttt{51}, cannot be printed by including it as a command, since it would be interpreted as a TAB; the desired effect can be obtained by using instead the string specification "\texttt{51}" or the form \texttt{(QUOTE 51)}.

For those instances when PRIN1 is not appropriate, e.g., PRIN2 is required, or a list structures must be prettyprinted, the following commands are available:
.P2 THING  [PRINTOUT command]  
Causes THING to be printed using PRIN2; translates as (PRIN2 THING).

.PPF THING  [PRINTOUT command]  
Causes THING to be prettyprinted at the current line position via PRINTDEF (page 26.42). The call to PRINTDEF specifies that THING is to be printed as if it were part of a function definition. That is, SELECTQ, PROG, etc., receive special treatment.

.PPV THING  [PRINTOUT command]  
Prettyprints THING as a variable; no special interpretation is given to SELECTQ, PROG, etc.

.PPFTL THING  [PRINTOUT command]  
Like .PPF, but prettyprints THING as a tail, that is, without the initial and final parentheses if it is a list. Useful for prettyprinting sub-lists of a list whose other elements are formatted with other commands.

.PPVTL THING  [PRINTOUT command]  
Like .PPV, but prettyprints THING as a tail.

25.6.4.1 Paragraph Format

Interlisp's prettyprint routines are designed to display the structure of expressions, but they are not really suitable for formatting unstructured text. If a list is to be printed as a textual paragraph, its internal structure is less important than controlling its left and right margins, and the indentation of its first line. The .PARA and .PARA2 commands allow these parameters to be conveniently specified.

.PARA LMARG RMARG LIST  [PRINTOUT command]  
Prints LIST in paragraph format, using PRIN1. Translates as (PRINTPARA LMARG RMARG LIST) (see page 25.32).

Example: (PRINTOUT T 10 .PARA 5 -5 LST) will print the elements of LST as a paragraph with left margin at 5, right margin at (LINELENGTH)-5, and the first line indented to 10.

.PARA2 LMARG RMARG LIST  [PRINTOUT command]  
Print as paragraph using PRIN2 instead of PRIN1. Translates as (PRINTPARA LMARG RMARG LIST T).
25.6.4.2 Right-Flushing

Two commands are provided for printing simple expressions flushed-right against a specified line position, using the function `FLUSHRIGHT` (page 25.32). They take into account the current position, the number of characters in the print-name of the expression, and the position the expression is to be flush against, and then print the appropriate number of spaces to achieve the desired effect. Note that this might entail going to a new line before printing. Note also that right-flushing of expressions longer than a line (e.g. a large list) makes little sense, and the appearance of the output is not guaranteed.

`.FR POS EXPR` [PRINTOUT command]

Flush-right using PRIN1. The value of `POS` determines the position that the right end of `EXPR` will line up at. As with the horizontal spacing commands, a negative position number means `[POS]` columns from the current position, a positive number specifies the position absolutely. `POS = 0` specifies the right-margin, i.e. is interpreted as `(LINELENGTH)`.

`.FR2 POS EXPR` [PRINTOUT command]

Flush-right using PRIN2 instead of PRIN1.

25.6.4.3 Centering

Commands for centering simple expressions between the current line position and another specified position are also available. As with right flushing, centering of large expressions is not guaranteed.

`.CENTER POS EXPR` [PRINTOUT command]

Centers `EXPR` between the current line position and the position specified by the value of `POS`. A positive `POS` is an absolute position number, a negative `POS` specifies a position relative to the current position, and 0 indicates the right-margin. Uses PRIN1 for printing.

`.CENTER2 POS EXPR` [PRINTOUT command]

Centers using PRIN2 instead of PRIN1.

25.6.4 Numbering

The following commands provide FORTRAN-like formatting capabilities for integer and floating-point numbers. Each command specifies a printing format and a number to be
printed. The format specification translates into a format-list for
the function PRINTNUM (see page 25.15).

`.FORMAT NUMBER' [PRINTOUT command]
Specifies integer printing. Translates as a call to the function
PRINTNUM with a FIX format-list constructed from FORMAT. The
atomic format is broken apart at internal periods to form the
format-list. For example, .I5.8.T yields the format-list (FIX 5 8 T),
and the command sequence (PRINTOUT T .I5.8.T FOO) translates
as (PRINTNUM '(FIX 5 8 T) FOO). This expression causes the value
of FOO to be printed in radix 8 right-flushed in a field of width 5,
with 0's used for padding on the left. Internal NIL's in the format
specification may be omitted, e.g., the commands .I5..T and
.I5.NIL.T are equivalent.

The format specification .I is often useful for forcing a number
to be printed in radix 10 (but not otherwise specially formatted),
independent of the current setting of RADIX.

`.FORMAT NUMBER' [PRINTOUT command]
Specifies floating-number printing. Like the .I format command,
except translates with a FLOAT format-list.

`.N FORMAT NUMBER' [PRINTOUT command]
The .I and .F commands specify calls to PRINTNUM with quoted
format specifications. The .N command translates as (PRINTNUM
FORMAT NUMBER), i.e., it permits the format to be the value of
some expression. Note that, unlike the .I and .F commands,
FORMAT is a separate element in the command list, not part of
an atom beginning with .N.

25.6.5 Escaping to Lisp
There are many reasons for taking control away from PRINTOUT
in the middle of a long printing expression. Common situations
involve temporary changes to system printing parameters (e.g.
LINELENGTH), conditional printing (e.g. print FOO only if FIE is T),
or lower-level iterative printing within a higher-level print
specification.

`# FORM' [PRINTOUT command]
The escape command. FORM is an arbitrary Lisp expression that
is evaluated within the context established by the PRINTOUT
form, i.e., FORM can assume that the primary output stream has
been set to be the FILE argument to PRINTOUT. Note that
nothing is done with the value of FORM; any printing desired is
accomplished by FORM itself, and the value is discarded.
25.6.6 User-Defined Commands

The collection of commands and options outlined above is aimed at fulfilling all common printing needs. However, certain applications might have other, more specialized printing idioms, so a facility is provided whereby the user can define new commands. This is done by adding entries to the global list PRINTOUTMACROS to define how the new commands are to be translated.

PRINTOUTMACROS

PRINTOUTMACROS is an association-list whose elements are of the form (COMM FN). Whenever COMM appears in command position in the sequence of PRINTOUT commands (as opposed to an argument position of another command), FN is applied to the tail of the command-list (including the command).

After inspecting as much of the tail as necessary, the function must return a list whose CAR is the translation of the user-defined command and its arguments, and whose CDR is the list of commands still remaining to be translated in the normal way.

For example, suppose the user wanted to define a command "?", which will cause its single argument to be printed with PRIN1 only if it is not NIL. This can be done by entering (? ?TRAN) on PRINTOUTMACROS, and defining the function ?TRAN as follows:

(DEFINEQ (?TRAN (COMS))
 (CONS
  (SUBST (CADR COMS) 'ARG
    '(PROG «TEMP ARG»
      (COND (TEMP (PRIN1 TEMP)))
    (CDDR COMS))]
Note that ?TRAN does not do any printing itself; it returns a form which, when evaluated in the proper context, will perform the
desired action. This form should direct all printing to the primary output file.

25.6.7 Special Printing Functions

The paragraph printing commands are translated into calls on the function PRINTPARA, which may also be called directly:

(PRINTPARA LMARG RMARG LIST P2FLAG PARENFLAG FILE)  [Function]

Prints LIST on FILE in line-filled paragraph format with its first element beginning at the current line position and ending at or before RMARG, and with subsequent lines appearing between LMARG and RMARG. If P2FLAG is non-NIL, prints elements using PRIN2, otherwise PRIN1. If PARENFLAG is non-NIL, then parentheses will be printed around the elements of LIST.

If LMARG is zero or positive, it is interpreted as an absolute column position. If it is negative, then the left margin will be at \(|LMARG| + (POSITION)\). If LMARG = NIL, the left margin will be at (POSITION), and the paragraph will appear in block format.

If RMARG is positive, it also is an absolute column position (which may be greater than the current (LINELENGTH)). Otherwise, it is interpreted as relative to (LINELENGTH), i.e., the right margin will be at (LINELENGTH) + \(|RMARG|\). Example: (TAB 10) (PRINTPARA 5 -5 LST T) will PRIN2 the elements of LST in a paragraph with the first line beginning at column 10, subsequent lines beginning at column 5, and all lines ending at or before (LINELENGTH)-5.

The current (LINELENGTH) is unaffected by PRINTPARA, and upon completion, FILE will be positioned immediately after the last character of the last item of LIST. PRINTPARA is a no-op if LIST is not a list.

The right-flushing and centering commands translate as calls to the function FLUSHRIGHT:

(FLUSHRIGHT POS X MIN P2FLAG CENTERFLAG FILE)  [Function]

If CENTERFLAG = NIL, prints X right-flushed against position POS on FILE; otherwise, centers X between the current line position and POS. Makes sure that it spaces over at least MIN spaces before printing by doing a TERPRI if necessary; MIN = NIL is equivalent to MIN = 1. A positive POS indicates an absolute position, while a negative POS signifies the position which is \(|POS|\) to the right of the current line position. POS = 0 is interpreted as (LINELENGTH), the right margin.
25.7 READFILE and WRITEFILE

For those applications where the user simply wants to simply read all of the expressions on a file, and not evaluate them, the function READFILE is available:

(READFILE FILE RDTBL ENDTOKEN) [NoSpread Function]

Reads successive expressions from file using READ (with read table RDTBL) until the single litatom ENDTOKEN is read, or an end of file encountered. Returns a list of these expressions.

If RDTBL is not specified, it defaults to FILERDTBL. If ENDTOKEN is not specified, it defaults to the litatom STOP.

(WRITEFILE X FILE) [Function]

Writes a date expression onto FILE, followed by successive expressions from X, using FILERDTBL as a read table. If X is atomic, its value is used. If FILE is not open, it is opened. If FILE is a list, (CAR FILE) is used and the file is left opened. Otherwise, when X is finished, the litatom STOP is printed on FILE and it is closed. Returns FILE.

(ENDFILE FILE) [Function]

Prints STOP on FILE and closes it.

25.8 Read Tables

Many Interlisp input functions treat certain characters in special ways. For example, READ recognizes that the right and left parenthesis characters are used to specify list structures, and that the quote character is used to delimit text strings. The Interlisp input and (to a certain extent) output routines are table driven by read tables. Read tables are objects that specify the syntactic properties of characters for input routines. Since the input routines parse character sequences into objects, the read table in use determines which sequences are recognized as literal atoms, strings, list structures, etc.

Most Interlisp input functions take an optional read table argument, which specifies the read table to use when reading an expression. If NIL is given as the read table, the "primary read table" is used. If T is specified, the system terminal read table is used. Some functions will also accept the atom ORIG (not the value of ORIG) as indicating the "original" system read table. Some output functions also take a read table argument. For
example, PRIN2 prints an expression so that it would be read in
correctly using a given read table.

The Interlisp-D system uses the following read tables: T for
input/output from terminals, the value of FILERDTBL for
input/output from files, the value of EDITRDTBL for input from
terminals while in the tty-based editor, the value of DEDITRDTBL
for input from terminals while in the display-based editor, and
the value of CODERDTBL for input/output from compiled files.
These five read tables are initially copies of the ORIG read table,
with changes made to some of them to provide read macros
(page 25.39) that are specific to terminal input or file input.
Using the functions described below, the user may further
change, reset, or copy these tables. However, in the case of
FILERDTBL and CODERDTBL, the user is cautioned that changing
these tables may prevent the system from being able to read files
made with the original tables, or prevent users possessing only
the standard tables from reading files made using the modified
tables.

The user can also create new read tables, and either explicitly
pass them to input/output functions as arguments, or install
them as the primary read table, via SETREADTABLE, and then not
specify a RDTBL argument, i.e., use NIL.

25.8.1 Read Table Functions

(READTABLEP RDTBL) [Function]
Returns RDTBL if RDTBL is a real read table (not T or ORIG),
otherwise NIL.

(GETREADTABLE RDTBL) [Function]
If RDTBL = NIL, returns the primary read table. If RDTBL = T,
returns the system terminal read table. If RDTBL is a real read
table, returns RDTBL. Otherwise, generates an ILLEGAL
READTABLE error.

(SETREADTABLE RDTBL FLG) [Function]
Sets the primary read table to RDTBL. If FLG = T, SETREADTABLE
sets the system terminal read table, T. Note that the user can
reset the other system read tables with SETQ, e.g., (SETQ
FILERDTBL (GETREADTABLE)).
Generates an ILLEGAL READTABLE error if RDTBL is not NIL, T, or
a real read table. Returns the previous setting of the primary
read table, so SETREADTABLE is suitable for use with
RESETFORM (page 14.26).
(COPYREADTABLE RDTBL)

Returns a copy of RDTBL. RDTBL can be a real read table, NIL, T, or ORIG (in which case COPYREADTABLE returns a copy of the original system read table), otherwise COPYREADTABLE generates an ILLEGAL READTABLE error.

Note that COPYREADTABLE is the only function that creates a read table.

(RESETREADTABLE RDTBL FROM)

Copies (smashes) FROM into RDTBL. FROM and RDTBL can be NIL, T, or a real read table. In addition, FROM can be ORIG, meaning use the system's original read table.

25.8.2 Syntax Classes

A read table is an object that contains information about the "syntax class" of each character. There are nine basic syntax classes: LEFTPAREN, RIGHTPAREN, LEFTBRACKET, RIGHTBRACKET, STRINGDElIM, ESCAPE, BREAKCHAR, SEPRCHAR, and OTHER, each associated with a primitive syntactic property. In addition, there is an unlimited assortment of user-defined syntax classes, known as "read macros". The basic syntax classes are interpreted as follows:

- LEFTPAREN (normally left parenthesis) Begins list structure.
- RIGHTPAREN (normally right parenthesis) Ends list structure.
- LEFTBRACKET (normally left bracket) Begins list structure. Also matches RIGHTBRACKET characters.
- RIGHTBRACKET (normally left bracket) Ends list structure. Can close an arbitrary numbers of LEFTPAREN lists, back to the last LEFTBRACKET.
- STRINGDElIM (normally double quote) Begins and ends text strings. Within the string, all characters except for the one(s) with class ESCAPE are treated as ordinary, i.e., interpreted as if they were of syntax class OTHER. To include the string delimiter inside a string, prefix it with the ESCAPE character.
- ESCAPE (normally percent sign) Inhibits any special interpretation of the next character, i.e., the next character is interpreted to be of class OTHER, independent of its normal syntax class.
- BREAKCHAR (None initially) Is a break character, i.e., delimits atoms, but is otherwise an ordinary character.
- SEPRCHAR (space, carriage return, etc.) Delimits atoms, and is otherwise ignored.
- OTHER Characters that are not otherwise special belong to the class OTHER.
Characters of syntax class LEFTPAREN, RIGHTPAREN, LEFTBRACKET, RIGHTBRACKET, and STRINGDELIM are all break characters. That is, in addition to their interpretation as delimiting list or string structures, they also terminate the reading of an atom. Characters of class BREAKCHAR serve only to terminate atoms, with no other special meaning. In addition, if a break character is the first non-separator encountered by RATOM, it is read as a one-character atom. In order for a break character to be included in an atom, it must be preceded by the ESCAPE character.

Characters of class SEPRCHAR also terminate atoms, but are otherwise completely ignored; they can be thought of as logically spaces. As with break characters, they must be preceded by the ESCAPE character in order to appear in an atom.

For example, if $ were a break character and * a separator character, the input stream ABC**DEF$GH*$$ would be read by 6 calls to RATOM returning respectively ABC, DEF, $, GH, $, $.

Although normally there is only one character in a read table having each of the list- and string-delimiting syntax classes (such as LEFTPAREN), it is perfectly acceptable for any character to have any syntax class, and for more than one to have the same class.

Note that a "syntax class" is an abstraction: there is no object referencing a collection of characters called a syntax class. Instead, a read table provides the association between a character and its syntax class, and the input/output routines enforce the abstraction by using read tables to drive the parsing.

The functions below are used to obtain and set the syntax class of a character in a read table. CH can either be a character code (a integer), or a character (a single-character atom). Single-digit integers are interpreted as character codes, rather than as characters. For example, 1 indicates control-A, and 49 indicates the character 1. Note that CH can be a full sixteen-bit NS character (see page 2.12).

Note: Terminal tables, described on page 30.4, also associate characters with syntax classes, and they can also be manipulated with the functions below. The set of read table and terminal table syntax classes are disjoint, so there is never any ambiguity about which type of table is being referred to.

<table>
<thead>
<tr>
<th>GETSYNTAX CH TABLE</th>
<th>[Function]</th>
</tr>
</thead>
</table>
| Returns the syntax class of CH, a character or a character code, with respect to TABLE. TABLE can be NIL, T, ORIG, or a real read table or terminal table. 

CH can also be a syntax class, in which case GETSYNTAX returns a list of the character codes in TABLE that have that syntax class. |

25 36 INPUT/OUTPUT FUNCTIONS
(SETSYNTAX CHAR CLASS TABLE)  [Function]
Sets the syntax class of CHAR, a character or character code, in TABLE. TABLE can be either NIL, T, or a real read table or terminal table. SETSYNTAX returns the previous syntax class of CHAR. CLASS can be any one of the following:

- The name of one of the basic syntax classes.
- A list, which is interpreted as a read macro (see page 25.39).
- NIL, T, ORIG, or a real read table or terminal table, which means to give CHAR the syntax class it has in the table indicated by CLASS. For example, (SETSYNTAX '%' 'ORIG TABLE) gives the left parenthesis character in TABLE the same syntax class that it has in the original system read table.
- A character code or character, which means to give CHAR the same syntax class as the character CHAR in TABLE. For example, (SETSYNTAX ' ' 'TABLE) gives the left brace character the same syntax class as the left bracket.

(SYNTAXP CODE CLASS TABLE)  [Function]
CODE is a character code; TABLE is NIL, T, or a real read table or terminal table. Returns T if CODE has the syntax class CLASS in TABLE; NIL otherwise.

CLASS can also be a read macro type (MACRO, SPLICE, INFIX), or a read macro option (FIRST, IMMEDIATE, etc.), in which case SYNTAXP returns T if the syntax class is a read macro with the specified property.

Note: SYNTAXP will not accept a character as an argument, only a character code.

For convenience in use with SYNTAXP, the atom BREAK may be used to refer to all break characters, i.e., it is the union of LEFTPAREN, RIGHTPAREN, LEFTBRACKET, RIGHTBRACKET, STRINGDELIM, and BREAKCHAR. For purely symmetrical reasons, the atom SEPR corresponds to all separator characters. However, since the only separator characters are those that also appear in SEPRCHAR, SEPR and SEPRCHAR are equivalent.

Note that GETSYNTAX never returns BREAK or SEPR as a value although SETSYNTAX and SYNTAXP accept them as arguments. Instead, GETSYNTAX returns one of the disjoint basic syntax classes that comprise BREAK. BREAK as an argument to SETSYNTAX is interpreted to mean BREAKCHAR if the character is not already one of the BREAK classes. Thus, if '%' is of class LEFTPAREN, then (SETSYNTAX '%' 'BREAK) doesn’t do anything, since '%' is already a break character, but (SETSYNTAX '%' 'BREAKCHAR) means make '%' be just a break character, and therefore disables the LEFTPAREN function of '%'. Similarly, if one of the format characters is disabled completely, e.g., by...
(SETSNTAX '%(' 'OTHER), then (SETSNTAX '%(' 'BREAK) would make %(' be only a break character; it would not restore %(' as LEFTPAREN.

The following functions provide a way of collectively accessing and setting the separator and break characters in a read table:

(SETSEPR LST FLG RDTBL) [Function]
Sets or removes the separator characters for RDTBL. LST is a list of characters or character codes. FLG determines the action of SETSEPR as follows: if FLG = NIL, makes RDTBL have exactly the elements of LST as separators, discarding from RDTBL any old separator characters not in LST. If FLG = 0, removes from RDTBL as separator characters all elements of LST. This provides an "UNSETSEPR". If FLG = 1, makes each of the characters in LST be a separator in RDTBL.

If LST = T, the separator characters are reset to be those in the system's read table for terminals, regardless of the value of FLG, i.e., (SETSEPR T) is equivalent to (SETSEPR (GETSEPR T)). If RDTBL is T, then the characters are reset to those in the original system table.

Returns NIL.

(SETBRK LST FLG RDTBL) [Function]
Sets the break characters for RDTBL. Similar to SETSEPR.

As with SETSNTAX to the BREAK class, if any of the list- or string-delimiting break characters are disabled by an appropriate SETBRK (or by making it be a separator character), its special action for READ will not be restored by simply making it be a break character again with SETBRK. However, making these characters be break characters when they already are will have no effect.

The action of the ESCAPE character (normally %) is not affected by SETSEPR or SETBRK. It can be disabled by setting its syntax to the class OTHER, and other characters can be used for escape on input by assigning them the class ESCAPE. As of this writing, however, there is no way to change the output escape character; it is "hardwired" as %. That is, on output, characters of special
syntax that need to be preceded by the ESCAPE character will always be preceded by %, independent of the syntax of % or which, if any characters, have syntax ESCAPE.

The following function can be used for defeating the action of the ESCAPE character or characters:

(ESCAPE FLG RDTBL) [Function]

If FLG = NIL, makes characters of class ESCAPE behave like characters of class OTHER on input. Normal setting is (ESCAPE T).

ESCAPE returns the previous setting.

25.8.3 Read Macros

Read macros are user-defined syntax classes that can cause complex operations when certain characters are read. Read macro characters are defined by specifying as a syntax class an expression of the form:

(TYPE OPTION1 ... OPTIONN FN)

where TYPE is one of MACRO, SPLICE, or INFIX, and FN is the name of a function or a lambda expression. Whenever READ encounters a read macro character, it calls the associated function, giving it as arguments the input stream and read table being used for that call to READ. The interpretation of the value returned depends on the type of read macro:

MACRO This is the simplest type of read macro. The result returned from the macro is treated as the expression to be read, instead of the read macro character. Often the macro reads more input itself. For example, in order to cause "EXPR" to be read as (NOT EXPR), one could define " as the read macro:

[MACRO (LAMBDA (FL RDTBL) 
 (LIST 'NOT (READ FL RDTBL) ]

SPLICE The result (which should be a list or NIL) is spliced into the input using NCONC. For example, if $ is defined by the read macro:

(SPLICE (LAMBDA NIL (APPEND FOO)))

and the value of FOO is (A B C), then when the user inputs (X $ Y), the result will be (X A B C Y).

INFIX The associated function is called with a third argument, which is a list, in TCONC format (page 3.6), of what has been read at the current level of list nesting. The function's value is taken as a new TCONC list which replaces the old one. For example, the infix operator + could be defined by the read macro:

(INFIX (LAMBDA (FL RDTBL Z) 
 (RPLACA (CDR Z) 
 (LIST (QUOTE IPLUS) )

INPUT/OUTPUT FUNCTIONS
If an Infix read macro character is encountered not in a list, the third argument to its associated function is nil. If the function returns nil, the read macro character is essentially ignored and reading continues. Otherwise, if the function returns a TCONC list of one element, that element is the value of the READ. If it returns a TCONC list of more than one element, the list is the value of the READ.

The specification for a read macro character can be augmented to specify various options option1 ... optionn, e.g., (MACRO FIRST IMMEDIATE FN). The following three disjoint options specify when the read macro character is to be effective:

- **ALWAYS** The default. The read macro character is always effective (except when preceded by the % character), and is a break character, i.e., a member of (GETSYNTAX 'BREAK RDTBL).

- **FIRST** The character is interpreted as a read macro character only when it is the first character seen after a break or separator character; in all other situations, the character is treated as having class OTHER. The read macro character is not a break character. For example, the quote character is a FIRST read macro character, so that DON'T is read as the single atom DON'T, rather than as DON followed by (QUOTE T).

- **ALONE** The read macro character is not a break character, and is interpreted as a read macro character only when the character would have been read as a separate atom if it were not a read macro character, i.e., when its immediate neighbors are both break or separator characters.

Making a FIRST or ALONE read macro character be a break character (with SETBRK) disables the read macro interpretation, i.e., converts it to syntax class BREAKCHAR. Making an ALWAYS read macro character be a break character is a no-op.

The following two disjoint options control whether the read macro character is to be protected by the ESCAPE character on output when a litatom containing the character is printed:

- **ESQUOTE** or ESC The default. When printed with PRIN2, the read macro character will be preceded by the output escape character (%) as needed to permit the atom containing it to be read correctly. Note that for FIRST macros, this means that the character need be quoted only when it is the first character of the atom.

- **NOESQUOTE** or NOESC The read macro character will always be printed without an escape. For example, the ? read macro in the T read table is a NOESQUOTE character. Unless you are very careful what you are doing, read macro characters in FILERDTBL should never be
NOESCQUOTE, since symbols that happen to contain the read macro character will not read back in correctly.

The following two disjoint options control when the macro's function is actually executed:

**IMMEDIATE or IMMED**

The read macro character is immediately activated, i.e., the current line is terminated, as if an EOL had been typed, a carriage-return line-feed is printed, and the entire line (including the macro character) is passed to the input function.

**IMMEDIATE** read macro characters enable the user to specify a character that will take effect immediately, as soon as it is encountered in the input, rather than waiting for the line to be terminated. Note that this is not necessarily as soon as the character is typed. Characters that cause action as soon as they are typed are interrupt characters (see page 30.1).

Note that since an **IMMEDIATE** macro causes any input before it to be sent to the reader, characters typed before an **IMMEDIATE** read macro character cannot be erased by control-A or control-Q once the **IMMEDIATE** character has been typed, since they have already passed through the line buffer. However, an **INFIX** read macro can still alter some of what has been typed earlier, via its third argument.

**NONIMMEDIATE or NONIMMED**

The default. The read macro character is a normal character with respect to the line buffering, and so will not be activated until a carriage-return or matching right parenthesis or bracket is seen.

Making a read macro character be both **ALONE** and **IMMEDIATE** is a contradiction, since **ALONE** requires that the next character be input in order to see if it is a break or separator character. Thus, **ALONE** read macros are always **NONIMMEDIATE**, regardless of whether or not **IMMEDIATE** is specified.

Read macro characters can be "nested". For example, if `•` is defined by

```
(Macro (lambda (fl rdtbl)
        (eval (read fl rdtbl)))
```

and `!` is defined by

```
(SPLICE (lambda (fl rdtbl)
        (read fl rdtbl)))
```

then if the value of `FOO` is `(A B C)`, and `(X = FOO Y)` is input, `(X (A B C) Y)` will be returned. If `(X ! = FOO Y)` is input, `(X A B C Y)` will be returned.

Note: If a read macro's function calls **READ**, and the **READ** returns NIL, the function cannot distinguish the case where a **RIGHTPAREN** or **RIGHTBRACKET** followed the read macro character, (e.g. "(A B ")"), from the case where the atom NIL (or "()") actually appeared. In Interlisp-D, a **READ** inside of a read macro when the next input character is a **RIGHTPAREN** or
RIGHTBRACKET reads the character and returns NIL, just as if the READ had not occurred inside a read macro.

If a call to READ from within a read macro encounters an unmatched RIGHTBRACKET within a list, the bracket is simply put back into the buffer to be read (again) at the higher level. Thus, inputting an expression such as \((A \ B \ '\{C \ D\})\) works correctly.

**Function**

RETURNS NIL if currently not under a read macro function; otherwise the number of unmatched left parentheses or brackets.

**Function**

If FLG = NIL, turns off action of read macros in read table RDTBL. If FLG = T, turns them on. Returns previous setting.

The following read macros are standardly defined in Interlisp in the T and EDITRDTBL read tables:

* (single-quote) Returns the next expression, wrapped in a call to QUOTE; e.g., 'FOO reads as (QUOTE FOO). The macro is defined as a FIRST read macro, so that the quote character has no effect in the middle of a symbol. The macro is also ignored if the quote character is immediately followed by a separator character. Defined in T and EDITRDTBL.

control-Y Defined in T and EDITRDTBL. Returns the result of evaluating the next expression. For example, if the value of FOO is \((A \ B)\), then \((LIST 1 \ control-YF O O 2)\) is read as \((LIST 1 \ (A \ B) \ 2)\). Note that no structure is copied; the third element of that input expression is still EQ to the value of FOO. Control-Y can thus be used to read structures that ordinarily have no read syntax. For example, the value returned from reading \((KEY1 \ control-YARRAY \ 10)\) has an array as its second element. Control-Y can be thought of as an "un-quote" character. The choice of character to perform this function is changeable with SETTERMCHARS (page 16.75).

` (backquote) Backquote makes it easier to write programs to construct complex data structures. Backquote is like quote, except that within the backquoted expression, forms can be evaluated. The general idea is that the backquoted expression is a "template" containing some constant parts (as with a quoted form) and some parts to be filled in by evaluating something. Unlike with control-Y, however, the evaluation occurs not at the time the form is read, but at the time the backquoted expression is evaluated. That is, the backquote macro returns an expression which, when evaluated, produces the desired structure.

Within the backquoted expression, the character ",@" (comma) introduces a form to be evaluated. The value of a form preceded by ",@" is to be spliced in, using APPEND. If it is permissible to
destroy the list being spliced in (i.e., NCONC may be used in the translation), then ",,", can be used instead of ",,@,"

For example, if the value of FOO is (1 2 3 4), then the form

\[ '(A ,(CAR FOO) ,@(CDDR FOO) D E) \]

evaluates to (A 1 3 4 D E); it is logically equivalent to writing

\[
(CONS 'A  
(CONS (CAR FOO)  
(APPEND (CDDR FOO) '(D E))))
\]

Backquote is particularly useful for writing macros. For example, the body of a macro that refers to X as the macro's argument list might be

\[
'(COND  
((FIXP ,(CAR X))  
,(CADR X))  
(T ,,(CDDR X)))
\]

which is equivalent to writing

\[
(LIST 'COND  
(LIST (LIST 'FIXP (CAR X))  
(CADR X))  
(CONS 'T (CDDR X)))
\]

Note that comma does not have any special meaning outside of a backquote context.

For users without a backquote character on their keyboards, backquote can also be written as |' (vertical-bar, quote).

? implements the ?= command for on-line help regarding the function currently being "called" in the typein (see page 26.33).

| (vertical bar) When followed by an end of line, tab or space, | is ignored, i.e., treated as a separator character, enabling the editor's CHANGECHAR feature (page 26.49). Otherwise it is a "dispatching" read macro whose meaning depends on the character(s) following it. The following are currently defined:

' (quote) -- A synonym for backquote.

. (period) -- Returns the evaluation of the next expression, i.e., this is a synonym for control-Y.

, (comma) -- Returns the evaluation of the next expression at load time, i.e., the following expression is quoted in such a manner that the compiler treats it as a literal whose value is not determined until the compiled expression is loaded.

O or o (the letter O) -- Treats the next number as octal, i.e., reads it in radix 8. For example, |o12 = 10 (decimal).

B or b -- Treats the next number as binary, i.e., reads it in radix 2. For example, |b101 = 5 (decimal).
X or x -- Treats the next number as hexadecimal, i.e., reads it in radix 16. The upper-case letters A though F are used as the digits after 9. For example, \texttt{x1A} = 26 (decimal).

R or r -- Reads the next number in the radix specified by the (decimal) number that appears between the \texttt{I} and the \texttt{R}. When inputting a number in a radix above ten, the upper-case letters A through Z can be used as the digits after 9 (but there is no digit above Z, so it is not possible to type all base-99 digits). For example, \texttt{3r120} reads 120 in radix 3, returning 15.

(, {, ↑ -- Used internally by \texttt{HPRINT} and \texttt{HREAD} (page 25.17) to print and read unusual expressions.

The dispatching characters that are letters can appear in either upper or lower case.
# 26. User Input/Output Packages

## 26.1. Inspector

26.1.1. Calling the Inspector 26.2
26.1.2. Multiple Ways of Inspecting 26.2
26.1.3. Inspect Windows 26.3
26.1.4. Inspect Window Commands 26.4
26.1.5. Interaction With Break Windows 26.5
26.1.6. Controlling the Amount Displayed During Inspection 26.5
26.1.7. Inspect Macros 26.6
26.1.8. INSPECTWs 26.6

## 26.2. PROMPTFORWORD

## 26.3. ASKUSER

26.3.1. Format of KEYLST 26.13
26.3.2. Options 26.15
26.3.3. Operation 26.17
26.3.4. Completing a Key 26.18
26.3.5. Special Keys 26.19
26.3.6. Startup Protocol and Typeahead 26.20

## 26.4. TTYIN Display Typein Editor

26.4.1. Entering Input With TTYIN 26.22
26.4.3. Display Editing Commands 26.25
26.4.4. Using TTYIN for Lisp Input 26.28
26.4.5. Useful Macros 26.29
26.4.6. Programming With TTYIN 26.29
26.4.7. Using TTYIN as a General Editor 26.32
26.4.8. ? = Handler 26.33
26.4.9. Read Macros 26.34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.4.10</td>
<td>Assorted Flags</td>
<td>26.36</td>
</tr>
<tr>
<td>26.4.11</td>
<td>Special Responses</td>
<td>26.38</td>
</tr>
<tr>
<td>26.4.12</td>
<td>Display Types</td>
<td>26.38</td>
</tr>
<tr>
<td>26.5.</td>
<td>Prettyprint</td>
<td>26.39</td>
</tr>
<tr>
<td>26.5.1</td>
<td>Comment Feature</td>
<td>26.42</td>
</tr>
<tr>
<td>26.5.2</td>
<td>Comment Pointers</td>
<td>26.44</td>
</tr>
<tr>
<td>26.5.3</td>
<td>Converting Comments to Lower Case</td>
<td>26.46</td>
</tr>
<tr>
<td>26.5.4</td>
<td>Special Prettyprint Controls</td>
<td>26.47</td>
</tr>
</tbody>
</table>
This chapter presents a number of packages that have been developed for displaying and allowing the user to enter information. These packages are used to implement the user interface of many system facilities.

- The Inspector (below) provides a window-based facility for displaying and changing the fields of a data object.
- PROMPTFORWORD (page 26.9) is a function used for entering a simple string of characters. Basic editing and prompting facilities are provided.
- ASKUSER (page 26.12) provides a more complicated prompting and answering facility, allowing a series of questions to be printed. Prompts and argument completion are supported.
- TTYIN (page 26.22) is a display typein editor, that provides complex text editing facilities when entering an input line.
- PRETTYPRINT (page 26.40) is used for printing function definitions and other list structures, using multiple fonts and indenting lines to show the structure of the list.

### 26.1 Inspector

The Inspector provides a display-oriented facility for looking at and changing arbitrary Interlisp-D data structures. The inspector can be used to inspect all user datatypes and many system datatypes (although some objects such as numbers have no inspectable structure). The inspector displays the field names and values of an arbitrary object in a window that allows setting of the properties and further inspection of the values. This latter feature makes it possible to "walk" around all of the data structures in the system at the touch of a button. In addition, the inspector is integrated with the break package to allow inspection of any object on the stack and with the display and teletype structural editors to allow the editors to be used to "inspect" list structures and the inspector to "edit" datatypes.

The underlying mechanisms of the data inspector have been designed to allow their use as specialized editors in user applications. This functionality is described at the end of this section.
Note: Currently, the inspector does not have UNDOing. Also, variables whose values are changed will not be marked as such.

26.1.1 Calling the Inspector

There are several ways to open an inspect window onto an object. In addition to calling INSPECT directly (below), the inspector can also be called by buttoning an Inspect command inside an existing inspector window. Finally, if a non-list is edited with EDITDEF (page 17.27), the inspector is called. This also causes the inspector to be called by the Dedit command from the display editor or the EV command from the teletype editor if the selected piece of structure is a non-list.

(INSP\textsc{ect} \textsc{object} \textsc{astype} \textsc{where}) \hspace*{1cm} [Function]

Creates an inspect window onto \textsc{object}. If \textsc{astype} is given, it will be taken as the record type of \textsc{object}. This allows records to be inspected with their property names. If \textsc{astype} is \textsc{nil}, the data type of \textsc{object} will be used to determine its property names in the inspect window.

\textsc{where} specifies the location of the inspect window. If \textsc{where} is \textsc{nil}, the user will be prompted for a location. If \textsc{where} is a window, it will be used as the inspect window. If \textsc{where} is a region, the inspect window will be created in that region of the screen. If \textsc{where} is a position, the inspect window will have its lower left corner at that position on the screen.

\textsc{ins\textsc{pect}} returns the inspect window onto \textsc{object}, or \textsc{nil} if no inspection took place.

(INSP\textsc{ect\textsc{code}} \textsc{fn} \textsc{where} \hspace*{1cm} [Function])

Opens a window and displays the compiled code of the function \textsc{fn} using \textsc{printf\textsc{code}}. The window is scrollable.

\textsc{where} determines where the window should appear. It can be a position, a region, or a window. If \textsc{nil}, the user is prompted to specify the position of the window.

Note: If the Tedit library package is loaded, \textsc{inspect\textsc{code}} uses it to create the code inspector window. Also, if \textsc{inspect\textsc{code}} is called to inspect the frame name in a break window (page 14.3), the location in the code that the frame's PC indicates it was executing at the time is highlighted.

26.1.2 Multiple Ways of Inspecting

For some datatypes there is more than one aspect that is of interest or more than one method of inspecting the object. In
these cases, the inspector will bring up a menu of the possibilities and wait for the user to select one.

If the object is a litatom, the commands are the types for which the litatom has definitions as determined by HASDEF. Some typical commands are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNS</td>
<td>Edit the definition of the selected litatom.</td>
</tr>
<tr>
<td>VARS</td>
<td>Inspect the value.</td>
</tr>
<tr>
<td>PROPS</td>
<td>Inspect the property list.</td>
</tr>
</tbody>
</table>

If the object is a list, there will be choice of how to inspect the list:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspect</td>
<td>Opens an inspect window in which the properties are numbers and the values are the elements of the list.</td>
</tr>
<tr>
<td>TtyEdit</td>
<td>Calls the teletype list structure editor on the list (page 16.1).</td>
</tr>
<tr>
<td>DisplayEdit</td>
<td>Calls the DEdit display editor on the list (page 16.1).</td>
</tr>
<tr>
<td>As a PLIST</td>
<td>Inspects the list as a property list, if the list is in property list form: (((PROP_1 VAL_1) \ldots (PROP_N VAL_N))).</td>
</tr>
<tr>
<td>As an ALIST</td>
<td>Inspects the list as an association-list, if the list is in ASSOC list form: ((PROP_1 VAL_1 \ldots PROP_N VAL_N)).</td>
</tr>
<tr>
<td>As a record</td>
<td>Brings up a submenu with all of the RECORDs in the system and inspect the list with the one chosen.</td>
</tr>
<tr>
<td>As a &quot;record type&quot;</td>
<td>Inspects the list as the record of the type named in its CAR, if the CAR of the list is the name of a TYPERECORD (page 8.7).</td>
</tr>
</tbody>
</table>

If the object is a bitmap, the choice is between inspecting the bitmap's contents with the bitmap editor (EDITBM) or inspecting the bitmap's fields.

Other datatypes may include multiple methods for inspecting objects of that type.

### 26.1.3 Inspect Windows

An inspect window displays two columns of values. The lefthand column lists the property names of the structure being inspected. The righthand column contains the values of the properties named on the left. For variable length data such as lists and arrays, the "property names" are numbers from 1 to the length of the inspected item and the values are the corresponding elements. For arrays, the property names are the array element numbers and the values are the corresponding elements of the array.

For large lists or arrays, or datatypes with many fields, the initial window may be too small to contain all of them. In these cases, the unseen elements can be scrolled into view (from the bottom) or the window can be reshaped to increase its size.
In an inspect window, the LEFT button is used to select things, the MIDDLE button to invoke commands that apply to the selected item. Any property or value can be selected by pointing the cursor directly at the text representing it, and clicking the LEFT button. There is one selected item per window and it is marked by having its surrounding box inverted.

The options offered by the MIDDLE button depend on whether the selection is a property or a value. If the selected item is a value, the options provide different ways of inspecting the selected structure. The exact commands that are given depend on the type of the value.

If the selected item is a property name, the command SET will appear. If selected, the user will be asked to type in an expression, and the selected property will be set to the result of evaluating the read form. The evaluation of the read form and the replacement of the selected item property will appear as their own history events and are individually undoable. Properties of system datatypes cannot be set. (There are often consistency requirements which can be inadvertently violated in ways that crash the system. This may be true of some user datatypes as well, however the system doesn’t know which ones. Users are advised to exercise caution.)

It is possible to copy-select property names or values out of an inspect window. Litatoms, numbers and strings are copied as they are displayed. Unprintable objects (such as bitmaps, etc.) come out as an appropriate system expression, such that if is evaluated, the object is re-created.

26.1.4 Inspect Window Commands

By pressing the MIDDLE button in the title of the inspect window, a menu of commands that apply to the inspect window is brought up:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReFetch</td>
<td>An inspect window is not automatically updated when the structure it is inspecting is changed. The &quot;ReFetch&quot; command will refetch all of the fields of the object being inspected in the inspect window.</td>
</tr>
<tr>
<td>IT&lt;datum</td>
<td>Sets the variable IT to the object being inspected in the inspect window.</td>
</tr>
</tbody>
</table>
IT←selection

[Inspect Window Command]

Sets the variable IT to the property name or value currently selected in the inspect window.

26.1.5 Interaction With Break Windows

The break window facility (page 14.3) knows about the inspector in the sense that the backtrace frame window is an inspect window onto the frame selected from the back trace menu during a break. Thus you can call the inspector on an object that is bound on the stack by selecting its frame in the back trace menu, selecting its value with the LEFT button in the back trace frame window, and selecting the inspect command with the MIDDLE button in the back trace frame window. The values of variables in frames can be set by selecting the variable name with the LEFT button and then the "Set" command with the MIDDLE button.

Note: The inspector will only allow the setting of named variables. Even with this restriction it is still possible to crash the system by setting variables inside system frames. Exercise caution in setting variables in other than your own code.

26.1.6 Controlling the Amount Displayed During Inspection

The amount of information displayed during inspection can be controlled using the following variables:

MAXINSPECTCDRLEVEL

[Variable]

The inspector prints only the first MAXINSPECTCDRLEVEL elements of a long list, and will make the tail containing the unprinted elements the last item. The last item can be inspected to see further elements. Initially 50.

MAXINSPECTARRAYLEVEL

[Variable]

The inspector prints only the first MAXINSPECTARRAYLEVEL elements of an array. The remaining elements can be inspected by calling the function (INSPECT/ARRAY ARRAY BEGINOFFSET) which inspects the BEGINOFFSET through the BEGINOFFSET + MAXINSPECTARRAYLEVEL elements of ARRAY. Initially 300.

INSPECTPRINTLEVEL

[Variable]

When printing the values, the inspector resets PRINTLEVEL (page 25.11) to the value of INSPECTPRINTLEVEL. Initially (2.5).
26.1.7 Inspect Macros

The Inspector can be extended to inspect new structures and datatypes by adding entries to the list INSPECTMACROS. An entry should be of the form (OBJECTTYPE . INSPECTINFO). OBJECTTYPE is used to determine the types of objects that are inspected with this macro. If OBJECTTYPE is a litatom, the INSPECTINFO will be used to inspect items whose type name is OBJECTTYPE. If OBJECTTYPE is a list of the form (FUNCTION DATUM-PREDICATE), DATUM-PREDICATE will be APPLIED to the item and if it returns non-NIL, the INSPECTINFO will be used to inspect the item.

INSPECTINFO can be one of two forms. If INSPECTINFO is a litatom, it should be a function that will be applied to three arguments (the item being inspected, OBJECTTYPE, and the value of WHERE passed to INSPECT) that should do the inspection. If INSPECTINFO is not a litatom, it should be a list of (PROPERTIES FETCHFN STOREFN PROCOMMANDFN VALUECOMMANDFN TITLECOMMANDFN TITLE SELECTIONFN WHERE PROPNAMEFN) where the elements of this list are the arguments for INSPECTW.CREATE, described below. From this list, the WHERE argument will be evaluated; the others will not. If WHERE is NIL, the value of WHERE that was passed to INSPECT will be used.

Examples:

The entry ((FUNCTION MYATOMP) PROPNAME GETPROP PUTPROP) on INSPECTMACROS would cause all objects satisfying the predicate MYATOMP to have their properties inspected with GETPROP and PUTPROP. In this example, MYATOMP should make sure the object is a litatom.

The entry (MYDATATYPE . MYINSPECTFN) on INSPECTMACROS would cause all datatypes of type MYDATATYPE to be passed to the function MYINSPECTFN.

26.1.8 INSPECTWs

The inspector is built on the abstraction of an INSPECTW. An INSPECTW is a window with certain window properties that display an object and respond to selections of the object's parts. It is characterized by an object and its list of properties. An INSPECTW displays the object in two columns with the property
names on the left and the values of those properties on the right. An INSPECTW supports the protocol that the LEFT mouse button can be used to select any property name or property value and the MIDDLE button calls a user provided function on the selected value or property. For the Inspector application, this function puts up a menu of the alternative ways of inspecting values or of the ways of setting properties. INSPECTWs are created with the following function:

\[
\text{INSPECTW.CREATE DATUM PROPERTIES FETCHFN STOREFN PROPCOMMANDFN VALUECOMMANDFN TITLECOMMANDFN TITLE SELECTIONFN WHERE PROPPRINTFN) [Function]}
\]

Creates an INSPECTW that views the object DATUM. If PROPERTIES is a list, it is taken as the list of properties of DATUM to display. If PROPERTIES is a litatom, it is APPLIED to DATUM and the result is used as the list of properties to display.

FETCHFN is a function of two arguments (OBJECT PROPERTY) that should return the value of the PROPERTY property of OBJECT. The result of this function will be printed (with PRIN2) in the INSPECTW as the value.

STOREFN is a function of three arguments (OBJECT PROPERTY NEWVALUE) that changes the PROPERTY property of OBJECT to NEWVALUE. It is used by the default PROPCOMMANDFN and VALUECOMMANDFN to change the value of a property and also by the function INSPECTW.REPLACE (described below). This can be NIL if the user provides command functions which do not call INSPECTW.REPLACE. Each replace action will be a separate event on the history list. Users are encouraged to provide UNDOable STOREFNs.

PROPCOMMANDFN is a function of three arguments (PROPERTY OBJECT INSPECTW) which gets called when the user presses the MIDDLE button and the selected item in the INSPECTW is a property name. PROPERTY will be the name of the selected property, OBJECT will be the datum being viewed, and INSPECTW will be the window. If PROPCOMMANDFN is a string, it will get printed in the PROMPTWINDOW when the MIDDLE button is pressed. This provides a convenient way to notify the user about disabled commands on the properties. DEFAULT:INSPECTWPROPCOMMANDFN, the default PROPCOMMANDFN, will present a menu with the single command Set on it. If selected, the Set command will read a value from the user and set the selected property to the result of EVALuating this read value.

VALUECOMMANDFN is a function of four arguments (VALUE PROPERTY OBJECT INSPECTW) that gets called when the user presses the MIDDLE button and the selected item in the
INSPECTW is a property value. VALUE will be the selected value (as returned by FETCHFN), PROPERTY will be the name of the property VALUE is the value of, OBJECT will be the datum being viewed, and INSPECTW will be the INSPECTW window. DEFAULT:INSPECTW.VALUECOMMANDFN, the default VALUECOMMANDFN, will present a menu of possible ways of inspecting the value and create a new Inspect window if one of the menu items is selected.

TITLECOMMANDFN is a function of two arguments (INSPECTW OBJECT) which gets called when the user presses the MIDDLE button and the cursor is in the title or border of the inspect window INSPECTW. This command function is provided so that users can implement commands that apply to the entire object. The default TITLECOMMANDFN (DEFAULT:INSPECTW.TITLECOMMANDFN) presents a menu with the commands ReFetch, ITdatum, and ITselection (see page 26.4).

TITLE specifies the title of the window. If TITLE is NIL, the title of the window will be the printed form of DATUM followed by the string " Inspector". If TITLE is the litatom DON'T, the inspect window will not have a title. If TITLE is any other litatom, it will be aplyed to the DATUM and the potential inspect window (if it is known). If this result is the litatom DON'T, the inspect window will not have a title; otherwise the result will be used as a title. If TITLE is not a litatom, it will be used as the title.

SELECTIONFN is a function of three arguments (PROPERTY VALUEFLG INSPECTW) which gets called when the user releases the left button and the cursor is on one of the items. The SELECTIONFN allows a program to take action on the user's selection of an item in the inspect window. At the time this function is called, the selected item has been "selected". The function INSPECTW.SELECTITEM (described below) can be used to turn off this selection. PROPERTY will be the name of the property of the selected item. VALUEFLG will be NIL if the selected item is the property name; T if the selected item is the property value.

WHERE indicates where the inspect window should go. Its interpretation is described in INSPECT (page 26.2).

PROPPRINTFN is a function of two arguments (PROPERTY DATUM) which gets called to determine what to print in the property place for the property PROPERTY. If PROPPRINTFN returns NIL, no property name will be printed and the value will be printed to the left of the other values.

An inspect window uses the following window property names to hold information: DATUM, FETCHFN, STOREFN, PROPCOMMANDFN, VALUECOMMANDFN, SELECTIONFN,
PROPPRINTFN, INSPECTWTITLE, PROPERTIES, CURRENTITEM and SELECTABLEITEMS.

\textbf{(INSPECTW.REDISPLAY INSPECTW PROPS \textemdash\textemdash)} \textbf{[Function]}

Updates the display of the objects being inspected in \textit{INSPECTW}. If \textit{PROPS} is a property name or a list of property names, only those properties are updated. If \textit{PROPS} is \textit{NIL}, all properties are redisplayed. This function is provided because inspect windows do not automatically update their display when the object they are showing changes.

This function is called by the ReFetch command in the title command menu of an \textit{INSPECTW} (page 26.4).

\textbf{(INSPECTW.REPLACE INSPECTW PROPERTY NEWVALUE)} \textbf{[Function]}

Calls the \textit{STOREFN} of the inspect window \textit{INSPECTW} to change the property named \textit{PROPERTY} to the value \textit{NEWVALUE} and updates the display of \textit{PROPERTY}'s value in the display. This provides a functional interface for user \textit{PROPCOMMANDFN}s.

\textbf{(INSPECTW.SELECTITEM INSPECTW PROPERTY VALUEFLG)} \textbf{[Function]}

Sets the selected item in an inspect window. The item is inverted on the display and put on the window property \textit{CURRENTITEM} of \textit{INSPECTW}. If \textit{INSPECTW} has a \textit{CURRENTITEM}, it is deselected. \textit{PROPERTY} is the name of the property of the selected item. \textit{VALUEFLG} is \textit{NIL} if the selected item is the property name; \textit{T} if the selected item is the property value. If \textit{PROPERTY} is \textit{NIL}, no item will be selected. This provides a way of deselecting all items.

26.2 \textbf{PROMPTFORWORD}

\textbf{PROMPTFORWORD} is a function that reads in a sequence of characters, generally from the keyboard, without involving \textit{READ}-like syntax. A user can supply a prompting string, as well as a "candidate" string, which is printed and used if the user types only a word terminator character (or doesn't type anything before a given time limit). As soon as any characters are typed the "candidate" string is erased and the new input takes its place.

\textbf{PROMPTFORWORD} accepts user type-in until one of the "word terminator" characters is typed. Normally, the word terminator characters are \textit{EOL}, \textit{ESCAPE}, \textit{LF}, \textit{SPACE}, or \textit{TAB}. This list can be changed using the \textit{TERMINCHAR.LST} argument to
PROMPTFORWORD, for example if it is desirable to allow the user to input lines including spaces.

PROMPTFORWORD also recognizes the following special characters:

Control-A, Backspace, or DELETE
Any of these characters deletes the last character typed and appropriately erases it from the echo stream if it is a display stream.

Control-Q
Erases all the type-in so far.

Control-R
Reprints the accumulated string.

Control-V
"Quotes" the next character: after typing Control-V, the next character typed is added to the accumulated string, regardless of any special meaning it has. Allows the user to include editing characters and word terminator characters in the accumulated string.

Control-W
Erases the last word.

? Calls up a "help" facility. The action taken is defined by the GENERATE?LIST.FN argument to PROMPTFORWORD (see below). Normally, this prints a list of possible candidates.

(PROMPTFORWORD PROMPT.STR CANDIDATE.STR GENERATE?LIST.FN ECHO.CHANNEL DONTECHOTYPEIN.FLG URGENCY.OPTION TERMINCHARS.LST KEYBD.CHANNEL) [Function]

PROMPTFORWORD has a multiplicity of features, which are specified through a rather large number of input arguments, but the default settings for them (i.e., when they aren't given, or are given as NIL) is such to minimize the number needed in the average case, and an attempt has been made to order the more frequently non-defaulted arguments at the beginning of the argument list. The default input and echo are both to the terminal; the terminal table in effect during input allows most control characters to be INDICATE'd.

PROMPTFORWORD returns NIL if a null string is typed; this would occur when no candidate is given and only a terminator is typed, or when the candidate is erased and a terminator is typed with no other input still un-erased. In all other cases, PROMPTFORWORD returns a string.

PROMPTFORWORD is controlled through the following arguments:

PROMPT.STR
If non-NIL, this is coerced to a string and used for prompting; an additional space is output after this string.

CANDIDATE.STR
If non-NIL, this is coerced to a string and offered as initial contents of the input buffer.

GENERATE?LIST.FN
If non-NIL, this is either a string to be printed out for help, or a function to be applied to PROMPT.STR and CANDIDATE.STR (after both have been coerced to strings), and which should
return a list of potential candidates. The help string or list of potential candidates will then be printed on a separate line, the prompt will be restarted, and any type-in will be re-echoed.

Note: If GENERATE?LIST.FN is a function, its value list will be cached so that it will be run at most once per call to PROMPTFORWORD.

**ECHO.CHANNEL**
Coerced to an output stream; NIL defaults to T, the "terminal output stream", normally (TTYDISPLAYSTREAM). To achieve echoing to the "current output stream", use (GETSTREAM NIL 'OUTPUT). If echo is to a display stream, it will have a flashing caret showing where the next input is to be echoed.

**DONTECHOTYPEIN.FLG**
If T, there is no echoing of the input characters. If the value of DONTECHOTYPEIN.FLG is a single-character atom or string, that character is echoed instead of the actual input. For example, LOGIN prompts for a password with DONTECHOTYPEIN.FLG being "*".

**URGENCY.OPTION**
If NIL, PROMPTFORWORD quietly wait for input, as READ does; if a number, this is the number of seconds to wait for the user to respond (if timeout is reached, then CANDIDATE.WORD is returned, regardless of any other type-in activity); if T, this means to wait forever, but periodically flash the window to alert the user; if TTY, then PROMPTFORWORD grabs the TTY immediately. When URGENCY.OPTION = TTY, the cursor is temporarily changed to a different shape to indicate the urgent nature of the request.

**TERMINCHARS.LST**
This is list of "word terminator" character codes; it defaults to (CHARCODE (EOL ESCAPE LF SPACE TAB)). This may also be a single character code.

**KEYBD.CHANNEL**
If non-NIL, this is coerced to a stream, and the input bytes are taken from that stream. NIL defaults to the keyboard input stream. Note that this is not the same as the terminal input stream T (page 25.1), which is a buffered keyboard input stream, not suitable for use with PROMPTFORWORD.

Examples:

(PROMPTFORWORD
   "What is your FOO word?" 'Mumble
   (FUNCTION (LAMBDA () '(Grumble Bletch)))
   PROMPTWINDOW NIL 30)

This first prompts the user for input by printing the first argument as a prompt into PROMPTWINDOW; then the proffered default answer, "Mumble", is printed out and the caret starts flashing just after it to indicate that the upcoming input will be echoed there. If the user fails to complete a word within 30 seconds, then the result will be the string "Mumble".

(FRESHLINE T)
This first prompts in whatever window is currently (TTYDISPLAYSTREAM), and then takes in a username; the second call prompts with "(password)" and takes in another word (the password) without proffering a candidate, echoing the typed-in characters as "*".

26.3 ASKUSER

DWIM, the compiler, the editor, and many other system packages all use ASKUSER, an extremely general user interaction package, for their interactions with the user at the terminal. ASKUSER takes as its principal argument KEYLST which is used to drive the interaction. KEYLST specifies what the user can type at any given point, how ASKUSER should respond to the various inputs, what value should be returned by ASKUSER, and is also used to present the user at any given point with a list of the possible responses. ASKUSER also takes other arguments which permit specifying a wait time, a default value, a message to be printed on entry, a flag indicating whether or not typeahead is to be permitted, a flag indicating whether the transaction is to be stored on the history list (page 13.1), a default set of options, and an (optional) input file/string.

(ASKUSER WAIT DEFAULT MESS KEYLST TYPEAHEAD LISPXRNTFLG OPTIONS LST FILE)

[Function]

WAIT is either NIL or a number (of seconds). DEFAULT is a single character or a sequence (list) of characters to be used as the default inputs for the case when WAIT is not NIL and more than WAIT seconds elapse without any input. In this case, the character(s) from DEFAULT are processed exactly as though they had been typed, except that ASKUSER first types "...".

MESS is the initial message to be printed by ASKUSER, if any, and can be a string, or a list. In the latter case, each element of the list is printed, separated by spaces, and terminated with a " ". KEYLST and OPTIONS LST are described. TYPEAHEAD is T if the user is permitted to typeahead a response to ASKUSER. NIL means any typeahead should be cleared and saved. LISPXRNTFLG determines whether or not the interaction is to be recorded on the history list. FILE can be either NIL (in which case
it defaults to the terminal input stream, T), a stream, or a string. If FILE is a string, and all of its characters are read before ASKUSER finishes, FILE will be reset to T, and the interaction will continue with ASKUSER reading from the terminal.

All input operations take place from FILE until an unacceptable input is encountered, i.e., one that does not conform to the protocol defined by KEYLST. At that point, FILE is set to T, DEFAULT is set to NIL, the input buffer is cleared, and a bell is rung. Unacceptable inputs are not echoed.

The value of ASKUSER is the result of packing all the keys that were matched, unless the RETURN option is specified (page 26.15).

(MAKEKEYLST LST DEFAULTKEY LCASEFLG AUTOCOMPLETEFLG) [Function]

LST is a list of atoms or strings. MAKEKEYLST returns an ASKUSER KEYLST which will permit the user to specify one of the elements on LST by either typing enough characters to make the choice unambiguous, or else typing a number between 1 and N, where N is the length of LST.

For example, if ASKUSER is called with KEYLST = (MAKEKEYLST 'CONNECT SUPPORT COMPILE), then the user can type C-O-N, S, C-O-M, 1, 2, or 3 to indicate one of the three choices.

If LCASEFLG = T, then echoing of upper case elements will be in lower case (but the value returned will still be one of the elements of LST). If DEFAULTKEY is non-NIL, it will be the last key on the KEYLST. Otherwise, a key which permits the user to indicate "No - none of the above" choices, in which case the value returned by ASKUSER will be NIL.

AUTOCOMPLETEFLG is used as the value of the AUTOCOMPLETEFLG option of the resulting key list.

26.3.1 Format of KEYLST

KEYLST is a list of elements of the form (KEY PROMPTSTRING . OPTIONS), where KEY is an atom or a string (equivalent), PROMPTSTRING is an atom or a string, and OPTIONS a list of options in property list format. The options are explained below. If an option is specified in OPTIONS, the value of the option is the next element. Otherwise, if the option is specified in the OPTIONSLSST argument to ASKUSER, its value is the next element on OPTIONSLSST. Thus, OPTIONSLSST can be used to provide default options for an entire KEYLST, rather than having to include the option at each level. If an option does not appear on either OPTIONS or OPTIONSLSST, its value is NIL.

For convenience, an entry on KEYLST of the form (KEY . ATOMISTRING), can be used as an abbreviation for (KEY
ATOM/STRING CONFIRMFLG T), and an entry of just the form KEY, i.e., a non-list, as an abbreviation for (KEY NIL CONFIRMFLG T).

As each character is read, it is matched against the currently active keys. A character matches a key if it is the same character as that in the corresponding position in the key, or, if the character is an alphabetic character, if the characters are the same without regard for upper/lower case differences, i.e. "A" matches "a" and vice versa (unless the NOCASEFLG option is T, see page 26.15). In other words, if two characters have already been input and matched, the third character is matched with each active key by comparing it with the third character of that key. If the character matches with one or more of the keys, the entries on KEYLST corresponding to the remaining keys are discarded. If the character does not match with any of the keys, the character is not echoed, and a bell is rung instead.

When a key is complete, PROMPTSTRING is printed (NIL is equivalent to "", the empty string, i.e., nothing will be printed). Then, if the value of the CONFIRMFLG option is T, ASKUSER waits for confirmation of the key by a carriage return or space. Otherwise, the key does not require confirmation.

Then, if the value of the KEVLST option is not NIL, its value becomes the new KEVLST, and the process recurses. Otherwise, the key is a "leaf," i.e., it terminates a particular path through the original, top-level KEVLST, and ASKUSER returns the result of packing all the keys that have been matched and completed along the way (unless the RETURN option is used to specify some other value, as described below).

For example, when ASKUSER is called with KEVLST=NIL, the following KEVLST is used as the default:

((Y "esCr") (N "oCr"))

This KEVLST specifies that if (as soon as) the user types Y (or y), ASKUSER echoes with Y, prompts with "esCr", and returns Y as its value. Similarly, if the user types N, ASKUSER echoes the N, prompts with "oCr", and returns N. If the user types ?, ASKUSER prints:

Yes
No

to indicate his possible responses. All other inputs are unacceptable, and ASKUSER will ring the bell and not echo or print anything.

For a more complicated example, the following is the KEVLST used for the compiler questions (page 18.1):

((ST "ore and redefine " KEVLST ("" (F . "orget exprs"))
(S . "ame as last time")

USER INPUT/OUTPUT PACKAGES
(F. "File only")
(T. "o terminal")
1
2
(Y. "es")
(N. "o")

When ASKUSER is called with this KEYLST, and the user types an S, two keys are matched: ST and S. The user can then type a T, which matches only the ST key, or confirm the S key by typing a Cr or space. If the user confirms the S key, ASKUSER prompts with "ame as last time", and returns S as its value. (Note that the confirming character is not included in the value.) If the user types a T, ASKUSER prompts with "ore and redefine", and makes ("" (F. "orget exprs")) be the new KEYLST, and waits for more input. The user can then type an F, or confirm the "" (which essentially starts out with all of its characters matched). If he confirms the "", ASKUSER returns ST as its value the result of packing ST and "". If he types F, ASKUSER prompts with "orget exprs", and waits for confirmation again. If the user then confirms, ASKUSER returns STF, the result of packing ST and F.

At any point the user can type a ? and be prompted with the possible responses. For example, if the user types S and then ?, ASKUSER will type:

STore and redefine Forget exprs
STore and redefine
Same as last time

26.3.2 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYLST</td>
<td>When a key is complete, if the value of the KEYLST option is not NIL, this value becomes the new KEYLST and the process recurses. Otherwise, the key terminates a path through the original, top-level KEYLST, and ASKUSER returns the indicated value.</td>
</tr>
<tr>
<td>CONFIRMFLG</td>
<td>If T, the key must be confirmed with either a carriage return or a space. If the value of CONFIRMFLG is a list, the confirming character may be any member of the list.</td>
</tr>
<tr>
<td>PROMPTCONFIRMFLG</td>
<td>If T, whenever confirmation is required, the user is prompted with the string &quot; [confirm] &quot;.</td>
</tr>
<tr>
<td>NOCASEFLG</td>
<td>If T, says do not perform case independent matching on alphabetic characters. If NIL, do perform case independent matching, i.e. &quot;A&quot; matches with &quot;a&quot; and vice versa.</td>
</tr>
<tr>
<td>RETURN</td>
<td>If non-NIL, EVAL of the value of the RETURN option is returned as the value of ASKUSER. Note that different RETURN options can be specified for different keys. The variable ANSWER is bound in ASKUSER to the list of keys that have been matched. In other</td>
</tr>
</tbody>
</table>
words, RETURN (PACK ANSWER) would be equivalent to what ASKUSER normally does.

NOECHOFLG
If non-NIL, characters that are matched (or automatically supplied as a result of typing $ (escape) or confirming) are not echoed, nor is the confirming character, if any. The value of NOECHOFLG is automatically NIL when ASKUSER is reading from a file or string. The decision about whether or not to echo a character that matches several keys is determined by the value of the NOECHOFLG option for the first key.

EXPLAINSTRING
If the value of the EXPLAINSTRING option is non-NIL, its value is printed when the user types a ?, rather than KEY + PROMPTSTRING. EXPLAINSTRING enables more elaborate explanations in response to a ? than what the user sees when he is prompted as a result of simply completing keys.

For example: One of the entries on the KEYLST used by ADDTOFILES? (page 17.13) is:

([] "Nowhere\r\nNOECHOFLG T
EXPLAINSTRING "] - nowhere, item is marked as a dummy\r\n")

When the user types ], ASKUSER just prints "Nowhere\r\n", i.e., the ] is not echoed. If the user types ?, the explanation corresponding to this entry will be:

] - nowhere, item is marked as a dummy

KEYSTRING
If non-NIL, characters that are matched are echoed as though the value of KEYSTRING were used in place of the key. KEYSTRING is also used for computing the value returned. The main reason for this feature is to enable echoing in lowercase.

PROMPTON
If non-NIL, PROMPTSTRING is printed only when the key is confirmed with a member of the value of PROMPTON.

COMPLETEON
When a confirming character is typed, the N characters that are automatically supplied, as specified in case (4), are echoed only when the key is confirmed with a member of the value of PROMPTON.

The PROMPTON and COMPLETEON options enable the user to construct a KEYLST which will cause ASKUSER to emulate the action of the TENEX exec. The protocol followed by the TENEX exec is that the user can type as many characters as he likes in specifying a command. The command can be completed with a carriage return or space, in which case no further output is forthcoming, or with a $ (escape), in which case the rest of the characters in the command are echoed, followed by some prompting information. The following KEYLST would handle the TENEX COPY and CONNECT commands:

((COPY " (FILE LIST) "
PROMPTON ($)
COMPLETEON ($) CONFIRMFLG ($) (CONNECT " (TO DIRECTORY) " PROMPTON ($) COMPLETEON ($) CONFIRMFLG ($))

AUTOCOMPLETF LG If the value of the AUTOCOMPLETF LG option is not NIL, ASKUSER will automatically supply unambiguous characters whenever it can, i.e., ASKUSER acts as though $ (escape) were typed after each character (except that it does not ring the bell if there are no unambiguous characters).

MACROCHARS value is a list of dotted pairs of form (CHARACTER . FORM). When CHARACTER is typed, and it does not match any of the current keys, FORM is evaluated and nothing else happens, i.e. the matching process stays where it is. For example, ? could have been implemented using this option. Essentially MACROCHARS provides a read macro facility while inside of ASKUSER (since ASKUSER does READC's, read macros defined via the readtable are never invoked).

EXPLAINDELMITER value is what is printed to delimit explanation in response to ?. Initially a carriage return, but can be reset, e.g. to a comma, for more linear output.

26.3.3 Operation

All input operations are executed with the terminal table in the variable ASKUSERTTBL, in which (1) (CONTROL T) has been executed (see page 30.10), so that ASKUSER can interact with the user after each character is typed; and (2) (ECHOMODE NIL) has been executed (see page 30.7), so that ASKUSER can decide after it reads a character whether or not the character should be echoed, and with what, e.g. unacceptable inputs are never echoed.

As each character is typed, it is matched against KEYLST, and appropriate echoing and/or prompting is performed. If the user types an unacceptable character, ASKUSER simply rings the bell and allows him to try again.

At any point, the user can type ? and receive a list of acceptable responses at that point (generated from KEYLST), or type a control-A, control-Q, control-X, or delete, which causes ASKUSER to reinitialize, and start over.

Note that ?, Control-A, Control-Q, and Control-X will not work if they are acceptable inputs, i.e., they match one of the keys on KEYLST. Delete will not work if it is an interrupt character, in which case it is not seen by ASKUSER.
When an acceptable sequence is completed, ASKUSER returns the indicated value.

26.3.4 Completing a Key

The decision about when a key is complete is more complicated than simply whether or not all of its characters have been matched. In the compiler questions example above, all of the characters in the S key are matched as soon as the S has been typed, but until the next character is typed, ASKUSER does not know whether the S completes the S key, or is simply the first character in the ST key. Therefore, a key is considered to be complete when:

(1) All of its characters have been matched and it is the only key left, i.e., there are no other keys for which this key is a substring.

(2) All of its characters have been matched and a confirming character is typed.

(3) All of its characters have been matched, and the value of the CONFIRMFLG option is NIL, and the value of the KEYLST option is not NIL, and the next character matches one of the keys on the value of the KEYLST option.

(4) There is only one key left and a confirming character is typed. Note that if the value of CONFIRMFLG is T, the key still has to be confirmed, regardless of whether or not it is complete. For example, if the first entry in the above example were instead

(ST "ore and redefine " CONFIRMFLG T KEYLST ("" (F "orget exprs")

and the user wanted to specify the STF path, he would have to type ST, then confirm before typing F, even though the ST completed the ST key by the rule in case (1). However, he would be prompted with "ore and redefine" as soon as he typed the T, and completed the ST key.

Case (2) says that confirmation can be used to complete a key in the case where it is a substring of another key, even where the value of CONFIRMFLG is NIL. In this case, the confirming character doubles as both an indicator that the key is complete, and also to confirm it, if necessary. This situation corresponds to typing $SCF$ in the above example.

Case (3) says that if there were another entry whose key was STX in the above example, so that after the user typed ST, two keys, ST and STX, were still active, then typing F would complete the ST key, because F matches the (F "$orget exprs") entry on the value of the KEYLST option of the ST entry. In this case, "ore and redefine" would be printed before the F was echoed.
Finally, case (4) says that the user can use confirmation to specify completion when only one key is left, even when all of its characters have not been matched. For example, if the first key in the above example were STORE, the user could type ST and then confirm, and ORE would be echoed, followed by whatever prompting was specified. In this case, the confirming character also confirms the key if necessary, so that no further action is required, even when the value of CONFIRMFLG is T.

Case (4) permits the user not to have to type every character in a key when the key is the only one left. Even when there are several active keys, the user can type $ (escape) to specify the next \( N>0 \) common characters among the currently active keys. The effect is exactly the same as though these characters had been typed. If there are no common characters in the active keys at that point, i.e. \( N=0 \), the \$ is treated as an incorrect input, and the bell is rung. For example, if KEYLST is (CLISPFLG CLISPIFYPACKFLG CLISPTRANFLG), and the user types C followed by $, ASKUSER will supply the L, I, S, and P. The user can then type F followed by a carriage return or space to complete and confirm CLISPFLG, as per case (4), or type I, followed by $, and ASKUSER will supply the F, etc. Note that the characters supplied do not have to correspond to a terminal segment of any of the keys. Note also that the $ does not confirm the key, although it may complete it in the case that there is only one key active.

If the user types a confirming character when several keys are left, the next \( N>0 \) common characters are still supplied, the same as with $. However, ASKUSER assumes the intent was to complete a key, i.e., case (4) is being invoked. Therefore, after supplying the next \( N \) characters, the bell is rung to indicate that the operation was not completed. In other words, typing a confirming character has the same effect as typing an $ in that the next \( N \) common characters are supplied. Then, if there is only one key left, the key is complete (case 4) and confirmation is not required. If the key is not the only key left, the bell is rung.

### 26.3.5 Special Keys

| & | This can be used as a key to match with any single character, provided the character does not match with some other key at that level. For the purposes of echoing and returning a value, the effect is the same as though the character that were matched actually appeared as the key. |
| $ (escape) | This can be used as a key to match with the result of a single call to READ. For example, if the KEYLST were: |

\[
\text{COPY " (FILE LIST) "} \\
\text{PROMPTON ($)}
\]
then if the user typed \texttt{COP FOO} , \texttt{(COPY FOO)} would be returned as the value of \texttt{ASKUSER}. One advantage of using \texttt{$}, rather than having the calling program perform the \texttt{READ}, is that the call to \texttt{READ} from inside \texttt{ASKUSER} is \texttt{ERRORSET} protected, so that the user can back out of this path and reinitialize \texttt{ASKUSER}, e.g. to change from a \texttt{COPY} command to a \texttt{CONNECT} command, simply by typing control-E.

\textbf{\texttt{$$} (escape, escape)}

This can be used as a key to match with the result of a single call to \texttt{READLINE}.

\textbf{A list}

A list can be used as a key, in which case the list/form is evaluated and its value "matches" the key. This feature is provided primarily as an escape hatch for including arbitrary input operations as part of an \texttt{ASKUSER} sequence. For example, the effect of \texttt{$$} (escape, escape) could be achieved simply by using \texttt{(READLINE T)} as a key.

\textbf{""}

The empty string can be used as a key. Since it has no characters, all of its characters are automatically matched. "" essentially functions as a place marker. For example, one of the entries on the \texttt{KEYLST} used by \texttt{ADDTOFILES} is:

\begin{verbatim}
("" "File/list: "
   EXPLAINSTRING "a file name or name of a function list"
   KEYLST ($)
)
\end{verbatim}

Thus, if the user types a character that does not match any of the other keys on the \texttt{KEYLST}, then the character completes the "" key, by virtue of case (4), since the character will match with the \texttt{$} in the inner \texttt{KEYLST}. \texttt{ASKUSER} then prints ""File/list: " before echoing the character, then calls \texttt{READ}. The character will be read as part of the \texttt{READ}. The value returned by \texttt{ASKUSER} will be the value of the \texttt{READ}.

\textbf{Note:}

For \texttt{\$} (escape), \texttt{$$} (escape, escape), or a list, if the last character read by the input operation is a separator, the character is treated as a confirming character for the key. However, if the last character is a break character, it will be matched against the next key.

\section*{26.3.6 Startup Protocol and Typeahead}

Interlisp permits and encourages the user to typeahead; in actual practice, the user frequently does this. This presents a problem for \texttt{ASKUSER}. When \texttt{ASKUSER} is entered and there has been typeahead, was the input intended for \texttt{ASKUSER}, or was the interaction unanticipated, and the user simply typing ahead to
some other program, e.g. the programmer's assistant? Even where there was no typeahead, i.e., the user starts typing after the call to ASKUSER, the question remains of whether the user had time to see the message from ASKUSER and react to it, or simply began typing ahead at an inauspicious moment. Thus, what is needed is an interlock mechanism which warns the user to stop typing, gives him a chance to respond to the warning, and then allows him to begin typing to ASKUSER.

Therefore, when ASKUSER is first entered, and the interaction is to take place with a terminal, and typeahead to ASKUSER is not permitted, the following protocol is observed:

1. If there is typeahead, ASKUSER clears and saves the input buffers and rings the bell to warn the user to stop typing. The buffers will be restored when ASKUSER completes operation and returns.

2. If MESS, the message to be printed on entry, is not NIL (the typical case), ASKUSER then prints MESS if it is a string, otherwise CAR of MESS, if MESS is a list.

3. After printing MESS or CAR of MESS, ASKUSER waits until the output has actually been printed on the terminal to make sure that the user has actually had a chance to see the output. This also give the user a chance to react. ASKUSER then checks to see if anything additional has been typed in the intervening period since it first warned the user in (1). If something has been typed, ASKUSER clears it out and again rings the bell. This latter material, i.e., that typed between the entry to ASKUSER and this point, is discarded and will not be restored since it is not certain whether the user simply reacted quickly to the first warning (bell) and this input is intended for ASKUSER, or whether the user was in the process of typing ahead when the call to ASKUSER occurred, and did not stop typing at the first warning, and therefore this input is a continuation of input intended for another program.

Anything typed after (3) is considered to be intended for ASKUSER, i.e., once the user sees MESS or CAR of MESS, he is free to respond. For example, UNDO (page 13.13) calls ASKUSER when the number of undosaves are exceeded for an event with MESS = (LIST NUMBER-UNDOSAVES "undosaves, continue saving"). Thus, the user can type a response as soon as NUMBER-UNDOSAVES is typed.

4. ASKUSER then types the rest of MESS, if any.

5. Then ASKUSER goes into a wait loop until something is typed. If WAIT, the wait time, is not NIL, and nothing is typed in WAIT seconds, ASKUSER will type "..." and treat the elements of DEFAULT, the default value, as a list of characters, and begin processing them exactly as though they had been typed. If the user does type anything within WAIT seconds, he can then wait.
as long as he likes, i.e., once something has been typed, ASKUSER will not use the default value specified in DEFAULT.

If the user wants to consider his response for more than WAIT seconds, and does not want ASKUSER to default, he can type a carriage return or a space, which are ignored if they are not specified as acceptable inputs by KEYLST (see below) and they are the first thing typed.

If the calling program knows that the user is expecting an interaction with ASKUSER, e.g. another interaction preceded this one, it can specify in the call to ASKUSER that typeahead is permitted. In this case, ASKUSER simply notes whether there is any typeahead, then prints MESS and goes into a wait loop as described above.

If there is typeahead that contains unacceptable input, ASKUSER will assume that the typeahead was not intended for ASKUSER, and will restore the typeahead when it completes operation and returns.

Finally, if the interaction is not with the terminal, i.e., the optional input file/string is specified, ASKUSER simply prints MESS and begins reading from the file/string.

26.4 TTYIN Display Typein Editor

TTYIN is an Interlisp function for reading input from the terminal. It features altmode completion, spelling correction, help facility, and fancy editing, and can also serve as a glorified free text input function. This document is divided into two major sections: how to use TTYIN from the user's point of view, and from the programmer's.

TTYIN exists in implementations for Interlisp-10 and Interlisp-D. The two are substantially compatible, but the capabilities of the two systems differ (Interlisp-D has a more powerful display and allows greater access to the system primitives needed to control it effectively; it also has a mouse, greatly reducing the need for keyboard-oriented editing commands). Descriptions of both are included in this document for completeness, but Interlisp-D users may find large sections irrelevant.

26.4.1 Entering Input With TTYIN

There are two major ways of using TTYIN: (1) set LISPXREADFN to TTYIN, so the LISPX executive uses it to obtain input, and (2) call TTYIN from within a program to gather text input. Mostly
the same rules apply to both; places where it makes a difference are mentioned below.

The following characters may be used to edit your input, independent of what kind of terminal you are on. The more TTYIN knows about your terminal, of course, the nicer some of these will behave. Some functions are performed by one of several characters; any character that you happen to have assigned as an interrupt character will, of course, not be read by TTYIN. There is a (somewhat inelegant) way of changing which characters perform which functions, described under TTYINREADMACROS later on.

control-A, Backspace, Delete
Deletes a character. At the start of the second or subsequent lines of your input, deletes the last character of the previous line.

control-W
Deletes a "word". Generally this means back to the last space or parenthesis.

control-Q
Deletes the current line, or if the current line is blank, deletes the previous line.

control-R
Refreshes the current line. Two in a row refreshes the whole buffer (when doing multi-line input).

Escape
Tries to complete the current word from the spelling list provided to TTYIN, if any. In the case of ambiguity, completes as far as is uniquely determined, or rings the bell. For LISPX input, the spelling list may be USERWORDS (see discussion of TTYINCOMPLETENGL, page 26.37).

Interlisp-10 only: If no spelling list was provided, but the word begins with a "<", tries directory name completion (or filename completion if there is already a matching " >" in the current word).

If typed in the middle of a word will supply alternative completions from the SPLST argument to TTYIN (if any). ?ACTIVATEFLG (page 26.36) must be true to enable this feature.

control-F
Tops20 only: Invokes filename completion on the current "word".

close-text

control-Y
Escapes to a Lisp user exec, from which you may return by the command OK. However, when in READ mode and the buffer is non-empty, control-Y is treated as Lisp's unquote macro instead, so you have to use meta-control-Y (below) to invoke the user exec.

Open key on Xerox 1132
Middle-blank key on Xerox 1132
LF in Interlisp-10

Retrieves characters from the previous non-empty buffer when it is able to; e.g., when typed at the beginning of the line this command restores the previous line you typed at TTYIN; when typed in the middle of a line fills in the remaining text from the
old line; when typed following ↑Q or ↑W restores what those commands erased.

; If typed as the first character of the line means the line is a comment; it is ignored, and TTYIN loops back for more input.

Note: The exact behaviour of this character is determined by the value of TTYINCOMMENTCHAR (page 26.37).

control-X Goes to the end of your input (or end of expression if there is an excess right parenthesis) and returns if parentheses are balanced, beeps if not. Currently implemented in Interlisp-D only.

During most kinds of input, TTYIN is in "autofill" mode: if a space is typed near the right margin, a carriage return is simulated to start a new line. In fact, on cursor-addressable displays, lines are always broken, if possible, so that no word straddles the end of the line. The "pseudo-carriage return" ending the line is still read as a space, however; i.e., the program keeps track of whether a line ends in a carriage return or is merely broken at some convenient point. You won't get carriage returns in your strings unless you explicitly type them.

### 26.4.2 Mouse Commands [Interlisp-D Only]

The mouse buttons are interpreted as follows during TTYIN input:

**LEFT** Moves the caret to where the cursor is pointing. As you hold down LEFT, the caret moves around with the cursor; after you let up, any typein will be inserted at the new position.

**MIDDLE** Like LEFT, but moves only to word boundaries.

**RIGHT** Deletes text from the caret to the cursor, either forward or backward. While you hold down RIGHT, the text to be deleted is complemented; when you let up, the text actually goes away. If you let up outside the scope of the text, nothing is killed (this is how to "cancel" the command). This is roughly the same as CTRL-RIGHT with no initial selection (below).

If you hold down CTRL and/or SHIFT while pressing the mouse buttons, you instead get secondary selection, move selection or delete selection. You make a selection by bugging LEFT (to select a character) or MIDDLE (to select a word), and optionally extend the selection either left or right using RIGHT. While you are doing this, the caret does not move, but your selected text is highlighted in a manner indicating what is about to happen. When you have made your selection (all mouse buttons up now), lift up on CTRL and/or SHIFT and the action you have selected will occur, which is:

**SHIFT** The selected text as typein at the caret. The text is highlighted with a broken underline during selection.
TTYIN DISPLAY TYPEIN EDITOR

CTRL
Delete the selected text. The text is complemented during selection.

CTRL-SHIFT
Combines the above: delete the selected text and insert it at the caret. This is how you move text about.

You can cancel a selection in progress by pressing LEFT or MIDDLE as if to select, and moving outside the range of the text.

The most recent text deleted by mouse command can be inserted at the caret by typing Middle-blank key (on the Xerox 1132) or the Open key (on the Xerox 1108). This is the same key that retrieves the previous buffer when issued at the end of a line.

26.4.3 Display Editing Commands

On terminals with a meta key: In Interlisp-10, TTYIN reads from the terminal in binary mode, allowing many more editing commands via the meta key, in the style of TVEDIT commands. Note that due to Tenex's unfortunate way of handling typeahead, it is not possible to type ahead edit commands before TTYIN has started (i.e., before its prompt appears), because the meta bit will be thrown away. Also, since Escape has numerous other meanings in Lisp and even in TTYIN (for completion), this is not used as a substitute for the meta key.

In Interlisp-D: Users will probably have little use for most of these commands, as cursor positioning can often be done more conveniently, and certainly more obviously, with the mouse. Nevertheless, some commands, such as the case changing commands, can be useful. The <bottom-blank> key can be used as a meta key if you perform (METASHIFT T) (see page 30.22). Alternatively, you can use the variable EDITPREFIXCHAR as described in the next paragraph.

On display terminals without a meta key: If you want to type any of these commands, you need to prefix them with the "edit prefix" character. Set the variable EDITPREFIXCHAR to the character code of the desired prefix char. Type the edit prefix twice to give an "meta-escape" command. Some users of the TENEX TVEDIT program like to make escape (33Q) be the edit prefix, but this makes it somewhat awkward to ever use escape completion. EDITPREFIXCHAR is initially NIL.

On hardcopy terminals without a meta key: You probably want to ignore this section, since you won't be able to see what's going on when you issue edit commands; there is no attempt made to echo anything reasonable.

In the descriptions below, "current word" means the word the cursor is under, or if under a space, the previous word. Currently parentheses are treated as spaces, which is usually what you want, but can occasionally cause confusion in the word deletion
commands. The notation [CHAR] means meta-CHAR, if you have a meta key, or CHAR preceded by the character number EDITPREFIXCHAR if you don't. The notation $ stands for the Escape key. Most commands can be preceded by numbers or escape (means infinity), only the first of which requires the meta key (or the edit prefix). Some commands also accept negative arguments, but some only look at the magnitude of the arg. Most of these commands are taken from the display editors TVEDIT and/or E, and are confined to work within one line of text unless otherwise noted.

Cursor Movement Commands:

[delete], [bs], [<] Back up one (or n) characters.
[space], [>] Move forward one (or n) characters.
[↑] Moves up one (or n) lines.
[If] Moves down one (or n) lines.
[()] Move back one (or n) words.
[)] Move ahead one (or n) words.
[tab] Moves to end of line; with an argument moves to nth end of line; [$tab] goes to end of buffer.
[control-L] Moves to start of line (or nth previous, or start of buffer).
[[] and [[]] Go to start and end of buffer, respectively (like [$control-L] and [$tab]).
[[]] (meta-left-bracket) Moves to beginning of the current list, where cursor is currently under an element of that list or its closing paren. (See also the auto-parenthesis-matching feature below under "Flags".)
[[[]] (meta-right-bracket) Moves to end of current list.
[Sx] Skips ahead to next (or nth) occurrence of character x, or rings the bell.
[Bx] Backward search, i.e., short for [-S] or [-nS].

Buffer Modification Commands:

[Zx] Zaps characters from cursor to next (or nth) occurrence of x. There is no unzap command yet.
[A] or [R] Repeat the last S, B or Z command, regardless of any intervening input (note this differs from Tvedit's A command).
[K] Kills the character under the cursor, or n chars starting at the cursor.
[cr] When the buffer is empty is the same as <If>, i.e. restores buffer's previous contents. Otherwise is just like a <cr> (except that it also terminates an insert). Thus, [<cr><cr>] will repeat the previous input (as will <If><cr> without the meta key).
[O] Does "Open line", inserting a crlf after the cursor, i.e., it breaks the line but leaves the cursor where it is.
TTYIN DISPLAY TYPEIN EDITOR

[T] Transposes the characters before and after the cursor. When typed at the end of a line, transposes the previous two characters. Refuses to handle funny cases, such as tabs.

[G] Grabs the contents of the previous line from the cursor position onward. [nG] grabs the nth previous line.

[L] Lowercases current word, or n words on line. [$L$] lowercases the rest of the line, or if given at the end of line lowercases the entire line.

[U] Uppercases analogously.

[C] Capitalize. If you give it an argument, only the first word is capitalized; the rest are just lowercased.

[control-Q] Deletes the current line. [control-Q] deletes from the current cursor position to the end of the buffer. No other arguments are handled.

[control-W] Deletes the current word, or the previous word if sitting on a space.

[J] "Justify" this line. This will break it if it is too long, or move words up from the next line if too short. Will not join to an empty line, or one starting with a tab (both of which are interpreted as paragraph breaks). Any new line breaks it introduces are considered spaces, not carriage returns. [nJ] justifies n lines.

The linelength is defined as TTYJUSTLENGTH, ignoring any prompt characters at the margin. If TTYJUSTLENGTH is negative, it is interpreted as relative to the right margin. TTYJUSTLENGTH is initially -8 in Interlisp-D, 72 in Interlisp-10.

[$F$] "Finishes" the input, regardless of where the cursor is. Specifically, it goes to the end of the input and enters a <cr>, control-Z or "]", depending on whether normal, REPEAT or READ input is happening. Note that a "]" won't necessarily end a READ, but it seems likely to in most cases where you would be inclined to use this command, and makes for more predictable behavior.

Miscellaneous Commands:

[P] Interlisp-D: Prettyprint buffer. Clears the buffer and reprints it using prettyprint. If there are not enough right parentheses, it will supply more; if there are too many, any excess remains unprettyprinted at the end of the buffer. May refuse to do anything if there is an unclosed string or other error trying to read the buffer.

[N] Refresh line. Same as control-R. [$N$] refreshes the whole buffer; [nN] refreshes n lines. Cursor movement in TTYIN depends on TTYIN being the only source of output to the screen; if you do a control-T, or a system message appears, or line noise occurs, you may need to refresh the line for best results. In Interlisp-10, if for
some reason your terminal falls out of binary mode (e.g. can happen when returning to a Lisp running in a lower fork), Meta-<anything> is unreadable, so you'd have to type control-R instead.

[control-Y] Gets user exec. Thus, this is like regular control-Y, except when doing a READ (when control-Y is a read macro and hence does not invoke this function).

[$control-Y] Gets a user exec, but first unreads the contents of the buffer from the cursor onward. Thus if you typed at TTYIN something destined for the Lisp executive, you can do [control-L$control-Y] and give it to Lisp.

[←] Adds the current word to the spelling list USERWORDS. With zero arg, removes word. See TTYINCOMPLETEFLG (page 26.37).

Note to Datamedia, Heath users: In addition to simple cursor movement commands and insert/delete, TTYIN uses the display's cursor-addressing capability to optimize cursor movements longer than a few characters, e.g. [tab] to go to the end of the line. In order to be able to address the cursor, TTYIN has to know where it is to begin with. Lisp keeps track of the current print position within the line, but does not keep track of the line on the screen (in fact, it knows precious little about displays, much like Tenex). Thus, TTYIN establishes where it is by forcing the cursor to appear on the last line of the screen. Ordinarily this is the case anyway (except possibly on startup), but if the cursor happens to be only halfway down the screen at the time, there is a possibly unsettling leap of the cursor when TTYIN starts.

26.4.4 Using TTYIN for Lisp Input

When TTYIN is loaded, or a sysout containing TTYIN is started up, the function SETREADFN is called. If the terminal is a display, it sets LISPREADFN (page 13.36) to be TTYINREAD. If the terminal is not a display terminal, SETREADFN will set the variable to READ. (SETREADFN 'READ) will also set it to READ.

There are two principal differences between TTYINREAD and READ: (1) parenthesis balancing. The input does not activate on an exactly balancing right paren/bracket unless the input started with a paren/bracket, e.g., "USE (FOO) FOR (FIE)" will all be on one line, terminated by <cr>; and (2) read macros.

In Interlisp-10, TTYIN does not use a read table (TTYIN behaves as though using the default initial Lisp terminal input readtable), so read macros and redefinition of syntax characters are not supported; however, " " (QUOTE) and "control-Y" (EVAL) are built in, and a simple implementation of ? and ? = is supplied. Also, the TTYINREADMACROS facility described below can
supply some of the functionality of immediate read macros in
the editor.
In Interlisp-D, read macros are (mostly) supported. Immediate
read macros take effect only if typed at the end of the input (it's
not clear what their semantics should be elsewhere).

26.4.5 Useful Macros

There are two useful edit macros that allow you to use TTYIN as a
character editor: (1) ED loads the current expression into the
ttyin buffer to be edited (this is good for editing comments and
strings). Input is terminated in the usual way (by typing a
balancing right parenthesis at the end of the input, typing < cr >
at the end of an already balanced expression, or control-X
anywhere inside the balanced expression). Typing control-E or
clearing the buffer aborts ED. (2) EE is like ED but prettyprints
the expression into the buffer, and uses its own window. The
variable TTYINEDITPROMPT controls what prompt, if any, EE
uses. If it is T (initial value), no prompt is printed. EE is not
implemented in Interlisp-10.

The macro BUF loads the current expression into the buffer,
preceded by E, to be used as input however desired; as a trivial
example, to evaluate the current expression, BUF followed by a
< cr > to activate the buffer will perform roughly what the edit
macro EVAL does. Of course, you can edit the E to something
else to make it an edit command.

BUF is also defined at the executive level as a programmer's
assistant command that loads the buffer with the VALUEOF the
indicated event, to be edited as desired.

TV is a programmer's assistant command like EV [EDITV] that
performs an ED on the value of the variable.

And finally, if the event is considered "short" enough, the
programmer's assistant command FIX will load the buffer with
the event's input, rather than calling the editor. If you really
wanted the interlisp editor for your fix, you could either say FIX
EVENT - TTY; or type control-U (or whatever on tops20) once
you got TTYIN's version to force you into the editor.

26.4.6 Programming With TTYIN

(TTYIN PROMPT SPLIT HELP OPTIONS ECHOTOFILE TABS UNREADBUF RDTBL) [Function]
TTYIN prints PROMPT, then waits for input. The value returned
in the normal case is a list of all atoms on the line, with comma
and parens returned as individual atoms; OPTIONS may be used
to get a different kind of value back.
**TTYIN DISPLAY TYPEIN EDITOR**

**PROMPT** is an atom or string (anything else is converted to a string). If **NIL**, the value of **DEFAULTPROMPT**, initially "***", will be used. If **PROMPT** is **T**, no prompt will be given. **PROMPT** may also be a dotted pair (**PROMPT1** . **PROMPT2**), giving the prompt for the first and subsequent (or overflow) lines, each prompt being a string/atom or **NIL** to denote absence of prompt. The default prompt for overflow lines is "... ". Note that rebinding **DEFAULTPROMPT** gives a convenient way to affect all the "ordinary" prompts in some program module.

**SPLST** is a spelling list, i.e., a list of atoms or dotted pairs (**SYNONYM** . **ROOT**). If supplied, it is used to check and correct user responses, and to provide completion if the user types escape. If **SPLST** is one of the Lisp system spelling lists (e.g., **USERWORDS** or **SPELLINGS3**), words that are escape-completed get moved to the front, just as if a **FIXSPELL** had found them. Autocompletion is also performed when user types a break character (cr, space, paren, etc), unless one of the "nofixspell" options below is selected; i.e., if the word just typed would uniquely complete by escape, TTYIN behaves as though escape had been typed.

**HELP**, if non-**NIL**, determines what happens when the user types ? or HELP. If **HELP** = **T**, program prints back **SPLST** in suitable form. If **HELP** is any other litatom, or a string containing no spaces, it performs **(DISPLAYHELP HELP)**. Anything else is printed as is. If **HELP** is **NIL**, ? and HELP are treated as any other atoms the user types. [**DISPLAYHELP** is a user-supplied function, initially a noop; systems with a suitable HASH package, for example, have defined it to display a piece of text from a hashfile associated with the key **HELP**.]

**OPTIONS** is an atom or list of atoms chosen from among the following:

- **NOFIXSPELL**
  
  Uses **SPLST** for HELP and Escape completion, but does not attempt any **FIXSPELL**ing. Mainly useful if **SPLST** is incomplete and the caller wants to handle corrections in a more flexible way than a straight **FIXSPELL**.

- **MUSTAPPROVE**
  
  Does spelling correction, but requires confirmation.

- **CRCOMPLETE**
  
  Requires confirmation on spelling correction, but also does autocompletion on <cr> (i.e. if what user has typed so far uniquely identifies a member of **SPLST**, completes it). This allows you to have the benefits of autocompletion and still allow new words to be typed.

- **DIRECTORY** (only if **SPLST** = **NIL**)
  
  Interprets Escape to mean directory name completion [Interlisp-10 only].

- **USER**
  
  Like **DIRECTORY**, but does username completion. This is identical to **DIRECTORY** under Tenex [Interlisp-10 only].
FILE  (only if SPLST = NIL)  Interprets Escape to mean filename completion [Sumex and Tops20 only].

FIX  If response is not on, or does not correct to, SPLST, interacts with user until an acceptable response is entered. A blank line (returning NIL) is always accepted. Note that if you are willing to accept responses that are not on SPLST, you probably should specify one of the options NOXSPELL, MUSTAPPROVE or CRCOMPLETE, lest the user's new response get FIXSPELLED away without their approval.

STRING  Line is read as a string, rather than list of atoms. Good for free text.

NORAISE  Does not convert lower case letters to upper case.

NOVALUE  For use principally with the ECHOTOFILE arg (below). Does not compute a value, but returns T if user typed anything, NIL if just a blank line.

REPEAT  For multi-line input. Repeatedly prompts until user types control-Z (as in Tenex sndmsg). Returns one long list; with STRING option returns a single string of everything typed, with carriage returns (EOL) included in the string.

TEXT  Implies REPEAT, NORAISE, and NOVALUE. Additionally, input may be terminated with control-V, in which case the global flag CTRLVFLG will be set true (it is set to NIL on any other termination). This flag may be utilized in any way the caller desires.

COMMAND  Only the first word on the line is treated as belonging to SPLST, the remainder of the line being arbitrary text; i.e., "command format". If other options are supplied, COMMAND still applies to the first word typed. Basically, it always returns (CMD . REST-OF-INPUT), where REST-OF-INPUT is whatever the other options dictate for the remainder. E.g. COMMAND NOVALUE returns (CMD) or (CMD . T), depending on whether there was further input; COMMAND STRING returns (CMD . "REST-OF-INPUT"). When used with REPEAT, COMMAND is only in effect for the first line typed; furthermore, if the first line consists solely of a command, the REPEAT is ignored, i.e., the entire input is taken to be just the command.

READ  Parentheses, brackets, and quotes are treated a la READ, rather than being returned as individual atoms. Control characters may be input via the control-Vx notation. Input is terminated roughly along the lines of READ conventions: a balancing or over-balancing right parenthesis will activate the input, or <cr> when no parenthesis remains unbalanced. READ overrides all other options (except NORAISE).

LISPXREAD  Like READ, but implies that TTYIN should behave even more like READ, i.e., do NORAISE, not be errorset-protected, etc.
TTYIN DISPLAY TYPEIN EDITOR

NOPROMPT

Interlisp-D only: The prompt argument is treated as usual, except that TTYIN assumes that the prompt for the first line has already been printed by the caller; the prompt for the first line is thus used only when redisplaying the line.

ECHOTOFILE if specified, user's input is copied to this file, i.e., TTYIN can be used as a simple text-to-file routine if NOVALUE is used. If ECHOTOFILE is a list, copies to all files in the list.

PROMPT is not included on the file.

TABS is a special addition for tabular input. It is a list of tabstops (numbers). When user types a tab, TTYIN automatically spaces over to the next tabstop (thus the first tabstop is actually the second "column" of input). Also treats specially the characters * and "; they echo normally, and then automatically tab over.

UNREADBUF allows the caller to "preload" the TTYIN buffer with a line of input. UNREADBUF is a list, the elements of which are unread into the buffer (i.e., "the outer parentheses are stripped off") to be edited further as desired; a simple carriage return (or control-Z for REPEAT input) will thus cause the buffer's contents to be returned unchanged. If doing READ input, the "PRIN2 names" of the input list are used, i.e., quotes and %'s will appear as needed; otherwise the buffer will look as though UNREADBUF had been PRIN1'ed. UNREADBUF is treated somewhat like READBUF, so that if it contains a pseudo-carriage return (the value of HISTSTRO), the input line terminates there.

Input can also be unread from a file, using the HISTSTR1 format: UNREADBUF = (<value of HISTSTR1> (FILE START . END)), where START and END are file byte pointers. This makes TTYIN a miniature text file editor.

RDTBL [Interlisp-D only] is the read table to use for READing the input when one of the READ options is given. A lot of character interpretations are hardwired into TTYIN, so currently the only effect this has is in the actual READ, and in deciding whether a character typed at the end of the input is an immediate read macro, for purposes of termination.

If the global variable TYPEAHEADFLG is T, or option LISPXREAD is given, TTYIN permits type-ahead; otherwise it clears the buffer before prompting the user.

26.4.7 Using TTYIN as a General Editor

The following may be useful as a way of outsiders to call TTYIN as an editor. These functions are currently only in Interlisp-D.

(TTYINEDIT EXPRS WINDOW PRINTFN PROMPT) [Function]

This is the body of the edit macro EE. Switches the tty to WINDOW, clears it, prettyprints EXPRS, a list of expressions, into
it, and leaves you in TTYIN to edit it as Lisp input. Returns a new list of expressions.

If PRINTFN is non-NIL, it is a function of two arguments, EXPRS and FILE, which is called instead of PRETTYPRINT to print the expressions to the window (actually to a scratch file). Note that EXPRS is a list, so normally the outer parentheses should not be printed. PRINTFN=T is shorthand for "unpretty"; use PRIN2 instead of PRETTYPRINT.

PROMPT determines what prompt is printed, if any. If T, no prompt is printed. If NIL, it defaults to the value of TTYINEDITPROMPT.

TTYINAUTOCLOSEFLG [Variable]
If TTYINAUTOCLOSEFLG is true, TTYINEDIT closes the window on exit.

TTYINEDITWINDOW [Variable]
If the WINDOW arg to TTYINEDIT is NIL, it uses the value of TTYINEDITWINDOW, creating it if it does not yet exist.

TTYINPRINTFN [Variable]
The default value for PRINTFN in EE's call to TTYINEDIT.

(SET.TTYINEDIT.WINDOW WINDOW) [Function]
Called under a RESETLIST. Switches the tty to WINDOW (defaulted as in TTYINEDIT) and clears it. The window's position is left so that TTYIN will be happy with it if you now call TTYIN yourself. Specifically, this means positioning an integral number of lines from the bottom of the window, the way the top-level tty window normally is.

(TTYIN.SCRATCHFILE) [Function]
Returns, possibly creating, the scratchfile that TTYIN uses for prettyprinting its input. The file pointer is set to zero. Since TTYIN does use this file, beware of multiple simultaneous use of the file.

26.4.8 [=] Handler
In Interlisp, the [=] read macro displays the arguments to the function currently "in progress" in the typein. Since TTYIN wants you to be able to continue editing the buffer after a [=], it processes this macro specially on its own, printing the arguments below your typein and then putting the cursor back where it was
when \( ? = \) was typed. For users who want special treatment of \( ? = \), the following hook exists:

<table>
<thead>
<tr>
<th>TTYIN? = FN</th>
<th>[Variable]</th>
</tr>
</thead>
<tbody>
<tr>
<td>The value of this variable, if non-NIL, is a user function of one argument that is called when ( ? = ) is typed. The argument is the function that ( ? = ) thinks it is inside of. The user function should return one of the following:</td>
<td></td>
</tr>
<tr>
<td>NIL</td>
<td>Normal ( ? = ) processing is performed.</td>
</tr>
<tr>
<td>T</td>
<td>Nothing is done. Presumably the user function has done something privately, perhaps diddled some other window, or called TTYIN.PRINTARGS (below).</td>
</tr>
<tr>
<td>a list (ARGS . STUFF)</td>
<td>Treats STUFF as the argument list of the function in question, and performs the normal ( ? = ) processing using it.</td>
</tr>
<tr>
<td>anything else</td>
<td>The value is printed in lieu of what ( ? = ) normally prints.</td>
</tr>
</tbody>
</table>

At the time that \( ? = \) is typed, nothing has been "read" yet, so you don't have the normal context you might expect inside a conventional readmacro. If the user function wants to examine the typed-in arguments being passed to the fn, however, it can call the function TTYIN.READ? = ARGS:

<table>
<thead>
<tr>
<th>(TTYIN.READ? = ARGS)</th>
<th>[Function]</th>
</tr>
</thead>
<tbody>
<tr>
<td>When called inside TTYIN? = FN user function, returns everything between the function and the typing of ( ? = ) as a list (like an arglist). Returns NIL if ( ? = ) was typed immediately after the function name.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(TTYIN.PRINTARGS FN ARGS ACTUALS ARGTYPE)</th>
<th>[Function]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the function/argument printing for ( ? = ). ARGS is an argument list, ACTUALS is a list of actual parameters (from the typein) to match up with args. ARGTYPE is a value of the function ARGTYPE; it defaults to (ARGTYPE FN).</td>
<td></td>
</tr>
</tbody>
</table>

### 26.4.9 Read Macros

When doing READ input in Interlisp-10, no Lisp-style read macros are available (but the ' and control-Y macros are built in). Principally because of the usefulness of the editor read macros (set by SETTERMCHARS), and the desire for a way of changing the meanings of the display editing commands, the following exists as a hack:
<table>
<thead>
<tr>
<th>TTYINREADMACROS</th>
<th>[Variable]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value is a set of shorthand inputs useable during READ input. It is an alist of entries (CHARCODE . SYNONYM). If the user types the indicated character (the meta bit is denoted by the 200Q bit in the char code), TTYIN behaves as though the synonym character had been typed.</td>
<td></td>
</tr>
</tbody>
</table>

Special cases: 0 - the character is ignored; 200Q - pure meta bit; means to read another char and turn on its meta bit; 400Q - macro quote: read another char and use its original meaning.

For example, if you have macros ((33Q . 200Q) (30Q . 33Q)), then Escape (33Q) will behave as an edit prefix, and control-X (30Q) will behave like Escape. Note: currently, synonyms for meta commands are not well-supported, working only when the command is typed with no argument.

Slightly more powerful macros also can be supplied; they are recognized when a character is typed on an empty line, i.e., as the first thing after the prompt. In this case, the TTYINREADMACROS entry is of the form (CHARCODE T . RESPONSE), or (CHARCODE CONDITION . RESPONSE), where CONDITION is a list that evaluates true. If RESPONSE is a list, it is EVALed; otherwise it is left unevaluated. The result of this evaluation (or RESPONSE itself) is treated as follows:

- **NIL** - The macro is ignored and the character reads normally, i.e., as though TTYINREADMACROS had never existed.
- **An integer** - A character code, treated as above. Special case: -1 is treated like 0, but says that the display may have been altered in the evaluation of the macro, so TTYIN should reset itself appropriately.
- **Anything else** - This TTYIN input is terminated (with a crlf) and returns the value of "response" (turned into a list if necessary). This is the principal use of this facility. The macro character thus stands for the (possibly computed) reponse, terminated if necessary with a crlf.

The original character is not echoed.

Interrupt characters, of course, cannot be read macros, as TTYIN never sees them, but any other characters, even non-control chars, are allowed. The ability to return NIL allows you to have conditional macros that only apply in specified situations (e.g., the macro might check the prompt (LISPXD) or other contextual variables). To use this specifically to do immediate editor read macros, do the following for each edit command and character you want to invoke it with:

**Example**

```
(ADDTOVAR TTYINREADMACROS (CHARCODE 'CHARMACRO? EDITCOM))
```

For example, (ADDTOVAR TTYINREADMACROS (12Q CHARMACRO? !NX)) will make linefeed do the !NX command.
TTYIN DISPLAY TYPEIN EDITOR

Note that this will only activate linefeed at the beginning of a line, not anywhere in the line. There will probably be a user function to do this in the next release.

Note that putting \(12Q~T .!NX\) on TTYINREADMACROS would also have the effect of returning "!NX" from the READ call so that the editor would do an INX. However, TTYIN would also return !NX outside the editor (probably resulting in a u.b.a. error, or convincing DWIM to enter the editor), and also the clearing of the output buffer (performed by CHARMACRO?) would not happen.

26.4.10 Assorted Flags

These flags control aspects of TTYIN's behavior. Some have already been mentioned. In Interlisp-D, the flags are all initially set to T.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPEAHEADFLG</td>
<td>[Variable] If true, TTYIN always permits typeahead; otherwise it clears the buffer for any but LISPREAD input.</td>
</tr>
<tr>
<td>?ACTIVATEFLG</td>
<td>[Variable] If true, enables the feature whereby ? lists alternative completions from the current spelling list.</td>
</tr>
<tr>
<td>SHOWPARENFLG</td>
<td>[Variable] If true, then whenever you are typing Lisp input and type a right parenthesis/bracket, TTYIN will briefly move the cursor to the matching parenthesis/bracket, assuming it is still on the screen. The cursor stays there for about 1 second, or until you type another character (i.e., if you type fast you'll never notice it). This feature was inspired by a similar EMACS feature, and turned out to be pretty easy to implement.</td>
</tr>
<tr>
<td>TTYINBSFLG</td>
<td>[Variable] Causes TTYIN to always physically backspace, even if you're running on a non-display (not a DM or Heath), rather than print 'deletedtext' (this assumes your hardcopy terminal or glass tty is capable of backspacing). If TTYINBSFLG is LF, then in addition to backspacing, TTYIN x's out the deleted characters as it backs up, and when you stop deleting, it outputs a linefeed to drop to a new, clean line before resuming. To save paper, this linefeed operation is not done when only a single character is deleted, on the grounds that you can probably figure out what you typed anyway.</td>
</tr>
</tbody>
</table>
TTYINRESPONSES  [Variable]
An association list of special responses that will be handled by
routines designated by the programmer. See "Special
Responses", below.

TTYINERRORSETFLG  [Variable]
[Interlisp-D only] If true, non-LISPXREAD inputs are
errorset-protected (control-E traps back to the prompt),
otherwise errors propagate upwards. Initially NIL.

TTYINCOMMENTCHAR  [Variable]
This variable affects the treatment of lines beginning with the
comment character (usually ";"). If TTYINCOMMENTCHAR is a
character code, and the first character on a line of typein is equal
to TTYINCOMMENTCHAR, then the line is erased from the screen
and no input function will see it. If TTYINCOMMENTCHAR is NIL,
this feature is disabled. TTYINCOMMENTCHAR is initially NIL.

TTYINCOMPLETEFLG  [Variable]
If true, enables Escape completion from USERWORDS during
READ inputs. Details below.

USERWORDS (page 20.17) contains words you mentioned
recently: functions you have defined or edited, variables you
have set or evaluated at the executive level, etc. This happens to
be a very convenient list for context-free escape completion; if
you have recently edited a function, chances are good you may
want to edit it again (typing "EF xxS") or type a call to it. If there
is no completion for the current word from USERWORDS, the
escape echoes as "$", i.e. nothing special happens; if there is
more than one possible completion, you get beeped. If typed
when not inside a word, Escape completes to the value of
LASTWORD, i.e., the last thing you typed that the p.a. “noticed”
(setting TTYINCOMPLETEFLG to 0 disables this latter feature),
except that Escape at the beginning of the line is left alone (it is a
p.a. command).

If you really wanted to enter an escape, you can, of course, just
quote it with a control-V, like you can other control chars.

You may explicitly add words to USERWORDS yourself that
wouldn't get there otherwise. To make this convenient online
the edit command [←] means "add the current atom to
USERWORDS" (you might think of the command as "pointing
out this atom"). For example, you might be entering a function
definition and want to "point to" one or more of its arguments
or prog variables. Giving an argument of zero to this command
will instead remove the indicated atom from USERWORDS.
Note that this feature loses some of its value if the spelling list is too long, for then the completion takes too long computationally and, more important, there are too many alternative completions for you to get by with typing a few characters followed by escape. Lisp's maintenance of the spelling list USERWORDS keeps the "temporary" section (which is where everything goes initially unless you say otherwise) limited to #USERWORDS atoms, initially 100. Words fall off the end if they haven't been used (they are "used" if FIXSPELL corrects to one, or you use <escape> to complete one).

26.4.11 Special Responses

There is a facility for handling "special responses" during any non-READ TTYIN input. This action is independent of the particular call to TTYIN, and exists to allow you to effectively "advise" TTYIN to intercept certain commands. After the command is processed, control returns to the original TTYIN call. The facility is implemented via the list TTYINRESPONSES.

TTYINRESPONSES is a list of elements, each of the form:

(COMMANDS RESPONSE-FORM OPT/ON)

COMMANDS is a single atom or list of commands to be recognized; RESPONSE-FORM is EVALed (if a list), or APPLYed (if an atom) to the command and the rest of the line. Within this form one can reference the free variables COMMAND (the command the user typed) and LINE (the rest of the line). If OPTION is the atom LINE, this means to pass the rest of line as a list; if it is STRING, this means to pass it as a string; otherwise, the command is only valid if there is nothing else on the line. If RESPONSE-FORM returns the atom IGNORE, it is not treated as a special response (i.e. the input is returned normally as the result of TTYIN).

Suggested use: global commands or options can be added to the toplevel value of TTYINRESPONSES. For more specialized commands, rebind TTYINRESPONSES to (APPEND NEWENTRIES TTYINRESPONSES) inside any module where you want to do this sort of special processing.

Special responses are not checked for during READ-style input.

26.4.12 Display Types

[This is not relevant in Interlisp-D]
TTYIN determines the type of display by calling DISPLAYTERMP, which is initially defined to test the value of the GTTYP jsys. It returns either NIL (for printing terminals) or a small number giving TTYIN's internal code for the terminal type. The types TTYIN currently knows about:

0 = glass tty (capable of deleting chars by backspacing, but little else);
1 = Datamedia;
2 = Heath.

Only the Datamedia has full editing power. DISPLAYTERMP has built into it the correct terminal types for Sumex and Stanford campus 20's: Datamedia = 11 on tenex, 5 on tops20; Heath = 18 on Tenex, 25 on tops20. You can override those values by setting the variable DISPLAYTYPES to be an association list associating the GTTYP value with one of these internal codes. For example, Sumex displays correspond to DISPLAYTYPES = ((11 . 1) (18 . 2)) [although this is actually compiled into DISPLAYTERMP for speed]. Any display terminal other than Datamedia and Heath can probably safely be assigned to "0" for glass tty.

To add new terminal types, you have to choose a number for it, add new code to TTYIN for it and recompile. The TTYIN code specifies what the capabilities of the terminal are, and how to do the primitive operations: up, down, left, right, address cursor, erase screen, erase to end of line, insert character, etc.

For terminals lacking a meta key (currently only Datamedias have it), set the variable EDITPREFIXCHAR to the ascii code of an edit "prefix" (i.e. anything typed preceded by the prefix is considered to have the meta bit on). If your EDITPREFIXCHAR is 33Q (Escape), you can type a real Escape by typing 3 of them (2 won't do, since that means "Meta-Escape", a legitimate argument to another command). You could also define an Escape synonym with TTYINREADMACROS if you wanted (but currently it doesn't work in filename completion). Setting EDITPREFIXCHAR for a terminal that is not equipped to handle the full range of editing functions (only the Heath and Datamedia are currently so equipped) is not guaranteed to work, i.e. the display will not always be up to date; but if you can keep track of what you're doing, together with an occasional control-R to help out, go right ahead.

26.5 Prettyprint

The standard way of printing out function definitions (on the terminal or into files) is to use PRETTYPRINT.
(PRETTYPRINT FNS PRETTYDEFLG —)

[Function]

FNS is a list of functions. If FNS is atomic, its value is used. The definitions of the functions are printed in a pretty format on the primary output file using the primary readtable. For example, if FACTORIAL were defined by typing

(DEFINEQ (FACTORIAL [LAMBDA (N) (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))))])

(PRETTYPRINT '(FACTORIAL)) would print out

(FACTORIAL
 [LAMBDA (N)
  (COND
   ((ZEROP N) 1)
   (T (TIMES N (FACTORIAL (SUB1 N)))))

PRETTYDEFLG is T when called from PRETTYDEF (and hence MAKEFILE). Among other actions taken when this argument is true, PRETTYPRINT indicates its progress in writing the current output file: whenever it starts a new function, it prints on the terminal the name of that function if more than 30 seconds (real time) have elapsed since the last time it printed the name of a function.

PRETTYPRINT operates correctly on functions that are BROKEN, BROKEN-IN, ADVISED, or have been compiled with their definitions saved on their property lists: it prints the original, pristine definition, but does not change the current state of the function. If a function is not defined but is known to be on one of the files noticed by the file package, PRETTYPRINT loads in the definition (using LOADFNS) and prints it (except when called from PRETTYDEF). If PRETTYPRINT is given an atom which is not the name of a function, but has a value, it prettyprints the value. Otherwise, PRETTYPRINT attempts spelling correction. If all fails, PRETTYPRINT returns (FN NOT PRINTABLE). Note that PRETTYPRINT will return (FN NOT PRINTABLE) if FN does not have an accessible expr definition, or if it doesn’t have any definition at all.

(P P F N 1 ... F N N)

[NLambda NoSpread Function]

For prettyprinting functions to the terminal. PP calls PRETTYPRINT with the primary output file set to T and the primary readtable set to T. The primary output file and primary readtable are restored after printing.

(PP FOO) is equivalent to (PRETTYPRINT '(FOO)); (PP FOO FIE) is equivalent to (PRETTYPRINT '(FOO FIE)).

As described above, when PRETTYPRINT, and hence PP, is called with the name of a function that is not defined, but whose
definition is on a file known to the file package, the definition is automatically read in and then prettyprinted. However, if the user does not intend on editing or running the definition, but simply wants to see the definition, the function PF described below can be used to simply copy the corresponding characters from the file to the terminal. This results in a savings in both space and time, since it is not necessary to allocate storage to actually read in the definition, and it is not necessary to re-prettyprint it (since the function is already in prettyprint format on the file).

(PF FN FROMFILES TOFILE)           [NLambda NoSpread Function]
Copies the definition of FN found on each of the files in FROMFILES to TOFILE. If TOFILE = NIL, defaults to T. If FROMFILES = NIL, defaults to (WHEREIS FN NIL T) (see page 17.14). The typical usage of PF is simply to type "PF FN".
PF prints a message if it can't find a file on FROMFILES, or it can't find the function FN on a file.

When printing to the terminal, PF performs several transformations on the characters in the file that comprise the definition for FN: (1) font information is stripped out (except in Interlisp-D, whose display supports multiple fonts); (2) occurrences of the CHANGECHAR (page 26.49) are not printed; (3) since functions typically tend to be printed to a file with a larger linelength than when printing to a terminal, the number of leading spaces on each line is cut in half (unless PFDEFAULT is T; initially NIL); and (4) comments are elided, if **COMMENT**FLG is non-NIL (see page 26.43).

(SEE FROMFILE TOFILE)               [NLambda NoSpread Function]
Copies all of the text from FROMFILE to TOFILE (defaults to T), processing all text as PF does. Used to display the contents of files on the terminal.

(PP* X)                              [NLambda NoSpread Function]

(PF* FN FROMFILES TOFILE)            [NLambda NoSpread Function]

(SEE* FROMFILE TOFILE)               [NLambda NoSpread Function]
These functions operate exactly like PP, PF, and SEE, except that they bind **COMMENT**FLG to NIL, so comments are printed in full (see page 26.43).
While the function `PRETTYPRINT` prints entire function definitions, the function `PRINTDEF` can be used to print parts of functions, or arbitrary Interlisp structures:

```
(PRINTDEF EXPR LEFT DEF TAILFLG FNSLST FILE) [Function]
```

Prints the expression `EXPR` in a pretty format on `FILE` using the primary readable. `LEFT` is the left hand margin (LINELENGTH determines the right hand margin). `PRINTDEF` initially performs `(TAB LEFT T)`, which means to space to position `LEFT`, unless already beyond this position, in which case it does nothing.

`DEF = T` means `EXPR` is a function definition, or a piece of one. If `DEF = NIL`, no special action is taken for LAMBDA's, PROG's, COND's, comments, CLISP, etc. `DEF` is NIL when `PRETTYDEF` calls `PRETTYPRINT` to print variables and property lists, and when `PRINTDEF` is called from the editor via the command PPV.

`TAILFLG = T` means `EXPR` is interpreted as a tail of a list, to be printed without parentheses.

`FNSLST` is for use for printing with multiple fonts (page 27.25). `PRINTDEF` prints occurrences of any function in the list `FNSLST` in a different font, for emphasis. `MAKEFILE` passes as `FNSLST` the list of all functions on the file being made.

### 26.5.1 Comment Feature

A facility for annotating Interlisp functions is provided in `PRETTYPRINT`. Any expression beginning with the atom `*` is interpreted as a comment and printed in the right margin. Example:

```
(FACTORIAL
 [LAMBDA (N) (* COMPUTES N!)]
 (COND
  ([ZEROP N] (* 0! = 1))
  1)
 (T (* RECURSIVE DEFINITION:
      N! = N!*N-1!))
)
```

These comments actually form a part of the function definition. Accordingly, `*` is defined as an `nlambda` nospread function that returns its argument, similar to `QUOTE`. When running an interpreted function, `*` is entered the same as any other Interlisp function. Therefore, comments should only be placed where they will not harm the computation, i.e., where a quoted expression could be placed. For example, writing

```
(ITIMES N (FACTORIAL (SUB1 N)) (* RECURSIVE DEFINITION))
```
in the above function would cause an error when ITIMES attempted to multiply N, N-1!, and RECURSIVE.

For compilation purposes, * is defined as a macro which compiles into no instructions (unless the comment has been placed where it has been used for value, in which case the compiler prints an appropriate error message and compiles * as QUOTE). Thus, the compiled form of a function with comments does not use the extra atom and list structure storage required by the comments in the source (interpreted) code. This is the way the comment feature is intended to be used.

A comment of the form (* E X) causes X to be evaluated at prettyprint time, as well as printed as a comment in the usual way. For example, (* E (RADIX 8)) as a comment in a function containing octal numbers can be used to change the radix to produce more readable printout.

The comment character * is stored in the variable COMMENTFLG. The user can set it to some other value, e.g. ",", and use this to indicate comments.

**COMMENT**FLG

If CAR of an expression is EQ to COMMENTFLG, the expression is treated as a comment by PRETTYPRINT. COMMENTFLG is initialized to *. Note that whatever atom is chosen for COMMENTFLG should also have an appropriate function definition and compiler macro, for example, by copying those of *.

Comments are designed mainly for documenting listings. Therefore, when prettyprinting to the terminal, comments are suppressed and printed as the string **COMMENT**. The value of **COMMENT**FLG determines the action.

**COMMENT**FLG

If **COMMENT**FLG is NIL, comments are printed. Otherwise, the value of **COMMENT**FLG is printed. Initially "**COMMENT**".

(COMMENT1 L —)

Prints the comment L. COMMENT1 is a separate function to permit the user to write prettyprint macros (page 26.48) that use the regular comment printer. For example, to cause comments to be printed at a larger than normal linelength, one could put an entry for * on PRETTYPRINTMACROS:

(* LAMBDA (X) (RESETFORM (LINELENGTH 100) (COMMENT1 X)))

This macro resets the line length, prints the comment, and then restores the line length.
COMMENT1 expects to be called from within the environment established by PRINTDEF, so ordinarily the user should call it only from within prettyprint macros.

26.5.2 Comment Pointers

For a well-commented collection of programs, the list structure, atom, and print name storage required to represent the comments in core can be significant. If the comments already appear on a file and are not needed for editing, a significant savings in storage can be achieved by simply leaving the text of the comment on the file when the file is loaded, and instead retaining in core only a pointer to the comment. When this feature is enabled, * is defined as a read macro (page 25.39) in FILERDTBL which, instead of reading in the entire text of the comment, constructs an expression containing (1) the name of the file in which the text of the comment is contained, (2) the address of the first character of the comment, (3) the number of characters in the comment, and (4) a flag indicating whether the comment appeared at the right hand margin or centered on the page. For output purposes, * is defined on PRETTYPRINTMACROS (page 26.48) so that it prints the comments represented by such pointers by simply copying the corresponding characters from one file to another, or to the terminal. Normal comments are processed the same as before, and can be intermixed freely with comment pointers.

The comment pointer feature is controlled by the function NORMALCOMMENTS.

(NORMALCOMMENTS FLG) [Function]
If FLG is NIL, the comment pointer feature is enabled. If FLG is T, the comment pointer feature is disabled (the default).

NORMALCOMMENTS can be changed as often as desired. Thus, some files can be loaded normally, and others with their comments converted to comment pointers.

For convenience of editing selected comments, an edit macro, GET*, is included, which loads in the text of the corresponding comment. The editor's PP* command, in contrast, prints the comment without reading it by simply copying the corresponding characters to the terminal. GET* is defined in terms of GETCOMMENT:

(GETCOMMENT X DESTFL —) [Function]
If X is a comment pointer, replaces X with the actual text of the comment, which it reads from its file. Returns X in all cases.
DESTFL is non-NIL, it is the name of an open file, to which GETCOMMENT copies the comment; in this case, X remains a comment pointer, but it has been changed to point to the new file (unless NORMALCOMMENTS has been set to DONTUPDATE).

(PRINTCOMMENT X)  [Function]
Defined as the prettyprint macro for *: copies the comment to the primary output file by using GETCOMMENT.

(READCOMMENT FL RDTBL LST)  [Function]
Defined as the read macro for * in FILERDTBL: if NORMALCOMMENTSFLG is NIL, it constructs a comment pointer, unless it believes the expression beginning with * is not actually a comment, e.g., if the next atom is "." or E.

Note that a certain amount of care is required in using the comment pointer feature. Since the text of the comment resides on the file pointed to by the comment pointer, that file must remain in existence as long as the comment is needed. GETCOMMENT helps out by changing the comment pointer to always point at the most recent file that the comment lives on. However, if the user has been performing repeated MAKEFILE's (page 17.10) in which differing functions have changed at each invocation of MAKEFILE, it is possible for the comment pointers in memory to be pointing at several versions of the same file, since a comment pointer is only updated when the function it lives in is prettyprinted, not when the function has been copied verbatim to the new file. This can be a problem for file systems that have a built-in limit on the number of versions of a given file that will be made before old versions are expunged. In such a case, the user should set the version retention count of any directories involved to be infinite. GETCOMMENT prints an error message if the file that the comment pointer points at has disappeared.

Similarly, one should be cognizant of comment pointers in sysouts, and be sure to retain any files thus pointed to.

When using comment pointers, the user should also not set PRETTYFLG (page 26.48) to NIL or call MAKEFILE with option FAST, since this will prevent functions from being prettyprinted, and hence not get the text of the comment copied into the new file.

If the user changes the value of COMMENTFLG but still wishes to use the comment pointer feature, the new COMMENTFLG should be given the same read-macro definition in FILERDTBL as * has, and the same entry be put on PRETTYPRINTMACROS. For example, if COMMENTFLG is reset to be ";", then (SETSYNTAX ";");
** FILERDTBL) should be performed, and (; PRINTCOMMENT) added to PRETTYPRINTMACROS.

### 26.5.3 Converting Comments to Lower Case

This section is for users using terminals without lower case, who nevertheless would like their comments to be converted to lower case for more readable listings. If the second atom in a comment is %%, the text of the comment is converted to lower case so that it looks like English instead of Lisp. Note that comments are converted only when they are actually written to a file by PRETTYPRINT.

The algorithm for conversion to lower case is the following: If the first character in an atom is ↑, do not change the atom (but remove the ↑). If the first character is %, convert the atom to lower case. Note that the user must type %% as % is the escape character. If the atom (minus any trailing punctuation marks) is an Interlisp word (i.e., is a bound or free variable for the function containing the comment, or has a top level value, or is a defined function, or has a non-NIL property list), do not change it. Otherwise, convert the atom to lower case. Conversion only affects the upper case alphabet, i.e., atoms already converted to lower case are not changed if the comment is converted again. When converting, the first character in the comment and the first character following each period are left capitalized. After conversion, the comment is physically modified to be the lower case text minus the %% flag, so that conversion is thus only performed once (unless the user edits the comment inserting additional upper case text and another %% flag).

<table>
<thead>
<tr>
<th>Variable</th>
<th>LCASELST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words on LCASELST will always be converted to lower case. LCASELST is initialized to contain words which are Interlisp functions but also appear frequently in comments as English words (AND, EVERY, GET, GO, LAST, LENGTH, LIST, etc.). Therefore, if one wished to type a comment including the lisp function GO, it would be necessary to type ↑ GO in order that it might be left in upper case.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>UCASELST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words on UCASELST (that do not appear on LCASELST) will be left in upper case. UCASELST is initialized to NIL.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>ABBREVLST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABBREVLST is used to distinguish between abbreviations and words that end in periods. Normally, words that end in periods and occur more than halfway to the right margin cause</td>
<td></td>
</tr>
</tbody>
</table>
26.5.4 Special Prettyprint Controls

**PRETTYTABFLG**

In order to save space on files, tabs are used instead of spaces for the initial spaces on each line, assuming that each tab corresponds to 8 spaces. This results in a reduction of file size by about 30%. Tabs are not used if PRETTYTABFLG is set to NIL (initially T).

**#RPARS**

Controls the number of right parentheses necessary for square bracketing to occur. If #RPARS = NIL, no brackets are used. #RPARS is initialized to 4.

**FIRSTCOL**

The starting column for comments. Comments run between FIRSTCOL and the line length set by LINELENGTH (page 25.11). If a word in a comment ends with a "." and is not on the list ABBREVLIST, and the position is greater than halfway between FIRSTCOL and LINELENGTH, the next word in the comment begins on a new line. Also, if a list is encountered in a comment, and the position is greater than halfway, the list begins on a new line.

**PRETTYLCOM**

If a comment has more than PRETTYLCOM elements (using COUNT), it is printed starting at column 10, instead of FIRSTCOL. Comments are also printed starting at column 10 if their second element is also a *, i.e., comments of the form (* * --).

**#CAREFULCOLUMNS**

In the interests of efficiency, PRETTYPRINT approximates the number of characters in each atom, rather than calling NCHARS, when computing how much will fit on a line. This procedure works satisfactorily in most cases. However, users with unusually long atoms in their programs, e.g., such as produced by CLISPIFY, may occasionally encounter some glitches in the output produced by PRETTYPRINT. The value of #CAREFULCOLUMNS tells PRETTYPRINT how many columns (counting from the right hand...
margin) in which to actually compute NCHARS instead of approximating. Setting #CAREFULCOLUMNS to 20 or 30 will eliminate the glitches, although it will slow down PRETTYPRINT slightly. #CAREFULCOLUMNS is initially 0.

(WIDEPAPER FLG) [Function]

(WIDEPAPER T) sets FILELINELENGTH (page 25.11), FIRSTCOL, and PRETYLCOM to large values appropriate for pretty printing files to be listed on wide paper. (WIDEPAPER) restores these parameters to their initial values. WIDEPAPER returns the previous setting of FLG.

PRETTYFLG [Variable]

If PRETTYFLG is NIL, PRINTDEF uses PRIN2 instead of prettyprinting. This is useful for producing a fast symbolic dump (see the FAST option of MAKEFILE, page 17.10). Note that the file loads the same as if it were prettyprinted. PRETTYFLG is initially set to T. PRETTYFLG should not be set to NIL if comment pointers (page 26.44) are being used.

CLISPIFYPRETTYFLG [Variable]

Used to inform PRETTYPRINT to call CLISPIFY on selected function definitions before printing them (see page 21.26).

PRETTYPRINTMACROS [Variable]

An association-list that enables the user to control the formatting of selected expressions. CAR of each expression being PRETTYPRINTed is looked up on PRETTYPRINTMACROS, and if found, CDR of the corresponding entry is applied to the expression. If the result of this application is NIL, PRETTYPRINT ignores the expression; i.e., it prints nothing, assuming that the prettyprintmacro has done any desired printing. If the result of applying the prettyprint macro is non-NIL, the result is prettyprinted in the normal fashion. This gives the user the option of computing some other expression to be prettyprinted in its place.

Note: "prettyprinted in the normal fashion" includes processing prettyprint macros, unless the prettyprint macro returns a structure EQ to the one it was handed, in which case the potential recursion is broken.

PRETTYPRINTYPEMACROS [Variable]

A list of elements of the form (TYPENAME . FN). For types other than lists and atoms, the type name of each datum to be prettyprinted is looked up on PRETTYPRINTYPEMACROS, and if
found, the corresponding function is applied to the datum about
to be printed, instead of simply printing it with PRIN2.

**PRETTYEQUIVLST**  [Variable]
An association-list that tells PRETTYPRINT to treat a CAR-of-form
the same as some other CAR-of-form. For example, if
(QLAMBDA . LAMBDA) appears on PRETTYEQUIVLST, then
expressions beginning with QLAMBDA are prettyprinted the
same as LAMBDA's. Currently, PRETTYEQUIVLST only allows (i.e.,
supports in an interesting way) equivalences to forms that
PRETTYPRINT internally handles. Equivalence to forms for which
the user has specified a prettyprint macro should be made by
adding further entries to PRETTYPRINTMACROS.

**CHANGECHAR**  [Variable]
If non-NIL, and PRETTYPRINT is printing to a file or display
terminal, PRETTYPRINT prints CHANGECHAR in the right hand
margin while printing those expressions marked by the editor as
having been changed (see page 16.30). CHANGECHAR is initially
[This page intentionally left blank]
# 27. Graphics Output Operations

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.1. Primitive Graphics Concepts</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1.1. Positions</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1.2. Regions</td>
<td>27.1</td>
</tr>
<tr>
<td>27.1.3. Bitmaps</td>
<td>27.3</td>
</tr>
<tr>
<td>27.1.4. Textures</td>
<td>27.6</td>
</tr>
<tr>
<td>27.2. Opening Image Streams</td>
<td>27.8</td>
</tr>
<tr>
<td>27.3. Accessing Image Stream Fields</td>
<td>27.10</td>
</tr>
<tr>
<td>27.5. Moving Bits Between Bitmaps With BITBLT</td>
<td>27.14</td>
</tr>
<tr>
<td>27.6. Drawing Lines</td>
<td>27.17</td>
</tr>
<tr>
<td>27.7. Drawing Curves</td>
<td>27.18</td>
</tr>
<tr>
<td>27.8. Miscellaneous Drawing and Printing Operations</td>
<td>27.20</td>
</tr>
<tr>
<td>27.9. Drawing and Shading Grids</td>
<td>27.22</td>
</tr>
<tr>
<td>27.10. Display Streams</td>
<td>27.23</td>
</tr>
<tr>
<td>27.12. Fonts</td>
<td>27.25</td>
</tr>
<tr>
<td>27.13. Font Files and Font Directories</td>
<td>27.31</td>
</tr>
<tr>
<td>27.15. Font Profiles</td>
<td>27.32</td>
</tr>
<tr>
<td>27.16. Image Objects</td>
<td>27.35</td>
</tr>
<tr>
<td>27.16.1. IMAGEFNS Methods</td>
<td>27.36</td>
</tr>
<tr>
<td>27.16.2. Registering Image Objects</td>
<td>27.39</td>
</tr>
<tr>
<td>27.16.3. Reading and Writing Image Objects on Files</td>
<td>27.40</td>
</tr>
<tr>
<td>27.16.4. Copying Image Objects Between Windows</td>
<td>27.41</td>
</tr>
<tr>
<td>27.17. Implementation of Image Streams</td>
<td>27.42</td>
</tr>
</tbody>
</table>
[This page intentionally left blank]
Streams are used as the basis for all I/O operations. Files are implemented as streams that can support character printing and reading operations, and file pointer manipulation. An image stream is a type of stream that also provides an interface for graphical operations. All of the operations that can applied to streams can be applied to image streams. For example, an image stream can be passed as the argument to PRINT, to print something on an image stream. In addition, special functions are provided to draw lines and curves and perform other graphical operations. Calling these functions on a stream that is not an image stream will generate an error.

27.1 Primitive Graphics Concepts

The Interlisp-D graphics system is based on manipulating bitmaps (rectangular arrays of pixels), positions, regions, and textures. These objects are used by all of the graphics functions.

27.1.1 Positions

A position denotes a point in an X,Y coordinate system. A POSITION is an instance of a record with fields XCOORD and YCOORD and is manipulated with the standard record package facilities. For example, (create POSITION XCOORD ← 10 YCOORD ← 20) creates a position representing the point (10,20).

**FUNCTION**

Returns X if X is a position; NIL otherwise.

27.1.2 Regions

A Region denotes a rectangular area in a coordinate system. Regions are characterized by the coordinates of their bottom left corner and their width and height. A REGION is a record with fields LEFT, BOTTOM, WIDTH, and HEIGHT. It can be manipulated with the standard record package facilities. There
are access functions for the REGION record that return the TOP and RIGHT of the region.

The following functions are provided for manipulating regions:

**(CREATEREGION LEFT BOTTOM WIDTH HEIGHT)**

Returns an instance of the REGION record which has LEFT, BOTTOM, WIDTH and HEIGHT as respectively its LEFT, BOTTOM, WIDTH, and HEIGHT fields.

Example: *(CREATEREGION 10 -20 100 200)* will create a region that denotes a rectangle whose width is 100, whose height is 200, and whose lower left corner is at the position (10,-20).

**(REGIONP X)**

Returns X if X is a region, NIL otherwise.

**(INTERSECTREGIONS REGION1 REGION2 ... REGIONn)**

Returns a region which is the intersection of a number of regions. Returns NIL if the intersection is empty.

**(UNIONREGIONS REGION1 REGION2 ... REGIONn)**

Returns a region which is the union of a number of regions, i.e. the smallest region that contains all of them. Returns NIL if there are no regions given.

**(REGIONSINTERSECTP REGION1 REGION2)**

Returns T if REGION1 intersects REGION2. Returns NIL if they do not intersect.

**(SUBREGIONP LARGEREGION SMALLREGION)**

Returns T if SMALLREGION is a subregion (is equal to or entirely contained in) LARGEREGION; otherwise returns NIL.

**(EXTENDREGION REGION INCLUDEREGION)**

Changes (destructively modifies) the region REGION so that it includes the region INCLUDEREGION. It returns REGION.

**(MAKEWITHINREGION REGION LIMITREGION)**

Changes (destructively modifies) the left and bottom of the region REGION so that it is within the region LIMITREGION, if possible. If the dimension of REGION are larger than LIMITREGION, REGION is moved to the lower left of LIMITREGION. If LIMITREGION is NIL, the value of the variable WHOLEDISPLAY (the screen region) is used. MAKEWITHINREGION returns the modified REGION.
27.1.3 Bitmaps

The display primitives manipulate graphical images in the form of bitmaps. A bitmap is a rectangular array of "pixels," each of which is an integer representing the color of one point in the bitmap image. A bitmap is created with a specific number of bits allocated for each pixel. Most bitmaps used for the display screen use one bit per pixel, so that at most two colors can be represented. If a pixel is 0, the corresponding location on the image is white. If a pixel is 1, its location is black. This interpretation can be changed for the display screen with the function VIDEOCOLOR (page 30.23). Bitmaps with more than one bit per pixel are used to represent color or grey scale images. Bitmaps use a positive integer coordinate system with the lower left corner pixel at coordinate (0,0). Bitmaps are represented as instances of the datatype BITMAP. Bitmaps can be saved on files with the VARS file package command (page 17.35).

(BitmapCreate WIDTH HEIGHT BITSPERPIXEL) [Function]
Creates and returns a new bitmap which is WIDTH pixels wide by HEIGHT pixels high, with BITSPERPIXEL bits per pixel. If BITSPERPIXEL is NIL, it defaults to 1.

(BitmapAPP X) [Function]
Returns X if X is a bitmap, NIL otherwise.

(BitmapWidth BITMAP) [Function]
Returns the width of BITMAP in pixels.

(BitmapHeight BITMAP) [Function]
Returns the height of BITMAP in pixels.

(BitPerPixel BITMAP) [Function]
Returns the number of bits per pixel of BITMAP.

(BitmapBit BITMAP X Y NEWVALUE) [Function]
If NEWVALUE is between 0 and the maximum value for a pixel in BITMAP, the pixel (X,Y) is changed to NEWVALUE and the old
value is returned. If NEWVALUE is NIL, BITMAP is not changed but the value of the pixel is returned. If NEWVALUE is anything else, an error is generated. If (X,Y) is outside the limits of BITMAP, 0 is returned and no pixels are changed. BITMAP can also be a window or display stream. Note: non-window image streams are "write-only"; the NEWVALUE argument must be non-NIL.

**BITMAPCOPY BITMAP**

Returns a new bitmap which is a copy of BITMAP (same dimensions, bits per pixel, and contents).

**EXPANDBITMAP BITMAP WIDTHFACTOR HEIGHTFACTOR**

Returns a new bitmap that is WIDTHFACTOR times as wide as BITMAP and HEIGHTFACTOR times as high. Each pixel of BITMAP is copied into a WIDTHFACTOR times HEIGHTFACTOR block of pixels. If NIL, WIDTHFACTOR defaults to 4, HEIGHTFACTOR to 1.

**SHRINKBITMAP BITMAP WIDTHFACTOR HEIGHTFACTOR DESTINATIONBITMAP**

Returns a copy of BITMAP that has been shrunken by WIDTHFACTOR and HEIGHTFACTOR in the width and height, respectively. If NIL, WIDTHFACTOR defaults to 4, HEIGHTFACTOR to 1. If DESTINATIONBITMAP is not provided, a bitmap that is 1/WIDTHFACTOR by 1/HEIGHTFACTOR the size of BITMAP is created and returned. WIDTHFACTOR and HEIGHTFACTOR must be positive integers.

**PRINTBITMAP BITMAP FILE**

Prints the bitmap BITMAP on the file FILE in a format that can be read back in by READBITMAP.

**READBITMAP FILE**

Creates a bitmap by reading an expression (written by PRINTBITMAP) from the file FILE.

**EDITBM BMSPEC**

EDITBM provides an easy-to-use interactive editing facility for various types of bitmaps. If BMSPEC is a bitmap, it is edited. If BMSPEC is an atom whose value is a bitmap, its value is edited. If BMSPEC is NIL, EDITBM asks for dimensions and creates a bitmap. If BMSPEC is a region, that portion of the screen bitmap is used. If BMSPEC is a window, it is brought to the top and its contents edited.

EDITBM sets up the bitmap being edited in an editing window. The editing window has two major areas: a gridded edit area in
the lower part of the window and a display area in the upper left part. In the edit area, the left button will add points, the middle button will erase points. The right button provides access to the normal window commands to reposition and reshape the window. The actual size bitmap is shown in the display area. For example, the following is a picture of the bitmap editing window editing a eight-high by eighteen-wide bitmap:

![Bitmap Editor](image)

If the bitmap is too large to fit in the edit area, only a portion will be editable. This portion can be changed by scrolling both up and down in the left margin and left and right in the bottom margin. Pressing the middle button while in the display area will bring up a menu that allows global placement of the portion of the bitmap being edited. To allow more of the bitmap to be editing at once, the window can be reshaped to make it larger or the GridSize+ command described below can be used to reduce the size of a bit in the edit area.

The bitmap editing window can be reshaped to provide more or less room for editing. When this happens, the space allocated to the editing area will be changed to fit in the new region.

Whenever the left or middle button is down and the cursor is not in the edit area, the section of the display of the bitmap that is currently in the edit area is complemented. Pressing the left button while not in the edit region will put the lower left 16 x 16 section of the bitmap into the cursor for as long as the left button is held down.

Pressing the middle button while not in either the edit area or the display area (i.e. while in the grey area in the upper right or in the title) will bring up a command menu. There are commands to stop editing, to restore the bitmap to its initial state and to clear the bitmap. Holding the middle button down over a command will result in an explanatory message being printed in the prompt window. The commands are described below:

- **Paint** Puts the current bitmap into a window and call the window PAINT command on it. The PAINT command implements drawing with various brush sizes and shapes but only on an
actual sized bitmap. The PAINT mode is left by pressing the RIGHT button and selecting the QUIT command from the menu. At this point, you will be given a choice of whether or not the changes you made while in PAINT mode should be made to the current bitmap.

**ShowAsTile**
Tessellates the current bitmap in the upper part of the window. This is useful for determining how a bitmap will look if it were made the display background (using the function CHANGEBACKGROUND). Note: The tiled display will not automatically change as the bitmap changes; to update it, use the ShowAsTile command again.

**Grid,On/Off**
Turns the editing grid display on or off.

**GridSize+−**
Allows specification of the size of the editing grid. Another menu will appear giving a choice of several sizes. If one is selected, the editing portion of the bitmap editor will be redrawn using the selected grid size, allowing more or less of the bitmap to be edited without scrolling. The original size is chosen heuristically and is typically about 8. It is particularly useful when editing large bitmaps to set the edit grid size smaller than the original.

**Reset**
Sets all or part of the bitmap to the contents it had when EDITBM was called. Another menu will appear giving a choice between resetting the entire bitmap or just the portion that is in the edit area. The second menu also acts as a confirmation, since not selecting one of the choices on this menu results in no action being taken.

**Clear**
Sets all or part of the bitmap to 0. As with the Reset command, another menu gives a choice between clearing the entire bitmap or just the portion that is in the edit area.

**Cursor+−**
Sets the cursor to the lower left part of the bitmap. This prompts the user to specify the cursor "hot spot" (see page 30.14) by clicking in the lower left corner of the grid.

**OK**
Copies the changed image into the original bitmap, stops the bitmap editor and closes the edit windows. The changes the bitmap editor makes during the interaction occur on a copy of the original bitmap. Unless the bitmap editor is exited via OK, no changes are made in the original.

**Stop**
Stops the bitmap editor without making any changes to the original bitmap.

### 27.1.4 Textures

A Texture denotes a pattern of gray which can be used to (conceptually) tessellate the plane to form an infinite sheet of gray. It is currently either a 4 by 4 pattern or a 16 by \( N \) (\( N \leq 16 \))
pattern. Textures are created from bitmaps using the following function:

(CREATETEXTUREFROMBITMAP BITMAP)  \[\text{Function}\]

Returns a texture object that will produce the texture of \textit{BITMAP}. If \textit{BITMAP} is too large, its lower left portion is used. If \textit{BITMAP} is too small, it is repeated to fill out the texture.

(TEXTUREP OBJECT)  \[\text{Function}\]

Returns \textit{OBJECT} if it is a texture; \textit{NIL} otherwise.

The functions which accept textures (TEXTUREP, BITBLT, DSPTEXTURE, etc.) also accept bitmaps up to 16 bits wide by 16 bits high as textures. When a region is being filled with a bitmap texture, the texture is treated as if it were 16 bits wide (if less, the rest is filled with white space).

The common textures white and black are available as system constants \texttt{WHITESHADE} and \texttt{BLACKSHADE}. The global variable \texttt{GRAYSHADE} is used by many system facilities as a background gray shade and can be set by the user.

(EDITSHADE SHADE)  \[\text{Function}\]

Opens a window that allows the user to edit textures. Textures can be either small (4 by 4) patterns or large (16 by 16). In the edit area, the left button adds bits to the shade and the middle button erases bits from the shade. The top part of the window is painted with the current texture whenever all mouse keys are released. Thus it is possible to directly compare two textures that differ by more than one pixel by holding a mouse key down until all changes are made. When the "quit" button is selected, the texture being edited is returned.

If \textit{SHADE} is a texture object, \texttt{EDITSHADE} starts with it. If \textit{SHADE} is \texttt{T}, it starts with a large (16 by 16) white texture. Otherwise, it starts with \texttt{WHITESHADE}.

The following is a picture of the texture editor, editing a large (16 by 16) pattern:
27.2 Opening Image Streams

An image stream is an output stream which "knows" how to process graphic commands to a graphics output device. Besides accepting the normal character-output functions (PRINT, etc.), an image stream can also be passed as an argument to functions to draw curves, to print characters in multiple fonts, and other graphics operations.

Each image stream has an "image stream type," a litatom that specifies the type of graphic output device that the image stream is processing graphics commands for. Currently, the built-in image stream types are DISPLAY (for the display screen), INTERPRESS (for Interpress format printers), and PRESS (for Press format printers). There are also library packages available that define image stream types for the IRIS display, 4045 printer, FX-80 printer, C150 printer, etc.

Image streams to the display (display streams) interpret graphics commands by immediately executing the appropriate operations to cause the desired image to appear on the display screen. Image streams for hardcopy devices such as Interpress printers interpret the graphic commands by saving information in a file, which can later be sent to the printer.
Note: Not all graphics operations can be properly executed for all image stream types. For example, BITBLT may not be supported to all printers. This functionality is still being developed, but even in the long run some operations may be beyond the physical or logical capabilities of some devices or image file formats. In these cases, the stream will approximate the specified image as best it can.

(OPENIMAGESTREAM FILE IMAGETYPE OPTIONS)  [Function]
Opens and returns an image stream of type IMAGETYPE on a destination specified by FILE. If FILE is a file name on a normal file storage device, the image stream will store graphics commands on the specified file, which can be transmitted to a printer by explicit calls to LISTFILES and SEND.FILE.TO.PRINTER. If IMAGETYPE is DISPLAY, then the user is prompted for a window to open. FILE in this case will be used as the title of the window.

If FILE is a file name on the LPT device, this indicates that the graphics commands should be stored in a temporary file, and automatically sent to the printer when the image stream is closed by CLOSEF. FILE = NIL is equivalent to FILE = {LPT}. File names on the LPT device are of the form {LPT}PRINTERNAME.TYPE, where PRINTERNAME, TYPE, or both may be omitted. PRINTERNAME is the name of the particular printer to which the file will be transmitted on closing; it defaults to the first printer on DEFAULTPRINTINGHOST that can print IMAGETYPE files. The TYPE extension supplies the value of IMAGETYPE when it is defaulted (see below). OPENIMAGESTREAM will generate an error if the specified printer does not accept the kind of file specified by IMAGETYPE.

If IMAGETYPE is NIL, the image type is inferred from the extension field of FILE and the EXTENSIONS properties in the list PRINTFILETYPES (see page 29.6). Thus, the extensions IP, IPR, and INTERPRESS indicate Interpress format, and the extension PRESS indicates Press format. If FILE is a printer file with no extension (of the form {LPT}PRINTERNAME), then IMAGETYPE will be the type that the indicated printer can print. If FILE has no extension but is not on the printer device {LPT}, then IMAGETYPE will default to the type accepted by the first printer on DEFAULTPRINTINGHOST.

OPTIONS is a list in property list format, (PROP1 VAL1 PROP2 VAL2 →), used to specify certain attributes of the image stream; not all attributes are meaningful or interpreted by all types of image streams. Acceptable properties are:

REGION Value is the region on the page (in stream scale units, 0,0 being the lower-left corner of the page) that text will fill up. It establishes the initial values for DSPLEFTMARGIN,
DSPRIGHTMARGIN, DSPBOTTOMMARGIN (the point at which carriage returns cause page advancement) and DSPTOPMARGIN (where the stream is positioned at the beginning of a new page). If this property is not given, the value of the variable DEFAULTPAGEREGION, is used.

**FONTS**
Value is a list of fonts that are expected to be used in the image stream. Some image streams (e.g. Interpress) are more efficient if the expected fonts are specified in advance, but this is not necessary. The first font in this list will be the initial font of the stream, otherwise the default font for that image stream type will be used.

**HEADING**
Value is the heading to be placed automatically on each page. NIL means no heading.

Examples: Suppose that Tremor: is an Interpress printer, Quake is a Press printer, and DEFAULTPRINTINGHOST is (Tremor: Quake):

**(OPENIMAGESTREAM)** returns an Interpress image stream on printer Tremor:.

**(OPENIMAGESTREAM NIL 'PRESS)** returns a Press stream on Quake.

**(OPENIMAGESTREAM '{LPT}.INTERPRESS)** returns an Interpress stream on Tremor:.

**(OPENIMAGESTREAM '{CORE}FOO.PRESS)** returns a Press stream on the file '{CORE}FOO.PRESS.

**(IMAGESTREAMP X IMAGETYPE)** [NoSpread Function]
Returns X (possibly coerced to a stream) if it is an output image stream of type IMAGETYPE (or of any type if IMAGETYPE = NIL), otherwise NIL.

**(IMAGESTREAMTYPE STREAM)** [Function]
Returns the image stream type of STREAM.

**(IMAGESTREAMTYPEP STREAM TYPE)** [Function]
Returns T if STREAM is an image stream of type TYPE.

### 27.3 Accessing Image Stream Fields

The following functions manipulate the fields of an image stream. These functions return the old value (the one being replaced). A value of NIL for the new value will return the
current setting without changing it. These functions do not change any of the bits drawn on the image stream; they just affect future operations done on the image stream.

(DSPCLIPPINGREGION REGION STREAM) [Function]
The clipping region is a region that limits the extent of characters printed and lines drawn (in the image stream's coordinate system). Initially set so that no clipping occurs.
Warning: For display streams, the window system maintains the clipping region during window operations. Users should be very careful about changing this field.

(DSPFONT FONT STREAM) [Function]
The font field specifies the font (see page 27.25) used when printing characters to the image stream.
Note: DSPFONT determines its new font descriptor from FONT by the same coercion rules that FONTPROP and FONTCREATE use (page 27.26), with one additional possibility: If FONT is a list of the form (PROP \_1 VAL \_1 PROP \_2 VAL \_2 ... ) where PROP \_1 is acceptable as a font-property to FONTCOPY (page 27.28), then the new font is obtained by (FONTCOPY (DSPFONT NIL STREAM) PROP \_1 VAL \_1 PROP \_2 VAL \_2 ... ). For example, (DSPFONT '(SIZE 12) STREAM) would change the font to the 12 point version of the current font, leaving all other font properties the same.

(DSPTOPMARGIN YPOSITION STREAM) [Function]
The top margin is an integer that is the Y position after a new page (in the image stream's coordinate system). This function has no effect on windows.

(DSPBOTTOMMARGIN YPOSITION STREAM) [Function]
The bottom margin is an integer that is the minimum Y position that characters will be printed by PRIN1 (in the image stream's coordinate system). This function has no effect on windows.

(DSPLEFTMARGIN XPOSITION STREAM) [Function]
The left margin is an integer that is the X position after an end-of-line (in the image stream's coordinate system). Initially the left edge of the clipping region.

(DSPRIGHTMARGIN XPOSITION STREAM) [Function]
The right margin is an integer that is the maximum X position that characters will be printed by PRIN1 (in the image stream's coordinate system). This is initially the position of the right edge of the window or page.
The line length of a window or image stream (as returned by LINELENGTH, page 25.11) is computed by dividing the distance between the left and right margins by the width of an uppercase "A" in the current font. The line length is changed whenever the font, left margin, or right margin are changed or whenever the window is reshaped.

**DSPOPERATION**

The operation is the default BITBLT operation (see page 27.15) used when printing or drawing on the image stream. One of REPLACE, PAINT, INVERT, or ERASE. Initially REPLACE. This is a meaningless operation for most printers which support the model that once dots are deposited on a page they cannot be removed.

**DSPLINEFEED**

The linefeed is an integer that specifies the Y increment for each linefeed, normally negative. Initially minus the height of the initial font.

**DSPSCALE**

Returns the scale of the image stream STREAM, a number indicating how many units in the streams coordinate system correspond to one printer's point (1/72 of an inch). For example, DSPSCALE returns 1 for display streams, and 35.27778 for Interpress and Press streams (the number of micas per printer's point). In order to be device-independent, user graphics programs must either not specify position values absolutely, or must multiply absolute point quantities by the DSPSCALE of the destination stream. For example, to set the left margin of the Interpress stream XX to one inch, do:

(DSPLPARAMETERS (TIMES 72 (DSPSCALE NIL XX)) XX)

The SCALE argument to DSPSCALE is currently ignored. In a future release it will enable the scale of the stream to be changed under user control, so that the necessary multiplication will be done internal to the image stream interface. In this case, it would be possible to set the left margin of the Interpress stream XX to one inch by doing:

(DSPSCALE 1 XX)
(DSPLPARAMETERS 72 XX)

**DSPSPACEFACTOR**

The space factor is the amount by which to multiply the natural width of all following space characters on STREAM; this can be used for the justification of text. The default value is 1. For example, if the natural width of a space in STREAM's current font...
is 12 units, and the space factor is set to two, spaces appear 24 units wide. The values returned by STRINGWIDTH and CHARWIDTH are also affected.

The following two functions only have meaning for image streams that can display color:

**DSPCOLOR COLOR STREAM**

Sets the default foreground color of STREAM. Returns the previous foreground color. If COLOR is NIL, it returns the current foreground color without changing anything. The default color is white.

**DSPBACKCOLOR COLOR STREAM**

Sets the background color of STREAM. Returns the previous background color. If COLOR is NIL, it returns the current background color without changing anything. The default background color is black.

### 27.4 Current Position of an Image Stream

Each image stream has a "current position," which is a position (in the image stream's coordinate system) where the next printing operation will start from. The functions which print characters or draw on an image stream update these values appropriately. The following functions are used to explicitly access the current position of an image stream:

**DSPXPOSITION XPOSITION STREAM**

Returns the X coordinate of the current position of STREAM. If XPOSITION is non-NIL, the X coordinate is set to it (without changing the Y coordinate).

**DSPYPOSITION YPOSITION STREAM**

Returns the Y coordinate of the current position of STREAM. If YPOSITION is non-NIL, the Y coordinate is set to it (without changing the X coordinate).

**MOVETO X Y STREAM**

Changes the current position of STREAM to the point (X,Y).
CURRENT POSITION OF AN IMAGE STREAM

(RELMOVETO DX DY STREAM)  [Function]
Changes the current position to the point (DX,DY) coordinates away from current position of STREAM.

(MOVETOUPPERLEFT STREAM REGION)  [Function]
Moves the current position to the beginning position of the top line of text. If REGION is non-NIL, it must be a REGION and the X position is changed to the left edge of REGION and the Y position changed to the top of REGION less the font ascent of STREAM. If REGION is NIL, the X coordinate is changed to the left margin of STREAM and the Y coordinate is changed to the top of the clipping region of STREAM less the font ascent of STREAM.

27.5 Moving Bits Between Bitmaps With BITBLT

BITBLT is the primitive function for moving bits from one bitmap to another, or from a bitmap to an image stream.

(BITBLT SOURCE SOURCELEFT SOURCEBOTTOM DESTINATION DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION)  [Function]
Transfers a rectangular array of bits from SOURCE to DESTINATION. SOURCE can be a bitmap, or a display stream or window, in which case its associated bitmap is used. DESTINATION can be a bitmap or an arbitrary image stream.

WIDTH and HEIGHT define a pair of rectangles, one in each of the SOURCE and DESTINATION whose left, bottom corners are at, respectively, (SOURCELEFT, SOURCEBOTTOM) and (DESTINATIONLEFT, DESTINATIONBOTTOM). If these rectangles overlap the boundaries of either source or destination they are both reduced in size (without translation) so that they fit within their respective boundaries. If CLIPPINGREGION is non-NIL it should be a REGION and is interpreted as a clipping region within DESTINATION; clipping to this region may further reduce the defining rectangles. These (possibly reduced) rectangles define the source and destination rectangles for BITBLT.

The mode of transferring bits is defined by SOURCETYPE and OPERATION. SOURCETYPE and OPERATION specify whether the source bits should come from SOURCE or TEXTURE, and how these bits are combined with those of DESTINATION. SOURCETYPE and OPERATION are described further below.

TEXTURE is a texture, as described on page 27.6. BITBLT aligns the texture so that the upper-left pixel of the texture coincides with the upper-left pixel of the destination bitmap.
SOURCELEFT, SOURCENOTTOM, DESTINATIONLEFT, and DESTINATIONBOFOM default to 0. WIDTH and HEIGHT default to the width and height of the SOURCE. TEXTURE defaults to white. SOURCETYPE defaults to INPUT. OPERATION defaults to REPLACE. If CLIPPINGREGION is not provided, no additional clipping is done. BITBLT returns T if any bits were moved; NIL otherwise.

Note: If SOURCE or DESTINATION is a window or image stream, the remaining arguments are interpreted as values in the coordinate system of the window or image stream and the operation of BITBLT is translated and clipped accordingly. Also, if a window or image stream is used as the destination to BITBLT, its clipping region further limits the region involved.

SOURCETYPE specifies whether the source bits should come from the bitmap SOURCE, or from the texture TEXTURE. SOURCETYPE is interpreted as follows:

**INPUT** The source bits come from SOURCE. TEXTURE is ignored.

**INVERT** The source bits are the inverse of the bits from SOURCE. TEXTURE is ignored.

**TEXTURE** The source bits come from TEXTURE. SOURCE, SOURCELEFT, and SOURCENOTTOM are ignored.

OPERATION specifies how the source bits (as specified by SOURCETYPE) are combined with the bits in DESTINATION and stored back into DESTINATION. DESTINATION is one of the following:

**REPLACE** All source bits (on or off) replace destination bits.

**PAINT** Any source bits that are on replace the corresponding destination bits. Source bits that are off have no effect. Does a logical OR between the source bits and the destination bits.

**INVERT** Any source bits that are on invert the corresponding destination bits. Does a logical XOR between the source bits and the destination bits.

**ERASE** Any source bits that are on erase the corresponding destination bits. Does a logical AND operation between the inverse of the source bits and the destination bits.

Different combinations of SOURCETYPE and OPERATION can be specified to achieve many different effects. Given the following bitmaps as the values of SOURCE, TEXTURE, and DESTINATION:
BITBLT would produce the results given below for the difference combinations of \textit{SOURCETYPE} and \textit{OPERATION} (assuming \textit{CLIPPINGREGION}, \textit{SOURCELEREF}, etc. are set correctly, of course):

\begin{itemize}
  \item \textit{SOURCETYPE} = \textit{INPUT}
    \begin{itemize}
      \item \textit{OP} = REPLACE
        \begin{itemize}
          \item Type: \textit{INPUT}
          \item \textit{OP} = REPLACE
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = PAINT
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = INVERT
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = ERASE
        \end{itemize}
    \end{itemize}
  \item \textit{SOURCETYPE} = \textit{INVERT}
    \begin{itemize}
      \item \textit{OP} = REPLACE
        \begin{itemize}
          \item Type: \textit{INPUT}
          \item \textit{OP} = PAINT
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = INVERT
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = ERASE
        \end{itemize}
    \end{itemize}
  \item \textit{SOURCETYPE} = \textit{TEXTURE}
    \begin{itemize}
      \item \textit{OP} = REPLACE
        \begin{itemize}
          \item Type: \textit{INPUT}
          \item \textit{OP} = PAINT
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = INVERT
          \item Type: \textit{TEXTURE}
          \item \textit{OP} = ERASE
        \end{itemize}
    \end{itemize}
\end{itemize}

\texttt{BITLSHADE TEXTURE DESTINATION LEFT DESTINATIONBOTTOM WIDTH}
\texttt{HEIGHT OPERATION CLIPPINGREGION) [Function]}

\textit{BITLSHADE} is the \textit{SOURCETYPE} = \textit{TEXTURE} case of \textit{BITBLT}. It fills the specified region of the destination bitmap \textit{DESTINATION} with the texture \textit{TEXTURE}. \textit{DESTINATION} can be a bitmap or image stream.

\texttt{BITMAPIMAGESIZE BITMAP DIMENSION STREAM) [Function]}

Returns the size that \textit{BITMAP} will be when \textit{BITBLT}ed to \textit{STREAM}, in \textit{STREAM}'s units. \textit{DIMENSION} can be one of \textit{WIDTH}, \textit{HEIGHT}, or \textit{NIL}, in which case the dotted pair (\textit{WIDTH}, \textit{HEIGHT}) will be returned.
27.6 Drawing Lines

Interlisp-D provides several functions for drawing lines and curves on image streams. The line drawing functions are intended for interactive applications where efficiency is important. They do not allow the use of "brush" patterns, like the curve drawing functions, but (for display streams) they support drawing a line in INVERT mode, so redrawing the line will erase it. DRAWCURVE (page 27.19) can be used to draw lines using a brush.

**DRAWLINE**

```
(DRAWLINE X_1 Y_1 X_2 Y_2 WIDTH OPERATION STREAM COLOR DASHING) [Function]
```

Draws a straight line from the point \((X_1, Y_1)\) to the point \((X_2, Y_2)\) on the image stream \(STREAM\). The position of \(STREAM\) is set to \((X_2, Y_2)\). If \(X_1\) equals \(X_2\) and \(Y_1\) equals \(Y_2\), a point is drawn at \((X_1, Y_1)\).

**WIDTH** is the width of the line, in the units of the device. If **WIDTH** is NIL, the default is 1.

**OPERATION** is the BITBLT operation (see page 27.15) used to draw the line. If **OPERATION** is NIL, the value of **DSPOPERATION** for the image stream is used.

**COLOR** is a color specification that determines the color used to draw the line for image streams that support color. If **COLOR** is NIL, the **DSPCOLOR** of \(STREAM\) is used.

**DASHING** is a list of positive integers that determines the dashing characteristics of the line. The line is drawn for the number of points indicated by the first element of the dashing list, is not drawn for the number of points indicated by the second element. The third element indicates how long it will be on again, and so forth. The dashing sequence is repeated from the beginning when the list is exhausted. If **DASHING** is NIL, the line is not dashed.

**DRAWBETWEEN**

```
(DRAWBETWEEN POSITION_1 POSITION_2 WIDTH OPERATION STREAM COLOR DASHING) [Function]
```

Draws a line from the point \(POSITION_1\) to the point \(POSITION_2\) onto the destination bitmap of \(STREAM\). The position of \(STREAM\) is set to \(POSITION_2\).

**DRAWTO**

```
(DRAWTO X Y WIDTH OPERATION STREAM COLOR DASHING) [Function]
```

Draws a line from the current position to the point \((X, Y)\) onto the destination bitmap of \(STREAM\). The position of \(STREAM\) is set to \((X, Y)\).
DRAWING LINES

(RELDRAWTO DX DY WIDTH OPERATION STREAM COLOR DASHING)  [Function]

Draws a line from the current position to the point (DX,DY) coordinates away onto the destination bitmap of STREAM. The position of STREAM is set to the end of the line. If DX and DY are both 0, nothing is drawn.

27.7 Drawing Curves

A curve is drawn by placing a brush pattern centered at each point along the curve's trajectory. A brush pattern is defined by its shape, size, and color. The predefined brush shapes are ROUND, SQUARE, HORIZONTAL, VERTICAL, and DIAGONAL; new brush shapes can be created using the INSTALLBRUSH function, described below. A brush size is an integer specifying the width of the brush in the units of the device. The color is a color specification, which is only used if the curve is drawn to an image stream that supports colors.

A brush is specified to the various drawing functions as a list of the form (SHAPE WIDTH COLOR), for example (SQUARE 2) or (VERTICAL 4 RED). A brush can also be specified as a positive integer, which is interpreted as a ROUND brush of that width. If a brush is a litatom, it is assumed to be a function which is called at each point of the curve's trajectory (with three arguments: the X-coordinate of the point, the Y-coordinate, and the image stream), and should do whatever image stream operations are necessary to draw each point. Finally, if a brush is specified as NIL, a (ROUND 1) brush is used as default.

The appearance of a curve is also determined by its dashing characteristics. Dashing is specified by a list of positive integers. If a curve is dashed, the brush is placed along the trajectory for the number of units indicated by the first element of the dashing list. The brush is off, not placed in the bitmap, for a number of units indicated by the second element. The third element indicates how long it will be on again, and so forth. The dashing sequence is repeated from the beginning when the list is exhausted. The units used to measure dashing are the units of the brush. For example, specifying the dashing as (1 1) with a brush of (ROUND 16) would put the brush on the trajectory, skip 16 points, and put down another brush. A curve is not dashed if the dashing argument to the drawing function is NIL.

The curve functions use the image stream's clipping region and operation. Most types of image streams only support the PAINT operation when drawing curves. When drawing to a display stream, the curve-drawing functions accept the operation INVERT if the brush argument is 1. For brushes larger than 1,
these functions will use the ERASE operation instead of INVERT. For display streams, the curve-drawing functions treat the REPLACE operation the same as PAINT.

(DRAWCURVE KNOTS CLOSED BRUSH DASHING STREAM) [Function]
Draws a "parametric cubic spline curve" on the image stream STREAM. KNOTS is a list of positions to which the curve will be fitted. If CLOSED is non-NIL, the curve will be closed; otherwise it ends at the first and last positions in KNOTS. BRUSH and DASHING are interpreted as described above.

For example,

(DRAWCURVE '((10 . 10)(50 . 50)(100 . 10)(150 . 50))
  NIL (ROUND 5)'(1 1 1 2) XX)

would draw a curve like the following on the display stream XX:

![Curve Example]

(DRAWCIRCLE CENTERX CENTERY RADIUS BRUSH DASHING STREAM) [Function]
Draws a circle of radius RADIUS about the point (CENTERX,CENTERY) onto the image stream STREAM. STREAM's position is left at (CENTERX,CENTERY). The other arguments are interpreted as described above.

(DRAWEllipse CENTERX CENTERY SEMIMINORADIUS SEMIMAJORRADIUS ORIENTATION
  BRUSH DASHING STREAM) [Function]
Draws an ellipse with a minor radius of SEMIMINORADIUS and a major radius of SEMIMAJORRADIUS about the point (CENTERX,CENTERY) onto the image stream STREAM. ORIENTATION is the angle of the major axis in degrees, positive in the counterclockwise direction. STREAM's position is left at (CENTERX,CENTERY). The other arguments are interpreted as described above.

New brush shapes can be defined using the following function:

(INSTALLBRUSH BRUSHNAME BRUSHFN BRUSHARRAY) [Function]
Installs a new brush called BRUSHNAME with creation-function BRUSHFN and optional array BRUSHARRAY. BRUSHFN should be a function of one argument (a width), which returns a bitmap of the brush for that width. BRUSHFN will be called to create new instances of BRUSHNAME-type brushes; the sixteen smallest instances will be pre-computed and cached. "Hand-crafted" brushes can be supplied as the BRUSHARRAY argument.
Changing an existing brush can be done by calling INSTALLBRUSH with new BRUSHFN and/or BRUSHARRAY.

(DRAWPOINT X Y BRUSH STREAM OPERATION)
[Function]
Draws BRUSH centered around point (X, Y) on STREAM, using the operation OPERATION. BRUSH may be a bitmap or a brush.

27.8 Miscellaneous Drawing and Printing Operations

(DSPFILL REGION TEXTURE OPERATION STREAM)
[Function]
Fills REGION of the image stream STREAM (within the clipping region) with the texture TEXTURE. If REGION is NIL, the whole clipping region of STREAM is used. If TEXTURE or OPERATION is NIL, the values for STREAM are used.

(FILLPOLYGON POINTS TEXTURE STREAM)
[Function]
Fills in the polygon outlined by POINTS on the image stream STREAM, using the texture TEXTURE.

POINTS is a list of positions (page 27.1) determining the vertices of a closed polygon. FILLPOLYGON fills in this polygon with the texture TEXTURE. POINTS can also be a list whose elements are lists of positions, in which case each sublist describes a separate polygon to be filled.

Note: When filling a polygon, there is more than one way of dealing with the situation where two polygon sides intersect, or one polygon is fully inside the other. Currently, FILLPOLYGON to a display stream uses the "odd" fill rule, which means that intersecting polygon sides define areas that are filled or not filled somewhat like a checkerboard. For example, (FILLPOLYGON '((125.125)(150.200)(190.110)(135.125)(160.125)(160.150)(135.150)) GRAYSHADE WINDOW) would produce a display something like this:

```
  ____________
 /             \
 |             |
 |             |
 |             |
 \____________/```

This fill convention also takes into account all polygons in POINTS, if it specifies multiple polygons. This can be used to put "holes" in filled polygons. For example,

(FILLPOLYGON
  '((110.110)(150.200)(190.110))
  '((135.125)(160.125)(160.150)(135.150))
)
GRAYSHADE WINDOW

will put a square hole in a triangular region:

Currently, FILLPOLYGON uses the "Replace" BITBLT operation (see page 27.15) to fill areas with the texture. However, any areas that are not filled are not changed. If there are "holes" in the filled polygon, this can be used to produce a "window" effect. For example, the following is the display produced by filling the star polygon (above) over a window full of text:

```
  Text  Text  Text
  Text  Text  Text
  Text  Text  Text
  Text  Text  Text
  Text  Text  Text
```

(FILLCIRCLE CENTERX CENTERY RADIUS TEXTURE STREAM) [Function]

Fills in a circular area of radius RADIUS about the point (CENTERX, CENTERY) in STREAM with TEXTURE. STREAM's position is left at (CENTERX, CENTERY).

(DSPRESET STREAM) [Function]

Sets the X coordinate of STREAM to its left margin, sets its Y coordinate to the top of the clipping region minus the font ascent. For a display stream, this also fills its destination bitmap with its background texture.

(DSPNEWPAGE STREAM) [Function]

Starts a new page. The X coordinate is set to the left margin, and the Y coordinate is set to the top margin plus the linefeed.

(CENTERPRINTINREGION EXP REGION STREAM) [Function]

Prints EXP so that it is centered within REGION of the STREAM. If REGION is NIL, EXP will be centered in the clipping region of STREAM.
A grid is a partitioning of an arbitrary coordinate system (hereafter referred to as the "source system") into rectangles. This section describes functions that operate on grids. It includes functions to draw the outline of a grid, to translate between positions in a source system and grid coordinates (the coordinates of the rectangle which contains a given position), and to shade grid rectangles. A grid is defined by its "unit grid," a region (called a grid specification) which is the origin rectangle of the grid in terms of the source system. Its LEFT field is interpreted as the X-coordinate of the left edge of the origin rectangle, its BOTTOM field is the Y-coordinate of the bottom edge of the origin rectangle, its WIDTH is the width of the grid rectangles, and its HEIGHT is the height of the grid rectangles.

\textbf{(GRID GRIDSPEC WIDTH HEIGHT BORDER STREAM GRIDSHADE)} \hspace{1cm} \textbf{[Function]}

Outlines the grid defined by \textit{GRIDSPEC} which is \textit{WIDTH} rectangles wide and \textit{HEIGHT} rectangles high on \textit{STREAM}. Each box in the grid has a border within it that is \textit{BORDER} points on each side; so the resulting lines in the grid are \(2 \times \textit{BORDER}\) thick. If \textit{BORDER} is the atom \textit{POINT}, instead of a border the lower left point of each grid rectangle will be turned on. If \textit{GRIDSHADE} is non-NIL, it should be a texture and the border lines will be drawn using that texture.

\textbf{(SHADEGRIDBOX X Y SHADE OPERATION GRIDSPEC GRIDBORDER STREAM)} \hspace{1cm} \textbf{[Function]}

Shades the grid rectangle \((X, Y)\) of \textit{GRIDSPEC} with texture \textit{SHADE} using \textit{OPERATION} on \textit{STREAM}. \textit{GRIDBORDER} is interpreted the same as for \textit{GRID}.

The following two functions map from the X,Y coordinates of the source system into the grid X,Y coordinates:

\textbf{(GRIDXCOORD XCOORD GRIDSPEC)} \hspace{1cm} \textbf{[Function]}

Returns the grid X-coordinate (in the grid specified by \textit{GRIDSPEC}) that contains the source system X-coordinate \textit{XCOORD}.

\textbf{(GRIDYCOORD YCOORD GRIDSPEC)} \hspace{1cm} \textbf{[Function]}

Returns the grid Y-coordinate (in the grid specified by \textit{GRIDSPEC}) that contains the source system Y-coordinate \textit{YCOORD}.

The following two functions map from the grid X,Y coordinates into the X,Y coordinates of the source system:
(LEFTOFGRIDCOORD GRIDX GRIDSPEC)  [Function]
Returns the source system X-coordinate of the left edge of a grid rectangle at grid X-coordinate GRIDX (in the grid specified by GRIDSPEC).

(BOTTOMOFGRIDCOORD GRIDY GRIDSPEC)  [Function]
Returns the source system Y-coordinate of the bottom edge of a grid rectangle at grid Y-coordinate GRIDY (in the grid specified by GRIDSPEC).

27.10 Display Streams

Display streams (image streams of type DISPLAY) are used to control graphic output operations to a bitmap, known as the "destination" bitmap of the display stream. For each window on the screen, there is an associated display stream which controls graphics operations to a specific part of the screen bitmap. Any of the functions that take a display stream will also take a window, and use the associated display stream. Display streams can also have a destination bitmap that is not connected to any window or display device.

(DSPCREATE DESTINATION)  [Function]
Creates and returns a display stream. If DESTINATION is specified, it is used as the destination bitmap, otherwise the screen bitmap is used.

(DSPDESTINATION DESTINATION DISPLAYSTREAM)  [Function]
Returns the current destination bitmap for DISPLAYSTREAM, setting it to DESTINATION if non-NIL. DESTINATION can be either the screen bitmap, or an auxiliary bitmap in order to construct figures, possibly save them, and then display them in a single operation.

Warning: The window system maintains the destination of a window's display stream. Users should be very careful about changing this field.

(DSPXOFFSET XOFFSET DISPLAYSTREAM)  [Function]
(DSPYOFFSET YOFFSET DISPLAYSTREAM)  [Function]
Each display stream has its own coordinate system, separate from the coordinate system of its destination bitmap. Having the coordinate system local to the display stream allows objects to be displayed at different places by translating the display stream's
coordinate system relative to its destination bitmap. This local coordinate system is defined by the X offset and Y offset.

\[ \text{DSPXOFFSET} \] returns the current X offset for \text{DISPLAYSTREAM}, the X origin of the display stream's coordinate system in the destination bitmap's coordinate system. It is set to \text{XOFFSET} if non-NIL.

\[ \text{DSPYOFFSET} \] returns the current Y offset for \text{DISPLAYSTREAM}, the Y origin of the display stream's coordinate system in the destination bitmap's coordinate system. It is set to \text{YOFFSET} if non-NIL.

The X offset and Y offset for a display stream are both initially 0 (no X or Y-coordinate translation).

Warning: The window system maintains the X and Y offset of a window's display stream. Users should be very careful about changing these fields.

\[ \text{(DSPTEXTURE TEXTURE DISPLAYSTREAM)} \] [Function]
Returns the current texture used as the background pattern for \text{DISPLAYSTREAM}. It is set to \text{TEXTURE} if non-NIL. Initially the value of \text{WHITESTRING}.

\[ \text{(DSPSOURCETYPE SOURCETYPE DISPLAYSTREAM)} \] [Function]
Returns the current \text{BITBLT} sourcetype used when printing characters to the display stream (see page 27.15). It is set to \text{SOURCETYPE}, if non-NIL. Must be either \text{INPUT} or \text{INVERT}. Initially \text{INPUT}.

\[ \text{(DSPSCROLL SWITCHSETTING DISPLAYSTREAM)} \] [Function]
Returns the current value of the "scroll flag," a flag that determines the scrolling behavior of the display stream; either \text{ON} or \text{OFF}. If \text{ON}, the bits in the display stream's destination bitmap are moved after any linefeed that moves the current position out of the destination bitmap. Any bits moved out of the current clipping region are lost. Does not adjust the X offset, Y offset, or clipping region of the display stream. Initially \text{OFF}.
Sets the scroll flag to \text{SWITCHSETTING}, if non-NIL.

Note: The word "scrolling" also describes the use of "scroll bars" on the left and bottom of a window to move an object displayed in a window. This feature is described on page 28.23.

Each window has an associated display stream. To get the window of a particular display stream, use \text{WFROMDS}: 
(WFROMDS DISPLAYSTREAM DONTCREATE) [Function]
Retuns the window associated with DISPLAYSTREAM, creating a
window if one does not exist (and DONTCREATE is NIL). Returns
NIL if the destination of DISPLAYSTREAM is not a screen bitmap
that supports a window system.

If DONTCREATE is non-NIL, WFROMDS will never create a
window, and returns NIL if DISPLAYSTREAM does not have an
associated window.

TTYDISPLAYSTREAM calls WFROMDS with DONTCREATE = T, so
it will not create a window unnecessarily. Also, if WFROMDS
does create a window, it calls CREATEW with NOOPENFLG = T.

(DSPBACKUP WIDTH DISPLAYSTREAM) [Function]
Backs up DISPLAYSTREAM over a character which is WIDTH
screen points wide. DSPBACKUP fills the backed over area with
the display stream's background texture and decreases the X
position by WIDTH. If this would put the X position less than
DISPLAYSTREAM's left margin, its operation is stopped at the left
margin. IT returns T if any bits were written, NIL otherwise.

27.12 Fonts

A font is the collection of images that are printed or displayed
when characters are output to a graphic output device. Some
simple displays and printers can only print characters using one
font. Bitmap displays and graphic printers can print characters
using a large number of fonts.

Fonts are identified by a distinctive style or family (such as
Modern or Classic), a size (such as 10 points), and a face (such as
bold or italic). Fonts also have a rotation that indicates the
orientation of characters on the screen or page. A normal
horizontal font (also called a portrait font) has a rotation of 0;
the rotation of a vertical (landscape) font is 90 degrees. While
any combination can be specified, in practice the user will find
that only certain combinations of families, sizes, faces, and
rotations are available for any graphic output device.

To specify a font to the functions described below, a FAMILY is
represented by a literal atom, a SIZE by a positive integer, and a
FACE by a three-element list of the form (WEIGHT SLOPE
EXPANSION). WEIGHT, which indicates the thickness of the
characters, can be BOLD, MEDIUM, or LIGHT; SLOPE can be
ITALIC or REGULAR; and EXPANSION can be REGULAR,
COMPRESSED, or EXPANDED, indicating how spread out the
characters are. For convenience, faces may also be specified by
three-character atoms, where each character is the first letter of the corresponding field. Thus, MRR is a synonym for (MEDIUM REGULAR REGULAR). In addition, certain common face combinations may be indicated by special literal atoms:

STANDARD = (MEDIUM REGULAR REGULAR) = MRR
ITALIC = (MEDIUM ITALIC REGULAR) = MIR
BOLD = (BOLD REGULAR REGULAR) = BRR
BOLDITALIC = (BOLD ITALIC REGULAR) = BIR

Interlisp represents all the information related to a font in an object called a font descriptor. Font descriptors contain the family, size, etc. properties used to represent the font. In addition, for each character in the font, the font descriptor contains width information for the character and (for display fonts) a bitmap containing the picture of the character.

The font functions can take fonts specified in a variety of different ways. DSPFONT, FONTCREATE, FONTCOPY, etc. can be applied to font descriptors, "font lists" such as '(MODERN 10), image streams (coerced to its current font), or windows (coerced to the current font of its display stream). The printout command ".FONT" (page 25.27) will also accept fonts specified in any of these forms.

(FONTCREATE FAMILY SIZE FACE ROTATION DEVICE NOERRORFLG CHARSET)  [Function]

Returns a font descriptor for the specified font. FAMILY is a litatom specifying the font family. SIZE is an integer indicating the size of the font in points. FACE specifies the face characteristics in one of the formats listed above; if FACE is NIL, STANDARD is used. ROTATION, which specifies the orientation of the font, is 0 (or NIL) for a portrait font and 90 for a landscape font. DEVICE indicates the output device for the font, and can be any image stream type (page 27.8), such as DISPLAY, INTERPRESS, etc. DEVICE may also be an image stream, in which case the type of the stream determines the font device. DEVICE defaults to DISPLAY.

The FAMILY argument to FONTCREATE may also be a list, in which case it is interpreted as a font-specification quintuple, a list of the form (FAMILY SIZE FACE ROTATION DEVICE). Thus, (FONTCREATE '(GACHA 10 BOLD)) is equivalent to (FONTCREATE 'GACHA 10 'BOLD). FAMILY may also be a font descriptor, in which case that descriptor is simply returned.

If a font descriptor has already been created for the specified font, FONTCREATE simply returns it. If it has not been created, FONTCREATE has to read the font information from a font file that contains the information for that font. The name of an appropriate font file, and the algorithm for searching depends on the device that the font is for, and is described in more detail
below. If an appropriate font file is found, it is read into a font descriptor. If no file is found, for DISPLAY fonts FONT CREATE looks for fonts with less face information and fakes the remaining faces (such as by doubling the bit pattern of each character or slanting it). For hardcopy printer fonts, there is no acceptable faking algorithm.

If no acceptable font is found, the action of FONT CREATE is determined by NOERRORFLG. If NOERRORFLG is NIL, it generates a FONT NOT FOUND error with the offending font specification; otherwise, FONT CREATE returns NIL.

CHARSET is the character set which will be read to create the font. Defaults to 0. For more information on character sets, see NS Characters, page 2.12.

(FONTP X) [Function]

Returns X if X is a font descriptor; NIL otherwise.

(FONTPROP FONT PROP) [Function]

Returns the value of the PROP property of font FONT. The following font properties are recognized:

FAMILY The style of the font, represented as a literal atom, such as CLASSIC or MODERN.

SIZE A positive integer giving the size of the font, in printer's points (1/72 of an inch).

WEIGHT The thickness of the characters; one of BOLD, MEDIUM, or LIGHT.

SLOPE The "slope" of the characters in the font; one of ITALIC or REGULAR.

EXPANSION The extent to which the characters in the font are spread out; one of REGULAR, COMPRESSED, or EXPANDED. Most available fonts have EXPANSION = REGULAR.

FACE A three-element list of the form (WEIGHT SLOPE EXPANSION), giving all of the typeface parameters.

ROTATION An integer that gives the orientation of the font characters on the screen or page, in degrees. A normal horizontal font (also called a portrait font) has a rotation of 0; the rotation of a vertical (landscape) font is 90.

DEVICE The device that the font can be printed on; one of DISPLAY, INTERPRESS, etc.

ASCENT An integer giving the maximum height of any character in the font from its base line (the printing position). The top line will be at BASELINE + ASCENT-1.

DESCENT An integer giving the maximum extent of any character below the base line, such as the lower part of a "p". The bottom line of a character will be at BASELINE-DESCENT.
HEIGHT  Equal to ASCENT + DESCENT.

SPEC   The (FAMILY SIZE FACE ROTATION DEVICE) quintuple by which the font is known to Lisp.

DEVICESPEC The (FAMILY SIZE FACE ROTATION DEVICE) quintuple that identifies what will be used to represent the font on the display or printer. It will differ from the SPEC property only if an implicit coercion is done to approximate the specified font with one that actually exists on the device.

SCALE  The units per printer's point (1/72 of an inch) in which the font is measured. For example, this is 35.27778 (the number of micas per printer's point) for Interpress fonts, which are measured in terms of micas.

(FONTCOPY OLDFONT PROP₁ VAL₁ PROP₂ VAL₂ ...)  [NoSpread Function]
Returns a font descriptor that is a copy of the font OLDFONT, but which differs from OLDFONT in that OLDFONT’s properties are replaced by the specified properties and values. Thus, (FONTCOPY FONT 'WEIGHT 'BOLD 'DEVICE 'INTERPRESS) will return a bold Interpress font with all other properties the same as those of FONT. FONTCOPY accepts the properties FAMILY, SIZE, WEIGHT, SLOPE, EXPANSION, FACE, ROTATION, and DEVICE. If the first property is a list, it is taken to be the PROP₁ VAL₁ PROP₂ VAL₂ ... sequence. Thus, (FONTCOPY FONT '(WEIGHT BOLD DEVICE INTERPRESS)) is equivalent to the example above.
If the property NOERROR is specified with value non-NIL, FONTCOPY will return NIL rather than causing an error if the specified font cannot be created.

(FONTAVAILABLE FAMILY SIZE FACE ROTATION DEVICE CHECKFILESTOO?)  [Function]
Returns a list of available fonts that match the given specification. FAMILY, SIZE, FACE, ROTATION, and DEVICE are the same as for FONTCREATE. Additionally, any of them can be the atom *, in which case all values of that field are matched.
If CHECKFILESTOO? is NIL, only fonts already loaded into virtual memory will be considered. If CHECKFILESTOO? is non-NIL, the font directories for the specified device will be searched. When checking font files, the ROTATION is ignored.
Note: The search is conditional on the status of the server which holds the font. Thus a file server crash may prevent FONTCREATE from finding a file that an earlier FONTAVAILABLE returned.
Each element of the list returned will be of the form (FAMILY SIZE FACE ROTATION DEVICE).
Examples:
(FONTAVAILABLE 'MODERN 10 'MRR 0 'DISPLAY)
will return «(MODERN 10 (MEDIUM REGULAR REGULAR) 0 DISPLAY)) if the regular Modern 10 font for the display is in virtual memory; NIL otherwise.

(FONTAVAILABLE '14 ' 'INTERPRESS T)
will return a list of all the size 14 Interpress fonts, whether they are in virtual memory or in font files.

Warning: One must be careful when using the function FONTAVAILABLE to determine what Press font files are available. For Press font families/faces, the font widths for different sizes are consistently scaled versions of the smallest font in the family/face. Therefore, instead of storing data about all of the sizes in the FONTS.WIDTHS file, only the widths for the font of SIZE = 1 are stored, and the other widths are calculated by scaling these widths up. This is signified in the FONTS.WIDTHS file by a font with SIZE=0. Therefore, if FONTAVAILABLE is called with CHECKFILESTOO? = T, and it finds such a "relative" font, it returns a font spec list with size of 0. For example,

←(FONTAVAILABLE 'GACHA ' ' 0 'PRESS T)
((GACHA 0 (BOLD ITALIC REGULAR) 0 PRESS)
 (GACHA 0 (BOLD REGULAR REGULAR) 0 PRESS)
 (GACHA 0 (MEDIUM ITALIC REGULAR) 0 PRESS)
 (GACHA 0 (MEDIUM REGULAR REGULAR) 0 PRESS))

This indicates that Press files can be created with GACHA files of any size with faces BIR, BRR, MIR, and MRR. Of course, this doesn't guarantee that these fonts are available in all sizes on your printer.

(SETFONTDESCRIPTOR FAMILY SIZE FACE ROTATION DEVICE FONT) [Function]
Indicates to the system that FONT is the font that should be associated with the FAMILY SIZE FACE ROTATION DEVICE characteristics. If FONT is NIL, the font associated with these characteristics is cleared and will be recreated the next time it is needed. As with FONTPROP and FONTCOPY, FONT is coerced to a font descriptor if it is not one already.

This function is useful when it is desirable to simulate an unavailable font or to use a font with characteristics different from the interpretations provided by the system.

(DEFAULTFONT DEVICE FONT —) [Function]
Returns the font that would be used as the default (if NIL were specified as a font argument) for image stream type DEVICE. If FONT is a font descriptor, it is set to be the default font for DEVICE.

GRAPHICS OUTPUT OPERATIONS 27.29
(CHARWIDTH CHARCODE FONT) [Function]
CHARCODE is an integer that represents a valid character (as returned by CHCON1). Returns the amount by which an image stream's X-position will be incremented when the character is printed.

(CHARWIDTHY CHARCODE FONT) [Function]
Like CHARWIDTH, but returns the Y component of the character's width, the amount by which an image stream's Y-position will be incremented when the character is printed. This will be zero for most characters in normal portrait fonts, but may be non-zero for landscape fonts or for vector-drawing fonts.

(STRINGWIDTH STR FONT FLG RDTBL) [Function]
Returns the amount by which a stream's X-position will be incremented if the printname for the Interlisp-D object STR is printed in font FONT. If FONT is an image stream, its font is used. If FLG is non-NIL, the PRIN2-pname of STR with respect to the readtable RDTBL is used.

(STRINGREGION STR STREAM PRIN2FLG RDTBL) [Function]
Returns the region occupied by STR if it were printed at the current location in the image stream STREAM. This is useful, for example, for determining where text is in a window to allow the user to select it. The arguments PRIN2FLG and RDTBL are passed to STRINGWIDTH.
Note: STRINGREGION does not take into account any carriage returns in the string, or carriage returns that may be automatically printed if STR is printed to STREAM. Therefore, the value returned is meaningless for multi-line strings.

The following functions allow the user to access and change the bitmaps for individual characters in a display font. Note: Character code 256 can be used to access the "dummy" character, used for characters in the font with no bitmap defined.

(GETCHARBITMAP CHARCODE FONT) [Function]
Returns a bitmap containing a copy of the image of the character CHARCODE in the font FONT.

(PUTCHARBITMAP CHARCODE FONT NEWCHARBITMAP NEWCHARDESCENT) [Function]
Changes the bitmap image of the character CHARCODE in the font FONT to the bitmap NEWCHARBITMAP. If NEWCHARDESCENT is non-NIL, the descent of the character is changed to the value of NEWCHARDESCENT.
(EDITCHAR CHARCODE FONT) [Function]
Calls the bitmap editor (EDITBM, page 27.4) on the bitmap image of the character CHARCODE in the font FONT. CHARCODE can be a character code (as returned by CHCON1) or an atom or string, in which case the first character of CHARCODE is used.

27.13 Font Files and Font Directories

If FONTCREATE is called to create a font that has not been loaded into Interlisp, FONTCREATE has to read the font information from a font file that contains the information for that font. For printer devices, the font files have to contain width information for each character in the font. For display fonts, the font files have to contain, in addition, bitmap images for each character in the fonts. The font file names, formats, and searching algorithms are different for each device. There are a set of variables for each device, that determine the directories that are searched for font files. All of these variables must be set before Interlisp can auto-load font files. These variables should be initialized in the site-specific INIT file.

DISPLAYFONTDIRECTORIES [Variable]
Value is a list of directories searched to find font bitmap files for display fonts.

DISPLAYFONTEXTENSIONS [Variable]
Value is a list of file extensions used when searching DISPLAYFONTDIRECTORIES for display fonts. Initially set to (DISPLAYFONT), but when using older font files it may be necessary to add STRIKE and AC to this list.

INTERPRESSFONTDIRECTORIES [Variable]
Value is a list of directories searched to find font widths files for Interpress fonts.

PRESSFONTWIDTHSFILES [Variable]
Value is a list of files (not directories) searched to find font widths files for Press fonts. Press font widths are packed into large files (usually named FONTS.WIDTHS).
27.15 Font Profiles

PRETTYPRINT contains a facility for printing different elements (user functions, system functions, clisp words, comments, etc.) in different fonts to emphasize (or deemphasize) their importance, and in general to provide for a more pleasing appearance. Of course, in order to be useful, this facility requires that the user is printing on a device (such as a bitmapped display or a laser printer) which supports multiple fonts.

PRETTYPRINT signals font changes by inserting into the file a user-defined escape sequence (the value of the variable FONTESCAPECHAR) followed by the character code which specifies, by number, which font to use, i.e. ↑A for font number 1, etc. Thus, if FONTESCAPECHAR were the character ↑F, ↑F ↑C would be output to change to font 3, ↑F ↑A to change to font 1, etc. If FONTESCAPECHAR consists of characters which are separator characters in FILERDTBL, then a file with font changes in it can also be loaded back in.

Currently, PRETTYPRINT uses the following font classes. The user can specify separate fonts for each of these classes, or use the same font for several different classes.

LAMBDAFONT The font for printing the name of the function being prettyprinted, before the actual definition (usually a large font).

CLISPFONT If CLISPFLEG is on, the font for printing any clisp words, i.e. atoms with property CLISWORD.

COMMENTFONT The font used for comments.

USERFONT The font for the name of any function in the file, or any member of the list FONTFNS.

SYSTEMFONT The font for any other (defined) function.

CHANGEFONT The font for an expression marked by the editor as having been changed.

PRETTYCOMFONT The font for the operand of a file package command.

DEFAULTFONT The font for everything else.

Note that not all combinations of fonts will be aesthetically pleasing (or even readable!) and the user may have to experiment to find a compatible set.

Although in some implementations LAMBDAFONT et al. may be defined as variables, one should not set them directly, but should indicate what font is to be used for each class by calling the function FONTPROFILE:

\[(\text{FONTPROFILE} \ PROFILE)\]  

[Function]  

Sets up the font classes as determined by PROFILE, a list of elements which defines the correspondence between font
classes and specific fonts. Each element of `PROFILE` is a list of the form:

```lisp
(FONTCLASS FONT# DISPLAYFONT PRESSFONT INTERPRESSFONT)
```

`FONTCLASS` is the font class name and `FONT#` is the font number for that class. For each font class name, the escape sequence will consist of `FONTESCAPECHAR` followed by the character code for the font number, e.g. `↑A` for font number 1, etc.

If `FONT#` is `NIL` for any font class, the font class named `DEFAULTFONT` (which must always be specified) is used. Alternatively, if `FONT#` is the name of a previously defined font class, this font class will be equivalenced to the previously defined one.

`DISPLAYFONT`, `PRESSFONT`, and `INTERPRESSFONT` are font specifications (of the form accepted by `FONTCREATE`) for the fonts to use when printing to the display and to Press and Interpress printers respectively.

`FONTPROFILE` [Variable]

This is the variable used to store the current font profile, in the form accepted by the function `FONTPROFILE`. Note that simply editing this value will not change the fonts used for the various font classes; it is necessary to execute `(FONTPROFILE PROFILE)` to install the value of this variable.

The process of printing with multiple fonts is affected by a large number of variables: `FONTPROFILE`, `FILELINELENGTH`, `PRETTYLCOM`, etc. To facilitate switching back and forth between various sets of values for the font variables, Interlisp supports the idea of named "font configurations" encapsulating the values of all relevant variables.

To create a new font configuration, set all "relevant" variables to the values you want, and then call `FONTNAME` to save them (on the variable `FONTDIFS`) under a given name. To install a particular font configuration, call `FONTSET` giving it your name. To change the values in a saved font configuration, edit the value of the variable `FONTDIFS`.

Note: The list of variables saved by `FONTNAME` is stored in the variable `FONTDIFSVARs`. This can be changed by the user.

`FONTNAME NAME` [Function]

Collects the names and values of the variables on `FONTDIFSVARs`, and saves them on `FONTDIFS`. 
(FONTSET NAME) [Function]
Installs font configuration for NAME. Also evaluates (FONTPROFILE FONTPROFILE) to install the font classes as specified in the new value of the variable FONTPROFILE. Generates an error if NAME not previously defined.

FONTDEFSVARS [Variable]
The list of variables to be packaged by a FONTNAME. Initially FONTCHANGEFLG, FILELINELENGTH, COMMENTLINELENGTH, FIRSTCOL, PRETTYLCOM, LISTFILESTR, and FONTPROFILE.

FONTDEFS [Variable]
An association list of font configurations. FONTDEFS is a list of elements of form (NAME . PARAMETER-PAIRS). To save a configuration on a file after performing a FONTNAME to define it, the user could either save the entire value of FONTDEFS, or use the ALISTS file package command (page 17.37) to dump out just the one configuration.

FONTESCAPECHAR [Variable]
The character or string used to signal the start of a font escape sequence.

FONTCHANGEFLG [Variable]
If T, enables fonts when prettyprinting. If NIL, disables fonts.

LISTFILESTR [Variable]
In Interlisp-10, passed to the operating system by LISTFILES (page 17.14). Can be used to specify subcommands to the LIST command, e.g. to establish correspondance between font number and font name.

COMMENTLINELENGTH [Variable]
Since comments are usually printed in a smaller font, COMMENTLINELENGTH is provided to offset the fact that Interlisp does not know about font widths. When FONTCHANGEFLG = T, CAR of COMMENTLINELENGTH is the linelength used to print short comments, i.e. those printed in the right margin, and CDR is the linelength used when printing full width comments.

(CHANGEFONT FONT STREAM) [Function]
Executes the operations on STREAM to change to the font FONT. For use in PRETTYPRINTMACROS.
27.16 Image Objects

An Image Object is an object that includes information about an image, such as how to display it, how to print it, and how to manipulate it when it is included in a collection of images (such as a document). More generally, it enables you to include one kind of image, with its own semantics, layout rules, and editing paradigms, inside another kind of image. Image Objects provide a general-purpose interface between image users who want to manipulate arbitrary images, and image producers, who create images for use, say, in documents.

Images are encapsulated inside a uniform barrier—the IMAGEOBJ data type. From the outside, you communicate to the image by calling a standard set of functions. For example, calling one function tells you how big the image is; calling another causes the image object to be displayed where you tell it, and so on. Anyone who wants to create images for general use can implement his own brand of IMAGEOBJ. IMAGEOBJs have been implemented (in library packages) for bitmaps, menus, annotations, graphs, and sketches.

Image Objects were originally implemented to support inserting images into TEdit text files, but the facility is available for use by any tools that manipulate images. The Image Object interface allows objects to exist in TEdit documents and be edited with their own editor. It also provides a facility in which objects can be shift-selected (or "copy-selected") between TEdit and non-TEdit windows. For example, the Image Objects interface allows you to copy-select graphs from a Grapher window into a TEdit window. The source window (where the object comes from) does not have to know what sort of window the destination window (where the object is inserted) is, and the destination does not have to know where the insertion comes from.

A new data type, IMAGEOBJ, contains the data and the procedures necessary to manipulate an object that is to be manipulated in this way. IMAGEOBJs are created with the function IMAGEOBJCREATE (below).

Another new data type, IMAGEFNS, is a vector of the procedures necessary to define the behavior of a type of IMAGEOBJ. Grouping the operations in a separate data type allows multiple instances of the same type of image object to share procedure vectors. The data and procedure fields of an IMAGEOBJ have a uniform interface through the function IMAGEOBJPROP. IMAGEFNS are created with the function IMAGEFNSCREATE:
IMAGE OBJECTS

(IIMAGEFNSCREATE DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN BUTTONEVENTINFN COPYBUTTONEVENTINFN WHENMOVEDFN WHENINSERTEDFN WHENDELETEDFN WHENCOPIEDFN WHENOPERATEDONFN PREPRINTFN —)  

Returns an IMAGDEFNS object that contains the functions necessary to define the behavior of an IMAGEOBJ.

The arguments DISPLAYFN through PREPRINTFN should all be function names to be stored as the "methods" of the IMAGDEFNS. The purpose of each IMAGDEFNS method is described below.

Note: Image objects must be "registered" before they can be read by TEdit or HREAD (see page 27.39). IMAGDEFNSCREATE implicitly registers its GETFN argument.

(IMAGEOBJCREATE OBJECTDATUM IMAGDEFNS)  

Returns an IMAGEOBJ that contains the object datum OBJECTDATUM and the operations vector IMAGDEFNS. OBJECTDATUM can be arbitrary data.

(IMAGEOBJPROP IMAGEOBJECT PROPERTY NEWVALUE)  

Accesses and sets the properties of an IMAGEOBJ. Returns the current value of the PROPERTY property of the image object IMAGEOBJECT. If NEWVALUE is given, the property is set to it.

IMAGEOBJPROP can be used on the system properties OBJECTDATUM, DISPLAYFN, IMAGEBOXFN, PUTFN, GETFN, COPYFN, BUTTONEVENTINFN, COPYBUTTONEVENTINFN, WHENOPERATEDONFN, and PREPRINTFN. Additionally, it can be used to save arbitrary properties on an IMAGEOBJ.

(IIMAGEFNSP X)  

Returns X if X is an IMAGDEFNS object, NIL otherwise.

(IMAGEOBJP X)  

Returns X if X is an IMAGEOBJ object, NIL otherwise.

27.16.1 IMAGDEFNS Methods

Note: Many of the IMAGDEFNS methods below are passed "host stream" arguments. The TEdit text editor passes the "text stream" (an object contain all of the information in the document being edited) as the "host stream" argument. Other editing programs that want to use image objects may want to pass the data structure being edited to the IMAGDEFNS methods as the "host stream" argument.
**DISPLAYFN IMAGEOBJ IMAGESTREAM IMAGESTREAMTYPE HOSTSTREAM**

**IMAGEFNS Method**

The DISPLAYFN method is called to display the object IMAGEOBJ at the current position on IMAGESTREAM. The type of IMAGESTREAM indicates whether the device is the display or some other image stream.

Note: When the DISPLAYFN method is called, the offset and clipping regions for the stream are set so the object's image is at (0,0), and only that image area can be modified.

**IMAGEBOXFN IMAGEOBJ IMAGESTREAM CURRENTX RIGHTMARGIN**

**IMAGEFNS Method**

The IMAGEBOXFN method should return the size of the object as an IMAGEBOX, which is a data structure that describes the image laid down when an IMAGEOBJ is displayed in terms of width, height, and descender height. An IMAGEBOX has four fields: XSIZE, YSIZE, YDESC, and XKERN. XSIZE and YSIZE are the width and height of the object image. YDESC and XKERN give the position of the baseline and the left edge of the image relative to where you want to position it. For characters, the YDESC is the descent (height of the descender) and the XKERN is the amount of left kerning (note: TEdit doesn't support left kerning).

The IMAGEBOXFN looks at the type of the stream to determine the output device if the object's size changes from device to device. (For example, a bit-map object may specify a scale factor that is ignored when the bit map is displayed on the screen.) CURRENTX and RIGHTMARGIN allow an object to take account of its environment when deciding how big it is. If these fields are not available, they are NIL.

Note: TEdit calls the IMAGEBOXFN only during line formatting, then caches the IMAGEBOX as the BOUNDBOX property of the IMAGEOBJ. This avoids the need to call the IMAGEBOXFN when incomplete position and margin information is available.

**PUTFN IMAGEOBJ FILESTREAM**

**IMAGEFNS Method**

The PUTFN method is called to save the object on a file. It prints a description on FILESTREAM that, when read by the corresponding GETFN method (see below), regenerates the image object. (TEdit and HPRINT take care of writing out the name of the GETFN.)

**GETFN FILESTREAM**

**IMAGEFNS Method**

The GETFN method is called when the object is encountered on the file during input. It reads the description that was written by the PUTFN method and returns an IMAGEOBJ.
The COPYFN method is called during a copy-select operation. It should return a copy of IMAGEOBJ. If it returns the litatom DON'T, copying is suppressed.

The BUTTONEVENTINFN method is called when you press a mouse button inside the object. The BUTTONEVENTINFN decides whether or not to handle the button, to track the cursor in parallel with mouse movement, and to invoke selections or edits supported by the object (but see the COPYBUTTONEVENTINFN method below). If the BUTTONEVENTINFN returns NIL, TEdit treats the button press as a selection at its level. Note that when this function is first called, a button is down. The BUTTONEVENTINFN should also support the button-down protocol to descend inside of any composite objects within it. In most cases, the BUTTONEVENTINFN relinquishes control (i.e., returns) when the cursor leaves its object’s region.

Note: When the BUTTONEVENTINFN is called, the window’s clipping region and offsets have been changed so that the lower-left corner of the object’s image is at (0.0), and only the object’s image can be changed. The selection is available for changing to fit your needs; the mouse button went down at (RELX, RELY) within the object’s image. You can affect how TEdit treats the selection by returning one of several values. If you return NIL, TEdit forgets that you selected an object; if you return the atom DON'T, TEdit doesn’t permit the selection; if you return the atom CHANGED, TEdit updates the screen. Use CHANGED to signal TEdit that the object has changed size or will have side effects on other parts of the screen image.

The COPYBUTTONEVENTINFN method is called when you button inside an object while holding down a copy key. Many of the comments about BUTTONEVENTINFN apply here too. Also, see the discussion below about copying image objects between windows (page 27.41).

The WHENMOVEDFN method provides hooks by which the object is notified when TEdit performs an operation (MOVEing) on the whole object. It allows objects to have side effects.
The WHENINSERTEDFN method provides hooks by which the object is notified when TEdit performs an operation (INSERTing) on the whole object. It allows objects to have side effects.

The WHENDELETEDFN method provides hooks by which the object is notified when TEdit performs an operation (DELETEing) on the whole object. It allows objects to have side effects.

The WHENCOPIEDFN method provides hooks by which the object is notified when TEdit performs an operation (COPYing) on the whole object. The WHENCOPIEDFN method is called in addition to (and after) the COPYFN method above. It allows objects to have side effects.

The WHENOPERATEDONFN method provides a hook for edit operations. HOWOPERATEDON should be one of SELECTED, DESELECTED, HIGHLIGHTED, and UNHILIGHTED. The WHENOPERATEDONFN differs from the BUTTONEVENTINFN because it is called when you extend a selection through the object. That is, the object is treated in toto as a TEdit character. HIGHLIGHTED refers to the selection being highlighted on the screen, and UNHIGHLIGHTED means that the highlighting is being turned off.

The PREPRINTFN method is called to convert the object into something that can be printed for inclusion in documents. It returns an object that the receiving window can print (using either PRIN1 or PRIN2, its choice) to obtain a character representation of the object. If the PREPRINTFN method is NIL, the OBJECTDATUM field of IMAGEOBJ itself is used. TEdit uses this function when you indicate that you want to print the characters from an object rather than the object itself (presumably using PRIN1 case).

27.16.2 Registering Image Objects

Each legitimate GETFN needs to be known to the system, to prevent various Trojan-horse problems and to allow the
automatic loading of the supporting code for infrequently used IMAGEOBJs. To this end, there is a global list, IMAGEOBJGETFNS, that contains an entry for each GETFN. The existence of the entry marks the GETFN as legitimate; the entry itself is a property list, which can hold information about the GETFN.

No action needs to be taken for GETFNs that are currently in use: the function IMAGEFN$CREATE automatically adds its GETFN argument to the list. However, packages that support obsolete versions of objects may need to explicitly add the obsolete GETFNs. For example, TEdit supports bit-map IMAGEOBJs. Recently, a change was made in the format in which objects are stored; to retain compatibility with the old object format, there are now two GETFNs. The current GETFN is automatically on the list, courtesy of IMAGEFNSCREATE. However, the code file that supports the old bit-map objects contains the clause: (ADDVARS (IMAGEOBJGETFNS (OLDGETFNNAME))), which adds the old GETFN to IMAGEOBJGETFNS.

For a given GETFN, the entry on IMAGEOBJGETFNS may be a property list of information. Currently the only recognized property is FILE.

FILE is the name of the file that can be loaded if the GETFN isn’t defined. This file should define the GETFN, along with all the other functions needed to support that kind of IMAGEOBJ.

For example, the bit-map IMAGEOBJ implemented by TEdit use the GETFN BMOBJ.GETFN2. Its entry on IMAGEOBJGETFNS is (BMOBJ.GETFN2 FILE IMAGEOBJ), indicating that the support code for bit-map image objects resides on the file IMAGEOBJ, and that the GETFN for them is BMOBJ.GETFN2.

This makes it possible to have entries for GETFNs whose supporting code isn’t loaded—you might, for instance, have your init file add entries to IMAGEOBJGETFNS for the kinds of image objects you commonly use. The system’s default reading method will automatically load the code when necessary.

27.16.3 Reading and Writing Image Objects on Files

Image Objects can be written out to files using HPRINT and read back using HREAD. The following functions can also be used:

(WRITEIMAGEOBJ IMAGEOBJ STREAM) [Function]
Prints (using PRIN2) a call to READIMAGEOBJ, then calls the PUTFN for IMAGEOBJ to write it onto STREAM. During input, then, the call to READIMAGEOBJ is read and evaluated; it in turn reads back the object’s description, using the appropriate GETFN.
(READIMAGEOBJ STREAM GETFN NOERROR) [Function]
Reads an IMAGEOBJ from STREAM, starting at the current file position. Uses the function GETFN after validating it (and loading support code, if necessary).

If the GETFN can't be validated or isn't defined, READIMAGEOBJ returns an "encapsulated image object", an IMAGEOBJ that safely encapsulates all of the information in the image object. An encapsulated image object displays as a rectangle that says, "Unknown IMAGEOBJ Type" and lists the GETFN's name. Selecting an encapsulated image object with the mouse causes another attempt to read the object from the file; this is so you can load any necessary support code and then get to the object.

Warning: You cannot save an encapsulated image object on a file because there isn't enough information to allow copying the description to the new file from the old one.

If NOERROR is non-NIL, READIMAGEOBJ returns NIL if it can't successfully read the object.

27.16.4 Copying Image Objects Between Windows

Copying between windows is implemented as follows: If a button event occurs in a window when a copy key is down, the window's COPYBUTTONEVENTFN window property is called. If this window supports copy-selection, it should track the mouse, indicating the item to be copied. When the button is released, the COPYBUTTONEVENTFN should create an image object out of the selected information, and call COPYINSERT to insert it in the current TTY window. COPYINSERT calls the COPYINSERTFN window property of the TTY window to insert this image object. Therefore, both the source and destination windows can determine how they handle copying image objects.

If the COPYBUTTONEVENTFN of a window is NIL, the BUTTONEVENTFN is called instead when a button event occurs in the window when a copy key is down, and copying from that window is not supported. If the COPYINSERTFN of the TTY window is NIL, COPYINSERT will turn the image object into a string (by calling the PREPRINTFN method of the image object, see page 27.39) and insert it by calling BKSYSBUF (page 30.11).

COPYBUTTONEVENTFN [Window Property]
The COPYBUTTONEVENTFN of a window is called (if it exists) when a button event occurs in the window and a copy key is down. If no COPYBUTTONEVENTFN exists, the BUTTONEVENTFN is called.
The COPYINSERTFN of the "destination" window is called by COPYINSERT to insert something into the destination window. It is called with two arguments: the object to be inserted and the destination window. The object to be inserted can be a character string, an IMAGEOBJ, or a list of IMAGEOBJs and character strings. As a convention, the COPYINSERTFN should call BKSYSBUF (page 30.11) if the object to be inserted insert is a character string.

COPYINSERT inserts IMAGEOBJ into the window that currently has the TTY. If the current TTY window has a COPYINSERTFN, it is called, passing it IMAGEOBJ and the window as arguments.

If no COPYINSERTFN exists and if IMAGEOBJ is an image object, BKSYSBUF is called on the result of calling its PREPRINTFN on it. If IMAGEOBJ is not an image object, it is simply passed to BKSYSBUF (page 30.11). In this case, BKSYSBUF will call PRIN2 with a read table taken from the process associated with the TTY window. A window that wishes to use PRIN1 or a different read table must provide its own COPYINSERTFN to do this.

27.17 Implementation of Image Streams

Interlisp does all image creation through a set of functions and data structures for device-independent graphics, known popularly as DIG. DIG is implemented through the use of a special type of stream, known as an image stream.

An image stream, by convention, is any stream that has its IMAGEOPS field (described in detail below) set to a vector of meaningful graphical operations. Using image streams, you can write programs that draw and print on an output stream without regard to the underlying device, be it a window, a disk, or a printer.

To define a new image stream type, it is necessary to put information on the variable IMAGESTREAMTYPES:

This variable describes how to create a stream for a given image stream type. The value of IMAGESTREAMTYPES is an association list, indexed by the image stream type (e.g., DISPLAY, INTERPRESS, etc.). The format of a single association list item is:

```
(IMAGETYPE
 (OPENSTREAM OPENSTREAMFN))
```
IMPLEMENTATION OF IMAGE STREAMS

(FONTCREATE FONTCREATEFN)
(FONTSAVAILABLE FONTSAVAILABLEFN))

OPENSTREAMFN, FONTCREATEFN, and FONTSAVAILABLEFN are "image stream methods," device-dependent functions used to implement generic image stream operations. For Interpress image streams, the association list entry is:

(INTERPRESS
 (OPENSTREAM OPENIPSTREAM)
 (FONTCREATE CREATEINTERPRESSFONT)
 (FONTSAVAILABLE SEARCHINTERPRESSFONTS))

(OPENSTREAMFN FILE OPTIONS) [Image Stream Method]

FILE is the file name as it was passed to OPENIMAGESTREAM, and OPTIONS is the OPTIONS property list passed to OPENIMAGESTREAM. The result must be a stream of the appropriate image type.

(FONTCREATEFN FAMILY SIZE FACE ROTATION DEVICE) [Image Stream Method]

FAMILY is the family name for the font, e.g., MODERN. SIZE is the body size of the font, in printer's points. FACE is a three-element list describing the weight, slope, and expansion of the face desired, e.g., (MEDIUM ITALIC EXPANDED). ROTATION is how much the font is to be rotated from the normal orientation, in minutes of arc. For example, to print a landscape page, fonts have the rotation 5400 (90 degrees). The function's result must be a FONTDESCRIPTOR with the fields filled in appropriately.

(FONTSAVAILABLEFN FAMILY SIZE FACE ROTATION DEVICE) [Image Stream Method]

This function returns a list of all fonts agreeing with the FAMILY, SIZE, FACE, and ROTATION arguments; any of them may be wild-carded (i.e., equal to *, which means any value is acceptable). Each element of the list should be a quintuple of the form (FAMILY SIZE FACE ROTATION DEVICE).

Where the function looks is an implementation decision: the FONTSAVAILABLEFN for the display device looks at DISPLAYFONTDIRECTORIES, the Interpress code looks on INTERPRESSFONTDIRECTORIES, and implementors of new devices should feel free to introduce new search path variables.

As indicated above, image streams use a field that no other stream uses: IMAGEOPS. IMAGEOPS is an instance of the IMAGEOPS data type and contains a vector of the stream's graphical methods. The methods contained in the IMAGEOPS object can make arbitrary use of the stream's IMAGEDATA field,
which is provided for their use, and may contain any data needed.

The **IMAGEOPS** data type has the following fields:

<table>
<thead>
<tr>
<th>IMAGETYPE</th>
<th>[IMAGEOPS Field]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>is the name of an image type. Monochrome display streams have an IMAGETYPE of DISPLAY; color display streams are identified as (COLOR DISPLAY). The IMAGETYPE field is informational and can be set to anything you choose.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMFONTCREATE</th>
<th>[IMAGEOPS Field]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>is the device name to pass to FONTCREATE when fonts are created for the stream.</td>
</tr>
</tbody>
</table>

The remaining fields are all image stream methods, whose value should be a device-dependent function that implements the generic operation. Most methods are called by a similarly-named function, e.g. the function DRAWLINE calls the IMDRAWLINE method. All coordinates that refer to points in a display device’s space are measured in the device’s units. (The IMSCALE method provides access to a device’s scale.) For arguments that have defaults (such as the BRUSH argument of DRAWCURVE), the default is substituted for the NIL argument before it is passed to the image stream method. Therefore, image stream methods do not have to handle defaults.

<table>
<thead>
<tr>
<th>(IMCLOSEFN STREAM)</th>
<th>[Image Stream Method]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called before a stream is closed with CLOSEF. This method should flush buffers, write header or trailer information, etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(IMDRAWLINE STREAM X1 Y1 X2 Y2 WIDTH OPERATION COLOR DASHING)</th>
<th>[Image Stream Method]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draws a line of width WIDTH from (X1, Y1) to (X2, Y2). See DRAWLINE, page 27.17.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(IMDRAWCURVE STREAM KNOTS CLOSED BRUSH DASHING)</th>
<th>[Image Stream Method]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draws a curve through KNOTS. See DRAWCURVE, page 27.19.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(IMDRAWCIRCLE STREAM CENTERX CENTERY RADIUS BRUSH DASHING)</th>
<th>[Image Stream Method]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draws a circle of radius RADIUS around (CENTERX, CENTERY). See DRAWCIRCLE, page 27.19.</td>
<td></td>
</tr>
</tbody>
</table>
(IMDRAWELLIPSE STREAM CENTERX CENTERY SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING) [Image Stream Method]
Draws an ellipse around (CENTERX, CENTERY). See DRAWELLIPSE, page 27.19.

(IMFILLPOLYGON STREAM POINTS TEXTURE) [Image Stream Method]
Fills in the polygon outlined by POINTS on the image stream STREAM, using the texture TEXTURE. See FILLPOLYGON, page 27.20.

(IMFILLCIRCLE STREAM CENTERX CENTERY RADIUS TEXTURE) [Image Stream Method]
Draws a circle filled with texture TEXTURE around (CENTERX, CENTERY). See FILLCIRCLE, page 27.21.

(IMBLTSHADE TEXTURE STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT OPERATION CLIPPINGREGION) [Image Stream Method]
The texture-source case of BITBLT (page 27.14). DESTINATIONLEFT, DESTINATIONBOTTOM, WIDTH, HEIGHT, and CLIPPINGREGION are measured in STREAM's units. This method is invoked by the functions BITBLT and BLTSHADE (page 27.16).

(IMBITBLT SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE) [Image Stream Method]
Contains the bit-map-source cases of BITBLT (page 27.14). SOURCELEFT, SOURCEBOTTOM, CLIPPEDSOURCELEFT, CLIPPEDSOURCEBOTTOM, WIDTH, and HEIGHT are measured in pixels; DESTINATIONLEFT, DESTINATIONBOTTOM, and CLIPPINGREGION are in the units of the destination stream.

(IMSCALEDBITBLT SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE) [Image Stream Method]
A scaled version of IMBITBLT. Each pixel in SOURCEBITMAP is replicated SCALE times in the X and Y directions; currently, SCALE must be an integer.

(IMMOVETO STREAM X Y) [Image Stream Method]
Moves to (X, Y). This method is invoked by the function MOVETO (page 27.13). If IMMOVETO is not supplied, a default method composed of calls to the IMXPOSITION and IMYPOSITION methods is used.
IMPLEMENTATION OF IMAGE STREAMS

(IMSTRINGWIDTH STREAM STR RDTHL) [Image Stream Method]
Returns the width of string STR in STREAM's units, using
STREAM's current font. This is invoked when STRINGWIDTH
(page 27.30) is passed a stream as its FONT argument. If
IMSTRINGWIDTH is not supplied, it defaults to calling
STRINGWIDTH on the default font of STREAM.

(IMCHARWIDTH STREAM CHARCODE) [Image Stream Method]
Returns the width of character CHARCODE in STREAM's units,
using STREAM's current font. This is invoked when CHARWIDTH
(page 27.30) is passed a stream as its FONT argument. If
IMCHARWIDTH is not supplied, it defaults to calling CHARWIDTH
on the default font of STREAM.

(IMCHARWIDTHY STREAM CHARCODE) [Image Stream Method]
Returns the Y component of the width of character CHARCODE
in STREAM's units, using STREAM's current font. This is invoked
when CHARWIDTH (page 27.30) is passed a stream as its FONT
argument. If IMCHARWIDTHY is not supplied, it defaults to calling
CHARWIDTH on the default font of STREAM.

(IMBITMAPSIZE STREAM BITMAP DIMENSION) [Image Stream Method]
Returns the size that BITMAP will be when BITBLTed to STREAM,
in STREAM's units. DIMENSION can be one of WIDTH, HEIGHT, or
NIL, in which case the dotted pair (WIDTH . HEIGHT) will be
returned.
This is invoked by BITMAPIMAGESIZE (page 27.16). If
IMBITMAPSIZE is not supplied, it defaults to a method that
multiplies the bitmap height and width by the scale of STREAM.

(IMNEWPAGE STREAM) [Image Stream Method]
Causes a new page to be started. The X position is set to the left
margin, and the Y position is set to the top margin plus the
linefeed. If not supplied, defaults to (OUTCHAR STREAM
(CHARCODE ↑ L)). Envoked by DSPNEWPAGE (page 27.21).

(IMTERPRI STREAM) [Image Stream Method]
Causes a new line to be started. The X position is set to the left
margin, and the Y position is set to the current Y position plus
the linefeed. If not supplied, defaults to (OUTCHAR STREAM
(CHARCODE EOL)). Envoked by TERPRI (page 25.9).

(IMRESET STREAM) [Image Stream Method]
Resets the X and Y position of STREAM. The X coordinate is set
to its left margin; the Y coordinate is set to the top of the
IMPLEMENTATION OF IMAGE STREAMS

clipping region minus the font ascent. Envoked by DSPRESET, page 27.21.

The following methods all have corresponding DSPxx functions (e.g., IMYPOSITION corresponds to DSPYPOSITION) that invoke them. They also have the property of returning their previous value; when called with NIL they return the old value without changing it.

(IMCLIPPINGREGION STREAM REGION) [Image Stream Method]
Sets a new clipping region on STREAM.

(IMXPOSITION STREAM XPOSITION) [Image Stream Method]
Sets the X-position on STREAM.

(IMYPOSITION STREAM YPOSITION) [Image Stream Method]
Sets a new Y-position on STREAM.

(IMFONT STREAM FONT) [Image Stream Method]
Sets STREAM's font to be FONT.

(IMLEFTMARGIN STREAM LEFTMARGIN) [Image Stream Method]
Sets STREAM's left margin to be LEFTMARGIN. The left margin is defined as the X-position set after the new line.

(IMRIGHTMARGIN STREAM RIGHTMARGIN) [Image Stream Method]
Sets STREAM's right margin to be RIGHTMARGIN. The right margin is defined as the maximum X-position at which characters are printed; printing beyond it causes a new line.

(IMTOPMARGIN STREAM YPOSITION) [Image Stream Method]
Sets STREAM's top margin (the Y-position of the tops of characters that is set after a new page) to be YPOSITION.

(IMBOTTOMMARGIN STREAM YPOSITION) [Image Stream Method]
Sets STREAM's bottom margin (the Y-position beyond which any printing causes a new page) to be YPOSITION.

(IMLINEFEED STREAM DELTA) [Image Stream Method]
Sets STREAM's line feed distance (distance to move vertically after a new line) to be DELTA.
IMPLEMENTATION OF IMAGE STREAMS

(IMSCALE STREAM SCALE) [Image Stream Method]
Returns the number of device points per screen point (a screen point being \(1/72\) inch). SCALE is ignored.

(IMSPACEFACTOR STREAM FACTOR) [Image Stream Method]
Sets the amount by which to multiply the natural width of all following space characters on STREAM; this can be used for the justification of text. The default value is 1. For example, if the natural width of a space in STREAM's current font is 12 units, and the space factor is set to two, spaces appear 24 units wide. The values returned by STRINGWIDTH and CHARWIDTH are also affected.

(IMOPERATION STREAM OPERATION) [Image Stream Method]
Sets the default BITBLT OPERATION argument (see page 27.15).

(INBACKCOLOR STREAM COLOR) [Image Stream Method]
Sets the background color of STREAM.

(INCOLOR STREAM COLOR) [Image Stream Method]
Sets the default color of STREAM.

In addition to the IMAGEOPS methods described above, there are two other important methods, which are contained in the stream itself. These fields can be installed using a form like (replace (STREAM OUTCHARFN) of STREAM with (FUNCTION MYOUTCHARFN)). Note: You need to have loaded the Interlisp-D system declarations to manipulate the fields of STREAMs. The declarations can be loaded by loading the Lisp Library package SYSEDIT.

(STRMBOUTFN STREAM CHARCODE) [Stream Method]
The function called by BOUT.

(OUTCHARFN STREAM CHARCODE) [Stream Method]
The function that is called to output a single byte. This is like STRMBOUTFN, except for being one level higher: it is intended for text output. Hence, this function should convert (CHARCODE EOL) into the stream’s actual end-of-line sequence and should adjust the stream’s CHARPOSITION appropriately before invoking the stream's STRMBOUTFN (by calling BOUT) to actually put the character. Defaults to \FILEOUTCHARFN, which is probably incorrect for an image stream.
28. Windows and Menus

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.2. Changing Window Command Menus</td>
<td>28.7</td>
</tr>
<tr>
<td>28.3. Interactive Display Functions</td>
<td>28.9</td>
</tr>
<tr>
<td>28.4.1. Window Properties</td>
<td>28.13</td>
</tr>
<tr>
<td>28.4.2. Creating Windows</td>
<td>28.13</td>
</tr>
<tr>
<td>28.4.3. Opening and Closing Windows</td>
<td>28.15</td>
</tr>
<tr>
<td>28.4.4. Redisplaying Windows</td>
<td>28.16</td>
</tr>
<tr>
<td>28.4.5. Reshaping Windows</td>
<td>28.16</td>
</tr>
<tr>
<td>28.4.6. Moving Windows</td>
<td>28.19</td>
</tr>
<tr>
<td>28.4.7. Exposing and Burying Windows</td>
<td>28.20</td>
</tr>
<tr>
<td>28.4.8. Shrinking Windows Into Icons</td>
<td>28.21</td>
</tr>
<tr>
<td>28.4.9. Coordinate Systems, Extents, And Scrolling</td>
<td>28.23</td>
</tr>
<tr>
<td>28.4.10. Mouse Activity in Windows</td>
<td>28.27</td>
</tr>
<tr>
<td>28.4.11. Terminal I/O and Page Holding</td>
<td>28.29</td>
</tr>
<tr>
<td>28.4.12. The TTY Process and the Caret</td>
<td>28.30</td>
</tr>
<tr>
<td>28.4.13. Miscellaneous Window Functions</td>
<td>28.31</td>
</tr>
<tr>
<td>28.4.15. Example: A Scrollable Window</td>
<td>28.34</td>
</tr>
<tr>
<td>28.5. Menus</td>
<td>28.37</td>
</tr>
<tr>
<td>28.5.1. Menu Fields</td>
<td>28.38</td>
</tr>
<tr>
<td>28.5.2. Miscellaneous Menu Functions</td>
<td>28.42</td>
</tr>
<tr>
<td>28.5.3. Examples of Menu Use</td>
<td>28.43</td>
</tr>
<tr>
<td>28.6. Attached Windows</td>
<td>28.45</td>
</tr>
<tr>
<td>28.6.1. Attaching Menus To Windows</td>
<td>28.48</td>
</tr>
<tr>
<td>28.6.2. Attached Prompt Windows</td>
<td>28.50</td>
</tr>
<tr>
<td>28.6.3. Window Operations And Attached Windows</td>
<td>28.50</td>
</tr>
<tr>
<td>28.6.4. Window Properties Of Attached Windows</td>
<td>28.53</td>
</tr>
</tbody>
</table>
[This page intentionally left blank]
Windows provide a means by which different programs can share a single display harmoniously. Rather than having every program directly manipulating the screen bitmap, all display input/output operations are directed towards windows, which appear as rectangular regions of the screen, with borders and titles. The Interlisp-D window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion (see page 28.12). This allows existing interlisp programs to be used without change, while providing a base for experimentation with more complex windows in new applications.

Menus are a special type of window provided by the window system, used for displaying a set of items to the user, and having the user select one using the mouse and cursor. The window system uses menus to provide the interactive interface for manipulating windows. The menu facility also allows users to create and use menus in interactive programs (see page 28.37).

Sometimes, a program needs to use a number of windows, displaying related information. The attached window facility (page 28.45) makes it easy to manipulate a group of windows as a single unit, moving and reshaping them together.

This chapter documents the Interlisp-D window system. First, it describes the default windows and menus supplied by the window system. Then, the programmatic facilities for creating windows. Next, the functions for using menus. Finally, the attached window facility.

Warning: The window system assumes that all programs follow certain conventions concerning control of the screen. All user programs should use perform display operations using windows and menus. In particular, user programs should not perform operate directly on the screen bitmap; otherwise the window system will not work correctly. For specialized applications that require taking complete control of the display, the window system can be turned off (and back on again) with the following function:

\[
\text{(WINDOWS\textworldline{WORLD} \text{FLAG})} \quad \text{[NoSpread Function]}
\]

The window system is turned on if \text{FLAG} is \text{T} and off if \text{FLAG} is \text{NIL}. \text{WINDOWS\textworldline{WORLD}} returns the previous state of the window system.

---

28. WINDOWS AND MENUS
system (T or NIL). If WINDOWWORLD is given no arguments, it simply returns the current state without affecting the window system.

28.1 Using The Window System

When Interlisp-D is initially started, the display screen lights up, showing a number of windows, including the following:

Interlisp-D

This window is the "logo window," used to identify the system. The logo window is bound to the variable LOGOW until it is closed. The user can create other windows like this by calling the following function:

(LOGOW STRING WHERE TITLE ANGLEDEL TA)  [Function]

Creates a window formatted like the "logo window." STRING is the string to be printed in big type in the window; if NIL, "Interlisp-D" is used. WHERE is the position of the lower-left corner of the window; if NIL, the user is asked to specify a position. TITLE is the window title to use; if NIL, it defaults to the Xerox copyright notice and date. ANGLEDELTA specifies the angle (in degrees) between the boxes in the picture; if NIL, it defaults to 23 degrees.
This window is the "executive window," used for typing expressions and commands to the Interlisp-D executive, and for the executive to print any results (see page 13.1). For example, in the above picture, the user typed in \((\text{PLUS} \ 3 \ 4)\), the executive evaluated it, and printed out the result, 7. The upward-pointing arrow (A) is the flashing caret, which indicates where the next keyboard typein will be printed (see page 28.30).

This window is the "prompt window," used for printing various system prompt messages. It is available to user programs through the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROMPTWINDOW</td>
<td>Global variable containing the prompt window.</td>
</tr>
<tr>
<td>(PROMPTPRINT (EXP_1) ... (EXP_N))</td>
<td>[NoSpread Function] Clear the prompt window, and prints (EXP_1) through (EXP_N) in the prompt window.</td>
</tr>
<tr>
<td>(CLRPROMPT)</td>
<td>[Function] Clears the prompt window.</td>
</tr>
</tbody>
</table>

The Interlisp-D window system allows the user to interactively manipulate the windows on the screen, moving them around, changing their shape, etc. by selecting various operations from a menu.

For most windows, depressing the RIGHT mouse key when the cursor is inside a window during I/O wait will cause the window to come to the top and a menu of window operations to appear. If a command is selected from this menu (by releasing the right mouse key while the cursor is over a command), the selected operation will be applied to the window in which the menu was brought up. It is possible for an applications program to redefine the action of the RIGHT mouse key. In these cases, there is a convention that the default command menu may be brought up by depressing the RIGHT key when the cursor is in the header or border of a window (page 28.28). The operations are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>[Window Menu Command] Closes the window, i.e., removes it from the screen. (See CLOSEW, page 28.15.)</td>
</tr>
</tbody>
</table>
Snap  [Window Menu Command]
Prompts for a region on the screen and makes a new window whose bits are a snapshot of the bits currently in that region. Useful for saving some particularly choice image before the window image changes.

Paint  [Window Menu Command]
Switches to a mode in which the cursor can be used like a paint brush to draw in a window. This is useful for making notes on a window. While the LEFT key is down, bits are added. While the MIDDLE key is down, they are erased. The RIGHT button pops up a command menu that allows changing of the brush shape, size and shade, changing the mode of combining the brush with the existing bits, or stopping paint mode.

Clear  [Window Menu Command]
Clears the window and repositions it to the left margin of the first line of text (below the upper left corner of the window by the amount of the font ascent).

Bury  [Window Menu Command]
Puts the window on the bottom of the occlusion stack, thereby exposing any windows that it was hiding.

Redisplay  [Window Menu Command]
Redisplays the window. (See REDISPLAYW, page 28.16.)

Hardcopy  [Window Menu Command]
Prints the contents of the window to the printer. If the window has a window property HARDCOPYFN (page 28.34), it is called with two arguments, the window and an image stream to print to, and the HARDCOPYFN must do the printing. In this way, special windows can be set up that know how to print their contents in a particular way. If the window does not have a HARDCOPYFN, the bitmap image of the window (including the border and title) are printed on the file or printer.

To save the image in a Press or Interpress-format file, or to send it to a non-default printer, use the submenu of the Hardcopy command, indicated by a gray triangle on the right edge of the Hardcopy menu item. If the mouse is moved off of the right of the menu item, another pop-up menu will appear giving the choices "To a file" or "To a printer." If "To a file" is selected, the user is prompted to supply a file name, and the format of the file (Press, Interpress, etc.), and the specified region will be stored in the file.
If "To a printer" is selected, the user is prompted to select a printer from the list of known printers, or to type the name of another printer. If the printer selected is not the first printer on DEFAULTPRINTER (page 29.4), the user will be asked whether to move or add the printer to the beginning of this list, so that future printing will go to the new printer.

**Move**

[Window Menu Command]

Moves the window to a location specified by depressing and then releasing the LEFT key. During this time a ghost frame will indicate where the window will reappear when the key is released. (See GETBOXPOSITION, page 28.9.)

**Shape**

[Window Menu Command]

Allows the user to specify a new region for the existing window contents. If the LEFT key is used to specify the new region, the reshaped window can be placed anywhere. If the MIDDLE key is used, the cursor will start out tugging at the nearest corner of the existing window, which is useful for making small adjustments in a window that is already positioned correctly. This is done by calling the function SHAPEW (page 28.16).

Occasionally, a user will have a number of large windows on the screen, making it difficult to access those windows being used. To help with the problem of screen space management, the Interlisp-D window system allows the creation of "icons." An icon is a small rectangle (containing text or a bitmap) which is a "shrunken-down" form of a particular window. Using the Shrink and Expand commands, the user can shrink windows not currently being used into icons, and quickly restore the original windows at any time.

**Shrink**

[Window Menu Command]

Removes the window from the screen and brings up its icon. (See SHRINKW, page 28.21.) The window can be restored by selecting Expand from the window command menu of the icon.

If the RIGHT button is pressed while the cursor is in an icon, the window command menu will contain a slightly different set of commands. The Redisplay and Clear commands are removed, and the Shrink command is replaced with the Expand command:

**Expand**

[Window Menu Command]

Restores the window associated with this icon and removes the icon. (See EXPANDW, page 28.22.)
If the RIGHT button is pressed while the cursor is not in any window, a "background menu" appears with the following operations:

**Idle** [Background Menu Command]
Enters "idle mode" (see page 12.4), which blacks out the display screen to save the phosphor. Idle mode can be exited by pressing any key on the keyboard or mouse. This menu command has subitems that allow the user to interactively set idle options to erase the password cache (for security), to request a password before exiting idle mode, to change the timeout before idle mode is entered automatically, etc.

**SaveVM** [Background Menu Command]
Calls the function SAVEVM (page 12.7), which writes out all of the dirty pages of the virtual memory. After a SAVEVM, and until the pagefault handler is next forced to write out a dirty page, your virtual memory image will be continuable (as of the SAVEVM) should you experience a system crash or other disaster.

**Snap** [Background Menu Command]
The same as the window menu command Snap described above.

**Hardcopy** [Background Menu Command]
Prompts for a region on the screen, and sends the bitmap image to the printer by calling HARDCOPYW (page 29.3). Note that the region can cross window boundaries.

Like the Hardcopy window menu command (above), the user can print to a file or specify a printer by using a submenu.

**PSW** [Background Menu Command]
Prompts the user for a position on the screen, and creates a "process status window" that allows the user to examine and manipulate all of the existing processes (see page 23.16).

Various system utilities (TEdit, DEdit, TTYIN) allow information to be "copy-inserted" at the current cursor position by selecting it with the "copy" key held down ( Normally the shift keys are the "copy" key; this action can be changed in the key action table.)

To "copy-insert" the bitmap of a snap into a Tedit document. If the right mouse button is pressed in the background with the copy key held down, a menu with the single item "SNAP" appears. If this item is selected, the user is prompted to select a region, and a bitmap containing the bits in that region of the screen is inserted into the current tty process, if that process is able to accept image objects.
Some built-in facilities and Lispusers packages add commands to the background menu, to provide an easy way of calling the different facilities. The user can determine what these new commands do by holding the RIGHT button down for a few seconds over the item in question; an explanatory message will be printed in the prompt window.

### 28.2 Changing Window Command Menus

The following functions provide a functional interface to the interactive window operations so that user programs can call them directly.

**Function**

**(DOWINDOWCOM WINDOW)**

If `WINDOW` is a `WINDOW` that has a `DOWINDOWCOMFN` window property, it APPLYs that property to `WINDOW`. Shrunken windows have a `DOWINDOWCOMFN` property that presents a window command menu that contains "expand" instead of "shrink".

If `WINDOW` is a `WINDOW` that doesn't have a `DOWINDOWCOMFN` window property, it brings up the window command menu. The initial items in these menus are described above. If the user selects one of the items from the provided menu, that item is APPLYed to `WINDOW`.

If `WINDOW` is `NIL`, `DOBACKGROUNDCOM` (below) is called.

If `WINDOW` is not a `WINDOW` or `NIL`, `DOWINDOWCOM` simply returns without doing anything.

**Function**

**(DOBACKGROUNDCOM)**

Brings up the background menu. The initial items in this menu are described above. If the user selects one of the items from the menu, that item is EVALed.

The window command menu for unshrunk windows is cached in the variable `WindowMenu`. To change the entries in this menu, the user should change the change the menu "command lists" in the variable `WindowMenuCommands`, and set the appropriate menu variable to a non-MENU, so the menu will be recreated. This provides a way of adding commands to the menu, of changing its font or of restoring the menu if it gets clobbered. The window command menus for icons and the background have similar pairs of variables, documented below. The "command lists" are in the format of the ITEMS field of a menu (see page 28.39), except as specified below.
Note: Command menus are recreated using the current value of MENUFONT.

**WindowMenu**

- [Variable]

**WindowMenuCommands**

- [Variable]
  
  The menu that is brought up in response to a right button in an unshrunken window is stored on the variable WindowMenu. If WindowMenu is set to a non-MENU, the menu will be recreated from the list of commands WindowMenuCommands. The CADR of each command added to WindowMenuCommands should be a function name that will be APPLYed to the window.

**IconWindowMenu**

- [Variable]

**IconWindowMenuCommands**

- [Variable]
  
  The menu that is brought up in response to a right button in a shrunken window is stored on the variable IconWindowMenu. If it is NIL, it is recreated from the list of commands IconWindowMenuCommands. The CADR of each command added a function name that will be APPLYed to the window.

**BackgroundMenu**

- [Variable]

**BackgroundMenuCommands**

- [Variable]
  
  The menu that is brought up in response to a right button in the background is stored on the variable BackgroundMenu. If it is NIL, it is recreated from the list of commands BackgroundMenuCommands. The CADR of each command added to BackgroundMenuCommands should be a form that will be EVALed.

**BackgroundCopyMenu**

- [Variable]

**BackgroundCopyMenuCommands**

- [Variable]
  
  The menu that is brought up in response to a right button in the background when the copy key is down is stored on the variable BackgroundCopyMenu. If it is NIL, it is recreated from the list of commands BackgroundCopyMenuCommands. The CADR of each command added to BackgroundCopyMenuCommands should be a form that will be EVALed.
28.3 Interactive Display Functions

The following functions can be used by programs to allow the user to interactively specify positions or regions on the display screen.

(GETPOSITION WINDOW CURSOR) [Function]

Returns a POSITION that is specified by the user. GETPOSITION waits for the user to press and release the left button of the mouse and returns the cursor position at the time of release. If WINDOW is a WINDOW, the position will be in the coordinate system of WINDOW's display stream. If WINDOW is NIL, the position will be in screen coordinates. If CURSOR is a CURSOR (page 30.14), the cursor will be changed to it while GETPOSITION is running. If CURSOR is NIL, the value of the system variable CROSSHAIRS will be used as the cursor.

(GETBOXPOSITION BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG) [Function]

Allows the user to position a "ghost" region of size BOXWIDTH by BOXHEIGHT on the screen, and returns the POSITION of the lower left corner of the region. If PROMPTMSG is non-NIL, GETBOXPOSITION first prints it in the PROMPTWINDOW. GETBOXPOSITION then changes the cursor to a box (using the global variable BOXCURSOR). If ORGX and ORGY are numbers, they are taken to be the original position of the region, and the cursor is moved to the nearest corner of that region. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. If ORGX and ORGY are numbers, the corner of the region formed by (ORGX ORGY BOXWIDTH BOXHEIGHT) that is nearest the cursor position is locked, otherwise the lower left corner is locked. The user can change to another corner by holding down the right button. With the right button down, the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the mouse will snap to the nearest corner, which will then become locked to the cursor. (The held corner can be changed after the left or middle button is down by holding both the original button and the right button down while the cursor is moved to the desired new corner, then letting up just the right button.) When the left or middle button is pressed and released, the lower left corner of the region at the time of release is returned. If WINDOW is a WINDOW, the returned position will be in WINDOW's coordinate system; otherwise it will be in screen coordinates.

Example:

(GETBOXPOSITION 100 200 NIL NIL NIL "Specify the position of the command area.")
prompts the user for a 100 wide by 200 high region and returns its lower left corner in screen coordinates.

\textbf{(GETREGION MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS)} [Function]

Lets the user specify a new region and returns that region in screen coordinates. \texttt{GETREGION} prompts for a region by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: \begin{figure}[h]
\centering
\includegraphics[width=0.2\textwidth]{ghost_region.png}
\caption{Example of a "ghost" region with cursor arrow.}
\end{figure} If the user presses the left button, the corner of a "ghost" region opposite the cursor is locked where it is. Once one corner has been fixed, the ghost region expands as the cursor moves.

To specify a region: (1) Move the ghost box so that the corner opposite the cursor is at one corner of the intended region. (2) Press the left button. (3) Move the cursor to the position of the opposite corner of the intended region while holding down the left button. (4) Release the left button.

Before one corner has been fixed, one can switch the cursor to another corner of the ghost region by holding down the right button. With the right button down, the cursor changes to a "forceps" (\begin{figure}[h]
\centering
\includegraphics[width=0.2\textwidth]{forceps.png}
\caption{Forceps cursor when right button is pressed.}
\end{figure}) and the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the cursor will snap to the nearest corner of the ghost region.

After one corner has been fixed, one can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the cursor will snap to the nearest corner, which will become the moving corner. In this way, the region may be moved all over the screen, before its size and position is finalized.

The size of the initial ghost region is controlled by the \texttt{MINWIDTH}, \texttt{MINHEIGHT}, \texttt{OLDREGION}, and \texttt{INITCORNERS} arguments.

If \texttt{INITCORNERS} is non-NIL, it should be a list specifying the initial corners of a ghost region of the form \texttt{(BASEX BASEY OPPX OPPY)}, where \texttt{(BASEX, BASEY)} describes the anchored corner of the box, and \texttt{(OPPX, OPPY)} describes the trackable corner (in screen coordinates). The cursor is moved to \texttt{(OPPX, OPPY)}.

If \texttt{INITCORNERS} is NIL, the ghost region will be \texttt{MINWIDTH} wide and \texttt{MINHEIGHT} high. If \texttt{MINWIDTH} or \texttt{MINHEIGHT} is NIL, 0 is used. Thus, for a call to \texttt{GETREGION} with no arguments specified, there will be no initial ghost region. The cursor will be in the lower right corner of the region, if there is one.
If \textit{OLDREGION} is a region and the user presses the middle button, the corner of \textit{OLDREGION} farthest from the cursor position is fixed and the corner nearest the cursor is locked to the cursor.

\textit{MINWIDTH} and \textit{MINHEIGHT}, if given, are the smallest \textit{WIDTH} and \textit{HEIGHT} that the returned region will have. The ghost image will not get any smaller than \textit{MINWIDTH} by \textit{MINHEIGHT}.

If \textit{NEWREGIONFN} is non-NIL, it will be called to determine values for the positions of the corners. This provides a way of "filtering" prospective regions; for instance, by restricting the region to lie on an arbitrary grid. When the user is specifying a region, the region is determined by two of its corners, one that is fixed and one that is tracking the cursor. Each time the cursor moves or a mouse button is pressed, \textit{NEWREGIONFN} is called with three arguments: \textit{FIXEDPOINT}, the position of the fixed corner of the prospective region; \textit{MOVINGPOINT}, the position of the opposite corner of the prospective region; and \textit{NEWREGIONFNARG}. \textit{NEWREGIONFNARG} allows the caller of \textit{GETREGION} to pass information to the \textit{NEWREGIONFN}.

The first time a button is pressed and when the user changes the moving corner via right buttoning, \textit{MOVINGPOINT} is NIL and \textit{FIXEDPOINT} is the position the user selected for the fixed corner of the new region. In this case, the position returned by \textit{NEWREGIONFN} will be used for the fixed corner instead of the one proposed by the user. For all other calls, \textit{FIXEDPOINT} is the position of the fixed corner (as returned by the previous call) and \textit{MOVINGPOINT} is the new position the user selected for the opposite corner. In these cases, the value of \textit{NEWREGIONFN} is used for the opposite corner instead of the one proposed by the user. In all cases, the ghost region is drawn with the values returned by \textit{NEWREGIONFN}. \textit{NEWREGIONFN} can be a list of functions in which case they are called in order with each being passed the result of calling the previous and the value of the last one used as the point.

\begin{verbatim}
(GETBOXREGION WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG) [Function]
Performs the same prompting as GETBOXPOSITION and returns the REGION specified by the user instead of the POSITION of its lower left corner.

(MOUSECONFIRM PROMPTSTRING HELPSTRING WINDOW DON'TCLEARWINDOWFLG) [Function]
MOUSECONFIRM provides a simple way for the user to confirm or abort some action simply by using the mouse buttons. It prints the strings \textit{PROMPTSTRING} and \textit{HELPSTRING} in the window \textit{WINDOW}, changes the cursor to a "little mouse" cursor: Π (stored in the variable \textit{MOUSECONFIRMCURSOR}), and waits for the user to press the left button to confirm, or any other button
\end{verbatim}
to abort. If the left button was the last button released, returns
T, else NIL.

If PROMPTSTRING is NIL, it is not printed out. If HELPSTRING is
NIL, the string "Click LEFT to confirm, RIGHT to abort." is used. If
WINDOW is NIL, the prompt window is used.

Normally, MOUSECONFIRM clears WINDOW before returning. If
DON'TCLEARWINDOWFLG is non-NIL, the window is not cleared.

28.4 Windows

A window specifies a region of the screen, a display stream,
functions that get called when the window undergoes certain
actions, and various other items of information. The basic model
is that a window is a passive collection of bits (on the screen). On
top of this basic level, the system supports many different types
of windows that are linked to the data structures displayed in
them and provide selection and redisplaying routines. In
addition, it is possible for the user to create new types of
windows by providing selection and displaying functions for
them.

Windows are ordered in depth from user to background.
Windows in front of others obscure the latter. Operating on a
window generally brings it to the top.

Windows are located at a certain position on the screen. Each
window has a clipping region that confines all bits written to it
to a region that allows a border around the window, and a title
above it.

Each window has a display stream associated with it (see page
27.23), and either a window or its display stream can be passed
interchangeably to all system functions. There are dependencies
between the window and its display stream that the user should
not disturb. For instance, the destination bitmap of the display
stream of a window must always be the screen bitmap. The X
offset, Y offset, and Clipping Region fields of the display stream
should not be changed.

Windows can be created by the user interactively, under
program control, or may be created automatically by the system.

Windows are in one of two states: "open" or "closed". In an
"open" state, a window is visible on the screen (unless it is
covered by other open windows or off the edge of the screen)
and accessible to mouse operations. In a "closed" state, a
window is not visible and not accessible to mouse operations.
Any attempt to print or draw on a closed window will open it.
28.4.1 Window Properties

The behavior of a window is controlled by a set of "window properties." Some of these are used by the system. However, any arbitrary property name may be used by a user program to associate information with a window. For many applications the user will associate the structure being displayed with its window using a property. The following functions provide for reading and setting window properties:

(WINDOWPROP WINDOW PROP NEWVALUE) [NoSpread Function]

Returns the previous value of WINDOW's PROP aspect. If NEWVALUE is given, (even if given as NIL), it is stored as the new PROP aspect. Some aspects cannot be set by the user and will generate errors. Any PROP name that is not recognized is stored on a property list associated with the window.

(WINDOWADDPROP WINDOW PROP ITEMTOADD FIRSTFLG) [Function]

WINDOWADDPROP adds a new item to a window property. If ITEMTOADD is EQ to an element of the PROP property of the window WINDOW, nothing is added. If the current property is not a list, it is made a list before ITEMTOADD added. WINDOWADDPROP returns the previous property. If FIRSTFLG is non-NIL, the new item goes on the front of the list; otherwise, it goes on the end of the list. If FIRSTFLG is non-NIL and ITEMTOADD is already on the list, it is moved to the front.

Many window properties (OPENFN, CLOSEFN, etc.) can be a list of functions. WINDOWADDPROP is useful for adding additional functions to a window property without affecting any existing functions. Note that if the order of items in a window property is important, the list can be modified using WINDOWPROP.

(WINDOWDELPROP WINDOW PROP ITEMTODELETE) [Function]

WINDOWDELPROP deletes ITEMTODELETE from the window property PROP of WINDOW and returns the previous list if ITEMTODELETE was an element. If ITEMTODELETE was not a member of window property PROP, NIL is returned.

28.4.2 Creating Windows

(CREATEW REGION TITLE BORDERSIZE NOOPENFLG) [Function]

Creates a new window. REGION indicates where and how large the window should be by specifying the exterior region of the window. The usable height and width of the resulting window will be smaller than the height and width of the region by twice the border size and further less the height of the title, if any. If
REGION is NIL, GETREGION is called to prompt the user for a region.

If TITLE is non-NIL, it is printed in the border at the top of the window. The TITLE is printed using the global display stream WindowTitleDisplayStream. Thus the height of the title will be (FONTPROP WindowTitleDisplayStream 'HEIGHT).

If BORDERSIZE is a number, it is used as the border size. If BORDERSIZE is not a number, the window will have a border WBORDER (initially 4) bits wide.

If NOOPENFLG is non-NIL, the window will not be opened, i.e. displayed on the screen.

The initial X and Y positions of the window are set to the upper left corner by calling MOVETOUPPERLEFT (page 27.14).

(DECODE.WINDOW.ARG WHERESPEC WIDTH HEIGHT TITLE BORDER NOOPENFLG)

[Function]

This is a useful function for creating windows. WHERESPEC can be a WINDOW, a REGION, a POSITION or NIL. If WHERESPEC is a WINDOW, it is returned. In all other cases, CREATEW is called with the arguments TITLE BORDER and NOOPENFLG. The REGION argument to CREATEW is determined from WHERESPEC as follows:

If WHERESPEC is a REGION, it is adjusted to be on the screen, then passed to CREATEW.

If WIDTH and HEIGHT are numbers and WHERESPEC is a POSITION, the region whose lower left corner is WHERESPEC, whose width is WIDTH and whose height is HEIGHT is adjusted to be on the screen, then passed to CREATEW.

If WIDTH and HEIGHT are numbers and WHERESPEC is not a POSITION, then GETBOXREGION is called to prompt the user for the position of a region that is WIDTH by HEIGHT.

If WIDTH and HEIGHT are not numbers, CREATEW is given NIL as a REGION argument.

If WIDTH and HEIGHT are used, they are used as interior dimensions for the window.

(WINDOWP X)

[Function]

Returns X if X is a window, NIL otherwise.
### 28.4.3 Opening and Closing Windows

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(OPENWP WINDOW)</strong></td>
<td>Returns WINDOW, if WINDOW is an open window (has not been closed); NIL otherwise.</td>
</tr>
<tr>
<td><strong>(OPENWINDOWS)</strong></td>
<td>Returns a list of all open windows.</td>
</tr>
<tr>
<td><strong>(OPENW WINDOW)</strong></td>
<td>If WINDOW is a closed window, OPENW calls the function or functions on the window property OPENFN of WINDOW, if any. If one of the OPENFNs is the atom DON'T, the window will not be opened. Otherwise the window is placed on the occlusion stack of windows and its contents displayed on the screen. If WINDOW is an open window, it returns NIL.</td>
</tr>
<tr>
<td><strong>(CLOSEW WINDOW)</strong></td>
<td>CLOSEW calls the function or functions on the window property CLOSEFN of WINDOW, if any. If one of the CLOSEFNs is the atom DON'T or returns the atom DON'T as a value, CLOSEW returns without doing anything further. Otherwise, CLOSEW removes WINDOW from the window stack and restores the bits it is obscuring. If WINDOW was closed, WINDOW is returned as the value. If it was not closed, (for example because its CLOSEFN returned the atom DON'T), NIL is returned as the value. WINDOW can be restored in the same place with the same contents (reopened) by calling OPENW or by using it as the source of a display operation.</td>
</tr>
</tbody>
</table>

**OPENFN**

The OPENFN window property can be a single function or a list of functions. If one of the OPENFNs is the atom DON'T, the window will not be opened. Otherwise, the OPENFNs are called after a window has been opened by OPENW, with the window as a single argument.

**CLOSEFN**

The CLOSEFN window property can be a single function or a list of functions that are called just before a window is closed by CLOSEW. The function(s) will be called with the window as a single argument. If any of the CLOSEFNs are the atom DON'T, or if the value returned by any of the CLOSEFNs is the atom DON'T, the window will not be closed.

Note: If the CAR of the CLOSEFN list is a LAMBDA word, it is treated as a single function.
Note: A CLOSEFN should not call CLOSEW on its argument.

28.4.4 Redisplaying Windows

(REDISPLAYW WINDOW REGION ALWAYSFLG)  [Function]

Redisplay the region REGION of the window WINDOW. If REGION is NIL, the entire window is redisplayed.

If WINDOW doesn’t have a REPAINTFN (page 28.16), the action depends on the value of ALWAYSFLG. If ALWAYSFLG is NIL, WINDOW will not change and the message "Window has no REPAINTFN. Can’t redisplay." will be printed in the prompt window. If ALWAYSFLG is non-NIL, REDISPLAYW acts as if REPAINTFN was NIL.

REPAINTFN  [Window Property]

The REPAINTFN window property can be a single function or a list of functions that are called to repaint parts of the window by REDISPLAYW. The REPAINTFNs are called with two arguments: the window and the region in the coordinates of the window’s display stream of the area that should be repainted. Before the REPAINTFN is called, the clipping region of the window is set to clip all display operations to the area of interest so that the REPAINTFN can display the entire window contents and the results will be appropriately clipped.

Note: CLEARW (page 28.31) should not be used in REPAINTFNs because it resets the window’s coordinate system. If a REPAINTFN wants to clear its region first, it should use DSPFILL (page 27.20).

28.4.5 Reshaping Windows

(SHAPEW WINDOW NEWREGION)  [Function]

Reshapes WINDOW. If the window property RESHAPEFN is the atom DON'T or a list that contains the atom DON'T, a message is printed in the prompt window, WINDOW is not changed, and NIL is returned. Otherwise, RESHAPEFN window property can be a single function or a list of functions that are called when a window is reshaped, to reformat or redisplay the window contents (see below). If the RESHAPEFN window property is NIL, RESHAPEBYREPAINTFN is the default.

If the region NEWREGION is NIL, it prompts for a region with GETREGION (page 28.10). When calling GETREGION, the function MINIMUMWINDOWSIZE is called to determine the minimum height and width of the window, the function
WINDOWREGION is called to get the region passed as the OLDREGION argument, the window property NEWREGIONFN is used as the NEWREGIONFN argument and WINDOW as the NEWREGIONFNARG argument. If the window property INITCORNERSFN is non-NIL, it is applied to the window, and the value is passed as the INITCORNERS argument to GETREGION, to determine the initial size of the "ghost region." These window properties allow the window to specify the regions used for interactive calls to SHAPEW.

If the region NEWREGION is a REGION and its WIDTH or HEIGHT less than the minimums returned by calling the function MINIMUMWINDOWSIZE, they will be increased to the minimums.

If WINDOW has a window property DOSHAPEFN, it is called, passing it WINDOW and NEWREGION (or the region returned by GETREGION). If WINDOW does not have a DOSHAPEFN window property, the function SHAPEW1 is called to reshape the window. DOSHAPEFNS are provided to implement window groups and few users should ever write them. They are tricky to write and must call SHAPEW1 eventually. The RESHAPEFN window property is a simpler hook into reshape operations.

**SHAPEW1 WINDOW REGION**

Changes WINDOW's size and position on the screen to be REGION. After clearing the region on the screen, it calls the window's RESHAPEFN, if any, passing it three arguments: (1) WINDOW, (2) a bitmap that contains WINDOW's previous screen image and (3) the region of WINDOW's old image within the bitmap.

**RESHAPEFN**

The RESHAPEFN window property can be a single function or a list of functions that are called when a window is reshaped by SHAPEW. If the RESHAPEFN is DON'T or a list containing DON'T, the window will not be reshaped. Otherwise, the function(s) are called after the window has been reshaped, its coordinate system readjusted to the new position, the title and border displayed, and the interior filled with texture. The RESHAPEFN should display any additional information needed to complete the window's image in the new position and shape. The RESHAPEFN is called with four arguments: (1) the window in its reshaped form, (2) a bitmap with the image of the old window in its old shape, and (3) the region within the bitmap that contains the window's old image, and (4) the region of the screen previously occupied by this window. This function is provided so that users can reformat window contents or whatever. RESHAPEBYREPAINTFN (below) is the default and should be useful for many windows.
NEWREGIONFN

[Window Property]

If SHAPEW calls GETREGION to prompt the user for a region, the value of the NEWREGIONFN window property is passed as the NEWREGIONFN argument to GETREGION (page 28.10).

INITCORNERSFN

[Window Property]

If this window property is non-NIL, it should be a function of one argument, a window, that returns a list specifying the initial corners of a "ghost region" of the form (BASEX BASEY OPPX OPPY), where (BASEX, BASEY) describes the anchored corner of the box, and (OPPX, OPPY) describes the trackable corner. If SHAPEW calls GETREGION to prompt the user for a region, this function is applied to the window, and the list returned is passed as the INITCORNERS argument to GETREGION (page 28.10), to specify the initial ghost region.

DOSHAPEFN

[Window Property]

If this window property is non-NIL, it is called by SHAPEW to reshape the window (instead of SHAPEW1). It is called with two arguments: the window and the new region.

(RESHAPEBYREPAINTFN WINDOW OLDIMAGE IMAGEREGION OLDSCREENREGION)

[Function]

This the default window RESHAPEFN. WINDOW is a window that has been reshaped from the screen region OLDSCREENREGION to its new region (available via (WINDOWPROP WINDOW 'REGION)). OLDIMAGE is a bitmap that contains the image of the window from its previous location. IMAGEREGION is the region within OLDIMAGE that contains the old image.

RESHAPEBYREPAINTFN BITBLTs the old region contents into the new region. If the new shape is larger in either or both dimensions, the newly exposed areas are redisplayed via calls WINDOW's REPAINTFN window property (page 28.16). RESHAPEBYREPAINTFN may call the REPAINTFN up to four times during a single reshape.

The choice of which areas of the window to remove or extend is done as follows. If WINDOW's new region shares an edge with OLDSCREENREGION, that edge of the window image will remain fixed and any addition or reduction in that dimension will be performed on the opposite side. If WINDOW has an EXTENT property and the newly exposed window area is outside of it, any extra will be added so as to show EXTENT that was previously not visible. An exception to these rules is that the current X,Y position is kept visible, if it was visible before the reshape.
28.4.6 Moving Windows

(MOVEW WINDOW POSorX Y) [Function]
Moves WINDOW to the position specified by POSorX and Y according to the following rules:

If POSorX is NIL, GETBOXPOSITION (page 28.9) is called to read a position from the user. If WINDOW has a CALCULATEREGION window property, it will be called with WINDOW as an argument and should return a region which will be used to prompt the user with. If WINDOW does not have a CALCULATEREGION window property, the region of WINDOW is used to prompt with.

If POSorX is a POSITION, POSorX is used.

If POSorX and Y are both NUMBERP, a position is created using POSorX as the XCOORD and Y as the YCOORD.

If POSorX is a REGION, a position is created using its LEFT as the XCOORD and BOTTOM as the YCOORD.

If WINDOW is not open and POSorX is non-NIL, the window will be moved without being opened. Otherwise, it will be opened.

If WINDOW has the atom DON'T as a MOVEFN window property, the window will not be moved. If WINDOW has any other non-NIL value as a MOVEFN property, it should be a function or list of functions that will be called before the window is moved with the WINDOW and the new position as its arguments. If it returns the atom DON'T, the window will not be moved. If it returns a position, the window will be moved to that position instead of the new one. If there are more than one MOVEFNs, the last one to return a value is the one that determines where the window is moved to.

If WINDOW is moved and WINDOW has an AFTERMOVEFN window property, it should be a function or a list of functions that will be called after the window is moved with WINDOW as an argument.

MOVEW returns the new position, or NIL if the window could not be moved.

Note: If MOVEW moves any part of the window from off-screen onto the screen, that part is redisplayed (by calling REDISPLAYW).

(RELMOVEW WINDOW POSITION) [Function]
Like MOVEW for moving windows but the POSITION is interpreted relative to the current position of WINDOW.

Example: The following code moves WINDOW to the right one screen point.

(RELMOVEW WINDOW (create POSITION XCOORD ← 1 YCOORD ← 0))
CALCULATEREGION  
[Window Property]  
If MOVEW calls GETBOXPOSITION to prompt the user for a region, the CALCULATEREGION window property is called (passing the window as an argument. The CALCULATEREGION should returns a region to be used to prompt the user with. If CALCULATEREGION is NIL, the region of the window is used to prompt with.

MOVEFN  
[Window Property]  
If the MOVEFN is DON'T, the window will not be moved by MOVEW. Otherwise, if the MOVEFN is non-NIL, it should be a function or a list of functions that will be called before a window is moved with two arguments: the window being moved and the new position of the lower left corner in screen coordinates. If the MOVEFN returns DON'T, the window will not be moved. If the MOVEFN returns a POSITION, the window will be moved to that position. Otherwise, the window will be moved to the specified new position.

AFTERMOVEFN  
[Window Property]  
If non-NIL, it should be a function or a list of functions that will be called after the window is moved (by MOVEW) with the window as an argument.

28.4.7 Exposing and Burying Windows

(TOTOPW WINDOW NOCALLTSTOPFNFLG)  
[Function]  
Brings WINDOW to the top of the stack of overlapping windows, guaranteeing that it is entirely visible. If WINDOW is closed, it is opened. This is done automatically whenever a printing or drawing operation occurs to the window.

If NOCALLTSTOPFNFLG is NIL, the TOTOPFN of WINDOW is called (page 28.20). If NOCALLTSTOPFNFLG is T, it is not called, which allows a TOTOPFN to call TOTOPW without causing an infinite loop.

(BURYW WINDOW)  
[Function]  
Puts WINDOW on the bottom of the stack by moving all the windows that it covers in front of it.

TOTOPFN  
[Window Property]  
If non-NIL, whenever the window is brought to the top, the TOTOPFN is called (with the window as a single argument). This function may be used to bring a collection of windows to the top together.
If the NOCALLTOPWFN argument of TOTOPW is non-NIL, the TOTOPFN of the window is not called, which provides a way of avoiding infinite loops when using TOTOPW from within a TOTOPFN.

28.4.8 Shrinking Windows Into Icons

Occasionally, a user will have a number of large windows on the screen, making it difficult to access those windows being used. To help with the problem of screen space management, the Interlisp-D window system allows the creation of Icons. An icon is a small rectangle (containing text or a bitmap) which is a "shrunken-down" form of a particular window. Using the Shrink and Expand window menu commands (page 28.5), the user can shrink windows not currently being used into icons, and quickly restore the original windows at any time. This facility is controlled by the following functions and window properties:

(SHRINKW WINDOW TOWHAT ICONPOSITION EXPANDFN) [Function]

SHRINKW makes a small icon which represents WINDOW and removes WINDOW from the screen. Icons have a different window command menu that contains "EXPAND" instead of "SHRINK". The EXPAND command calls EXPANDW which returns the shrunken window to its original size and place. The icon can also be moved by pressing the LEFT button in it, or expanded by pressing the MIDDLE button in it.

The SHRINKFN property of the window WINDOW affects the operation of SHRINKW. If the SHRINKFN property of WINDOW is the atom DON'T, SHRINKW returns. Otherwise, the SHRINKFN property of the window is treated as a (list of) function(s) to apply to WINDOW; if any returns the atom DON'T, SHRINKW returns.

TOWHAT, if given, indicates the image the icon window will have. If TOWHAT is a string, atom or list, the icon's image will be that string (currently implemented as a title-only window with TOWHAT as the title.) If TOWHAT is a BITMAP, the icon's image will be a copy of the bitmap. If TOWHAT is a WINDOW, that window will be used as the icon.

If TOWHAT is not given (as is the case when invoked from the SHRINK window command), then the following apply in turn: (1) If the window has an ICONFN property, it gets called with the two arguments WINDOW and OLDICON, where WINDOW is the window being shrunk and OLDICON is the previously created icon, if any. The ICONFN should return one of the TOWHAT entities described above or return the OLDICON if it does not want to change it. (2) If the window has an ICON property, it is used as the value of TOWHAT. (3) If the window has neither an
ICONFN or ICON property, the icon will be WINDOW's title or, if WINDOW doesn't have a title, the date and time of the icon creation.

ICONPOSITION gives the position that the new icon will be on the screen. If it is NIL, the icon will be in the corner of the window furthest from the center of the screen.

In all but the default case, the icon is cached on the property ICONWINDOW of WINDOW so repeating SHRINKW reuses the same icon (unless overridden by the ICONFN described above). Thus to change the icon it is necessary to remove the ICONWINDOW property or call SHRINKW explicitly giving a TOWHAT argument.

( EXPANDW ICONW )  [Function]
Restores the window for which ICONW is an icon, and removes the icon from the screen. If the EXPANDFN window property of the main window is the atom DON'T, the window won't be expanded. Otherwise, the window will be restored to its original size and location and the EXPANDFN (or list of functions) will be applied to it.

SHRINKFN  [Window Property]
The SHRINKFN window property can be a single function or a list of functions that are called just before a window is shrunken by SHRINKW, with the window as a single argument. If any of the SHRINKFNS are the atom DON'T, or if the value returned by any of the SHRINKFNS is the atom DON'T, the window will not be shrunk.

ICONFN  [Window Property]
If SHRINKW is called without begin given a TOWHAT argument (as is the case when invoked from the SHRINK window command) and the window's ICONFN property is non-NIL, then it gets called with two arguments, the window being shrunk and the previously created icon, if any. The ICONFN should return one of the TOWHAT entities described above or return the previously created icon if it does not want to change it.

ICON  [Window Property]
If SHRINKW is called without being given a TOWHAT argument, the window's ICONFN property is NIL, and the ICON property is non-NIL, then it is used as the value of TOWHAT.
iconwindow

Whenever an icon is created, it is cached on the property ICONWINDOW of the window, so calling SHRINKW again will reuse the same icon (unless overridden by the ICONFN).

Thus, to change the icon it is necessary to remove the ICONWINDOW property or call SHRINKW (page 28.21) explicitly giving a TOWHAT argument.

expandfn

The EXPANDFN window property can be a single function or a list of functions. If one of the EXPANDFNs is the atom DON'T, the window will not be expanded. Otherwise, the EXPANDFNs are called after the window has been expanded by EXPANDW, with the window as a single argument.

28.4.9 Coordinate Systems, Extents, And Scrolling

Note: The word "scrolling" has two distinct meanings when applied to Interlisp-D windows. This section documents the use of "scroll bars" on the left and bottom of a window to move an object displayed in the window. "Scrolling" also describes the feature where trying to print text off the bottom of a window will cause the contents to "scroll up." This second feature is controlled by the function DSPSCROLL (page 27.24).

One way of thinking of a window is as a "view" onto an object (e.g. a graph, a file, a picture, etc.) The object has its own natural coordinate system in terms of which its subparts are laid out. When the window is created, the X Offset and Y Offset of the window's display stream are set to map the origin of the object's coordinate system into the lower left point of the window's interior region. At the same time, the Clipping Region of the display stream is set to correspond to the interior of the window.

From then on, the display stream's coordinate system is translated and its clipping region adjusted whenever the window is moved, scrolled or reshaped.

There are several distinct regions associated with a window viewing an object. First, there is a region in the window's coordinate system that contains the complete image of the object. This region (which can only be determined by application programs with knowledge of the "semantics" of the object) is stored as the EXTENT property of the window (below). Second, the clipping region of the display stream (obtainable with the function DSPCLIPPINGREGION, page 27.11) specifies the portion of the object that is actually visible in the window. This is set so that it corresponds to the interior of the window (not including the border or title). Finally, there is the region on the screen that specifies the total area that the window occupies, including the
border and title. This region (in screen coordinates) is stored as the REGION property of the window (page 28.34).

The window system supports the idea of scrolling the contents of a window. Scrolling regions are on the left and the bottom edge of each window. The LEFT key is used to indicate upward or leftward scrolling by the amount necessary to move the selected position to the top or the left edge. The RIGHT key is used to indicate downward or rightward scrolling by the amount necessary to move the top or left edge to the selected position. The MIDDLE key is used to indicate global placement of the object within the window (similar to “thumbing” a book). In the scroll region, the part of the object that is being viewed by the window is marked with a gray shade. If the whole scroll bar is thought of as the entire object, the shaded portion is the portion currently being viewed. This will only occur when the window “knows” how big the object is (see window property EXTENT, page 28.26).

When the button is released in a scroll region, the function SCROLLW is called. SCROLLW calls the scrolling function associated with the window to do the actual scrolling and provides a programmable entry to the scrolling operation.

(SCROLLW WINDOW DELTAX DELTAY CONTINUOUSFLG) [Function]
Calls the SCROLLFN window property of the window WINDOW with arguments WINDOW, DELTAX, DELTAY and CONTINUOUSFLG. See SCROLLFN window property, page 28.26.

(SCROLL.HANDLER WINDOW) [Function]
This is the function that tracks the mouse while it is in the scroll region. It is called when the cursor leaves a window in either the left or downward direction. If WINDOW does not have a scroll region for this direction (e.g. the window has moved or reshaped since it was last scrolled), a scroll region is created that is SCROLLBARWIDTH wide. It then waits for SCROLLWAITTIME milliseconds and if the cursor is still inside the scroll region, it opens a window the size of the scroll region and changes the cursor to indicate the scrolling is taking place.

When a button is pressed, the cursor shape is changed to indicate the type of scrolling (up, down, left, right or thumb). After the button is held for WAITBEFORESCROLLTIME milliseconds, until the button is released SCROLLW is called each WAITBETWEENSCROLLTIME milliseconds. These calls are made with the CONTINUOUSFLG argument set to T. If the button is released before WAITBEFORESCROLLTIME milliseconds, SCROLLW is called with the CONTINUOUSFLG argument set to NIL.
The arguments passed to SCROLLW depend on the mouse button. If the LEFT button is used in the vertical scroll region, \(DY\) is distance from cursor position at the time the button was released to the top of the window and \(DX\) is 0. If the RIGHT button is used, the inverse of this quantity is used for \(DY\) and 0 for \(DX\). If the LEFT button is used in the horizontal scroll region, \(DX\) is distance from cursor position to left of the window and \(DY\) is 0. If the RIGHT button is used, the inverse of this quantity is used for \(DX\) and 0 for \(DY\).

If the MIDDLE button is pressed, the distance argument to SCROLLW will be a FLOATP between 0.0 and 1.0 that indicates the proportion of the distance the cursor was from the left or top edge to the right or bottom edge.

Note: The scrolling regions will not come up if the window has a SCROLLFN window property of NIL, has a non-NIL NOSCROLLBARS window property, or if its SCROLLEXTENTUSE (page 28.26) property has certain values and its EXTENT is fully visible.

(SCROLLBYREPAINTFN WINDOW DELTAX DELTAY CONTINUOUSFLG) [Function]

SCROLLBYREPAINTFN is the standard scrolling function which should be used as the SCROLLFN property for most scrolling windows.

This function, when used as a SCROLLFN, BITBLTs the bits that will remain visible after the scroll to their new location, fills the newly exposed area with texture, adjusts the window's coordinates and then calls the window's REPAINFTFN on the newly exposed region. Thus this function will scroll any window that has a repaint function.

If WINDOW has an EXTENT property (page 28.26), SCROLLBYREPAINTFN will limit scrolling in the X and Y directions according to the value of the window property SCROLLEXTENTUSE (page 28.26).

If DELTAX or DELTAY is a FLOATP, SCROLLBYREPAINTFN will position the window so that its top or left edge will be positioned at that proportion of its EXTENT. If the window does not have an EXTENT, SCROLLBYREPAINTFN will do nothing.

If CONTINUOUSFLG is non-NIL, this indicates that the scrolling button is being held down. In this case, SCROLLBYREPAINTFN will scroll the distance of one linefeed height (as returned by DSPLINEFEED, page 27.12).

Scrolling is controlled by the following window properties:
EXTENT

[Window Property]

Used to limit scrolling operations. Accesses the extent region of the window. If non-NIL, the EXTENT is a region in the window's display stream that contains the complete image of the object being viewed by the window. User programs are responsible for updating the EXTENT. The functions UNIONREGIONS, EXTENDREGION, etc. (page 27.2) are useful for computing a new extent region.

In some situations, it is useful to define an EXTENT that only exists in one dimension. This may be done by specifying an EXTENT region with a width or height of -1. SCROLLFN handling recognizes this situation as meaning that the negative EXTENT dimension is unknown.

SCROLLFN

[Window Property]

If the SCROLLFN property is NIL, the window will not scroll. Otherwise, it should be a function of four arguments: (1) the window being scrolled, (2) the distance to scroll in the horizontal direction (positive to right, negative to left), (3) the distance to scroll in the vertical direction (positive up, negative down), and (4) a flag which is T if the scrolling button is being held down. For more information, see SCROLL.HANDLER (page 28.24). For most scrolling windows, the SCROLLFN function should be SCROLLBYREPAINTFN (page 28.25).

NOSCROLLBARS

[Window Property]

If the NOSCROLLBARS property is non-NIL, scroll bars will not be brought up for this window. This disables mouse-driven scrolling of a window. This window can still be scrolled using SCROLLW (page 28.24).

SCROLLEXTENTUSE

[Window Property]

SCROLLBYREPAINTFN uses the SCROLLEXTENTUSE window property to limit how far scrolling can go in the X and Y directions. The possible values for SCROLLEXTENTUSE and their interpretations are:

NIL This will keep the extent region visible or near visible. It will not scroll the window so that the top of the extent is below the top of the window, the bottom of the extent is more than one point above the top of the window, the left of the extent is to the right of the window and the right of the extent is to the left of the window. The EXTENT can be scrolled to just above the window to provide a way of "hiding" the contents of a window. In this mode the extent is either in the window or just of the top of the window.

T The extent is not used to control scrolling. The user can scroll the window to anywhere. Having the EXTENT window property
does all thumb scrolling to be supported so that the user can get back to the EXTENT by thumb scrolling.

**LIMIT**
This will keep the extent region visible. The window is only allowed to view within the extent.

+ This will keep the extent region visible or just off in the positive direction in either X or Y (i.e. the image will be either be visible or just off to the top and/or right.)

- This will keep the extent region visible or just off in the negative direction in either X or Y (i.e. the image will be either be visible or just off to the left and/or bottom).

++ This will keep the extent region visible or just off in the window (i.e. the image will be either be visible or just off to the left, bottom, top or right).

-+ This will keep the extent region visible or just off in the window (i.e. the image will be either be visible or just off to the left, bottom, top or right).

**XBEHAVIOR** and **YBEHAVIOR** should each be one of the atoms (NIL T LIMIT + - + - +). The interpretations of the atoms is the same as above except that NIL is equivalent to LIMIT.

Note: The NIL value of **SCROLLXEXTENTUSE** is equivalent to (LIMIT . +).

Example: If the **SCROLLXEXTENTUSE** window property of a window (with an extent defined) is (LIMIT . T), the window will scroll uncontrolled in the Y dimension but be limited to the extent region in the X dimension.

### 28.4.10 Mouse Activity in Windows

The following window properties allow the user to control the response to mouse activity in a window. The value of these properties, if non-NIL, should be a function that will be called (with the window as argument) when the specified event occurs.

Note: these functions should be "self-contained", communicating with the outside world solely via their window argument, e.g., by setting window properties. In particular, these functions should not expect to access variables bound on the stack, as the stack context is formally undefined at the time these functions are called. Since the functions are invoked asynchronously, they perform any terminal input/output operations from their own window.

**WINDOWENTRYFN**
Whenever a button goes down in the window and the process associated with the window is not the tty process, the
WINDOWENTRYFN is called. The default is GIVE.TTY.PROCESS (page 23.13) which gives the process associated with the window the tty and calls the BUTTONEVENTFN. WINDOWENTRYFN can be a list of functions and all will be called.

CURSORINFN [Window Property]
Whenever the mouse moves into the window, the CURSORINFN is called. If CURSORINFN is a list of functions, all will be called.

CURSOROUTFN [Window Property]
The CURSOROUTFN is called when the cursor leaves the window. If CURSOROUTFN is a list of functions, all will be called.

CURSORMOVEDFN [Window Property]
The CURSORMOVEDFN is called whenever the cursor has moved and is inside the window. CURSORMOVEDFN can be a list of functions and all will be called. This allows a window function to implement "active" regions within itself by having its CURSORMOVEDFN determine if the cursor is in a region of interest, and if so, perform some action.

BUTTONEVENTFN [Window Property]
The BUTTONEVENTFN is called whenever there is a change in the state (up or down) of the mouse buttons inside the window. Changes to the mouse state while the BUTTONEVENTFN is running will not be interpreted as new button events, and the BUTTONEVENTFN will not be re-invoked.

RIGHTBUTTONFN [Window Property]
The RIGHTBUTTONFN is called in lieu of the standard window menu operation (DOWINDOWCOM) when the RIGHT key is depressed in a window. More specifically, the RIGHTBUTTONFN is called instead of the BUTTONEVENTFN when (MOUSESTATE (ONLY RIGHT)). If the RIGHT key is to be treated like any other key in a window, supply RIGHTBUTTONFN and BUTTONEVENTFN with the same function.

When an application program defines its own RIGHTBUTTONFN, there is a convention that the default RIGHTBUTTONFN, DOWINDOWCOM (page 28.7), may be executed by depressing the RIGHT key when the cursor is in the header or border of a window. User RIGHTBUTTONFNS are encouraged to follow this convention, by calling DOWINDOWCOM if the cursor is not in the interior region of the window.
BACKGROUNDUTTONEVENTFN

BACKGROUNDCURSORINFN

BACKGROUNDCURSOROUTFN

BACKGROUNDCURSORMOVEDFN

These variables provide a way of taking action when there is cursor action and the cursor is in the background. They are interpreted like the corresponding window properties. If set to the name of a function, that function will be called, respectively, whenever the cursor is in the background and a button changes, when the cursor moves into the background from a window, when the cursor moved from the background into a window and when the cursor moves from one place in the background to another.

28.4.11 Terminal I/O and Page Holding

Each process has its own terminal i/o stream (accessed as the stream T, page 25.1). The terminal i/o stream for the current process can be changed to point to a window by using the function TTYDISPLAYSTREAM, so that output and echoing of type-in is directed to a window.

(TTYDISPLAYSTREAM DISPLAYSTREAM)

Selects the display stream or window DISPLAYSTREAM to be the terminal output channel, and returns the previous terminal output display stream. TTYDISPLAYSTREAM puts DISPLAYSTREAM into scrolling mode and calls PAGEHEIGHT with the number of lines that will fit into DISPLAYSTREAM given its current Font and Clipping Region. The line length of TTYDISPLAYSTREAM is computed (like any other display stream) from its Left Margin, Right Margin, and Font. If one of these fields is changed, its line length is recalculated. If one of the fields used to compute the number of lines (such as the Clipping Region or Font) changes, PAGEHEIGHT is not automatically recomputed. (TTYDISPLAYSTREAM (TTYDISPLAYSTREAM)) will cause it to be recomputed.

If the window system is active, the line buffer is saved in the old TTY window, and the line buffer is set to the one saved in the window of the new display stream, or to a newly created line buffer (if it does not have one). Caution: It is possible to move the TTYDISPLAYSTREAM to a nonvisible display stream or to a window whose current position is not in its clipping region.
(PAGEHEIGHT N) [Function]

If \( N \) is greater than 0, it is the number of lines of output that will be printed to TTYDISPLAYSTREAM before the page is held. A page is held before the \( N+1 \) line is printed to TTYDISPLAYSTREAM without intervening input if there is no terminal input waiting to be read. The output is held with the screen video reversed until a character is typed. Output holding is disabled if \( N \) is 0. PAGEHEIGHT returns the previous setting.

PAGEFULLFN [Window Property]

If the PAGEFULLFN window property is non-NIL, it will be called with the window as a single argument when the window is full (i.e., when enough has been printed since the last TTY interaction so that the next character printed will cause information to be scrolled off the top of the window.)

If the PAGEFULLFN window property is NIL, the system function PAGEFULLFN is called. PAGEFULLFN simply returns if there are characters in the type-in buffer for WINDOW, otherwise it inverts the window and waits for the user to type a character. PAGEFULLFN is user advisable.

Note: The PAGEFULLFN window property is only called on windows which are the TTYDISPLAYSTREAM of some process.

28.4.12 The TTY Process and the Caret

At any time, one process is designated as the TTY process, which is used for accepting keyboard input. The TTY process can be changed to a given process by calling GIVE.TTY.PROCESS (page 23.13), or by clicking the mouse in a window associated with the process. The latter mechanism is implemented with the following window property:

PROCESS [Window Property]

If the PROCESS window property is non-NIL, it should be a PROCESS and will be made the TTY process by GIVE.TTY.PROCESS (page 23.13), the default WINDOWENTRYFN property (page 28.27). This implements the mechanism by which the keyboard is associated with different processes.

The window system uses a flashing caret (A) to indicate the position of the next window typeout. There is only one caret visible at any one time. The caret in the current TTY process is always visible; if it is hidden by another window, its window is brought to the top. An exception to this rule is that the flashing caret's window is not brought to the top if the user is buttoning or has a shift key down. This prevents the destination window
(which has the tty and caret flashing) from interfering with the window one is trying to select text to copy from.

(CARET NEWCARET) [Function]

Sets the shape that blinks at the location of the next output to the current process. NEWCARET should be one of the following:

- a CURSOR object: If NEWCARET is a CURSOR object (see page 30.14), it is used to give the new caret shape.
- OFF: Turns the caret off.
- NIL: The caret is not changed. CARET returns a CURSOR representing the current caret.
- T: Reset the caret to the value of DEFAULTCARET. DEFAULTCARET can be set to change the initial caret for new processes.

The hotspot of NEWCARET indicates which point in the new caret bitmap should be located at the current output position. The previous caret is returned. Note: the bitmap for the caret is not limited to the dimensions CURSORWIDTH by CURSORHEIGHT.

(CARETRATE ONRATE OFFRATE) [Function]

Sets the rate at which the caret for the current process will flash. The caret will be visible for ONRATE milliseconds, then not visible for OFFRATE milliseconds. If OFFRATE is NIL then it is set to be the same as ONRATE. If ONRATE is T, both the "on" and "off" times are set to the value of the variable DEFAULTCARETRATE (initially 333). The previous value of CARETRATE is returned. If the caret is off, CARETRATE return NIL.

28.4.13 Miscellaneous Window Functions

(CLEARW WINDOW) [Function]

Fills WINDOW with its background texture, changes its coordinate system so that the origin is the lower left corner of the window, sets its X position to the left margin and sets its Y position to the base line of the uppermost line of text, ie. the top of the window less the font ascent.

(INVERTW WINDOW SHADE) [Function]

Fills the window WINDOW with the texture SHADE in INVERT mode. If SHADE is NIL, BLACKSHADE is used. INVERTW returns WINDOW so that it can be used inside RESETFORM.
(FLASHWINDOW WIN? N FLASHINTERVAL SHADE) [Function]

Flashes the window WIN? by "inverting" it twice. N is the number of times to flash the window (default is 1). FLASHINTERVAL is the length of time in milliseconds to wait between flashes (default is 200). SHADE is the shade that will be used to invert the window (default is BLACKSHADE).

If WIN? is NIL, the whole screen is flashed. In this case, the SHADE argument is ignored (can only invert the screen).

(WHICHW X Y) [Function]

Returns the window which contains the position in screen coordinates of X if X is a POSITION, the position (X, Y) if X and Y are numbers, or the position of the cursor if X is NIL. Returns NIL if the coordinates are not in any window. If they are in more than one window, it returns the uppermost.

Example: (WHICHW) returns the window that the cursor is in.

(DECODEWINDOW/OR/DISPLAYSTREAM DSORW WINDOWVAR TITLE BORDER) [Function]

Returns a display stream as determined by the DSORW and WINDOWVAR arguments. If DSORW is a display stream, it is returned. If DSORW is a window, its display stream is returned. If DSORW is NIL, the litatom WINDOWVAR is evaluated. If its value is a window, its display stream is returned. If its value is not a window, WINDOWVAR is set to a newly created window (prompting user for region) whose display stream is then returned. If DSORW is NEW, the display stream of a newly created window is returned. If a window is involved in the decoding, it is opened and if TITLE or BORDER are given, the TITLE or BORDER property of the window are reset. The DSORW = NIL case is most useful for programs that want to display their output in a window, but want to reuse the same window each time they are called. The non-NIL cases are good for decoding a display stream argument passed to a function.

(WIDTHIFWINDOW INTERIORWIDTH BORDER) [Function]

Returns the width of the window necessary to have INTERIORWIDTH points in its interior if the width of the border is BORDER. If BORDER is NIL, the default border size WBorder is used.

(HEIGHTIFWINDOW INTERIORHEIGHT TITLEFLG BORDER) [Function]

Returns the height of the window necessary to have INTERIORHEIGHT points in its interior with a border of BORDER and, if TITLEFLG is non-NIL, a title. If BORDER is NIL, the default border size WBorder is used.
 WIDTHIFWINDOW and HEIGHTIFWINDOW are useful for calculating the width and height for a call to GETBOXPOSITION for the purpose of positioning a prospective window.

**(MINIMUMWINDOWSIZ**E WINDOW)**

Returns a dotted pair, the CAR of which is the minimum width WINDOW needs and the CDR or which is the minimum height WINDOW needs.

The minimum size is determined by the value of the window property MINSIZE of WINDOW. If the value of the MINSIZE window property is NIL, the width is 26 and the height is the height WINDOW needs to have its title, border and one line of text visible. If MINSIZE is a dotted pair, it is returned. If it is a litatom, it should be a function which is called with WINDOW as its first argument, which should return a dotted pair.

28.4.14 Miscellaneous Window Properties

**TITLE**

[Window Property]

Accesses the title of the window. If a title is added to a window whose title is NIL or the title is removed (set to NIL) from a window with a title, the window's exterior (its region on the screen) is enlarged or reduced to accommodate the change without changing the window's interior. For example, (WINDOWPROP WINDOW 'TITLE "Results") changes the title of WINDOW to be "Results". (WINDOWPROP WINDOW 'TITLE NIL) removes the title of WINDOW.

**BORDER**

[Window Property]

Accesses the width of the border of the window. The border will have at most 2 point of white (but never more than half) and the rest black. The default border is the value of the global variable WBorder (initially 4).

**WINDOWTITLESHADE**

[Window Property]

Accesses the window title shade of the window. If non-NIL, it should be a texture which is used as the "background texture" for the title bar on the top of the window. If it is NIL, the value of the global variable WINDOWTITLESHADE (initially BLACKSHADE) is used. Note that black is always used as the background of the title printed in the title bar, so that the letters can be read. The remaining space is painted with the "title shade".
HARDCOPYFN

If non-NIL, it should be a function that is called by the window menu command Hardcopy (page 28.4) to print the contents of a window. The HARDCOPYFN property is called with two arguments, the window and an image stream to print to. If the window does not have a HARDCOPYFN, the bitmap image of the window (including the border and title) are printed on the file or printer.

DSP

Value is the display stream of the window. All system functions will operate on either the window or its display stream. This window property cannot be changed using WINDOWPROP.

HEIGHT

Value is the height and width of the interior of the window (the usable space not counting the border and title). These window properties cannot be changed using WINDOWPROP.

WIDTH

Value is a region (in screen coordinates) indicating where the window (counting the border and title) is located on the screen. This window property cannot be changed using WINDOWPROP.

REGION

28.4.15 Example: A Scrollable Window

The following is a simple example showing how one might create a scrollable window.

CREATE.PPWINDOW creates a window that displays the pretty printed expression EXPR. The window properties PPEXPR, PPORIGX, and PPORIGY are used for saving this expression, and the initial window position. Using this information, REPAINT.PPWINDOW simply reinitializes the window position, and prettyprints the expression again. Note that the whole expression is reformatted every time, even if only a small part actually lies within the window. If this window was going to be used to display very large structures, it would be desirable to implement a more sophisticated REPAINTFN that only redisplays that part of the expression within the window. However, this scheme would be satisfactory if most of the items to be displayed are small.

RESHAPE.PPWINDOW resets the window (and stores the initial window position), calls REPAINT.PPWINDOW to display the window's expression, and then sets the EXTENT property of the
window so that SCROLLBYREPAINTFN will be able to handle scrolling and "thumbing" correctly.

(DEFINEQ

(CREATE.PPWINDOW
 [LAMBDA (EXPR) (* rrb "4-OCT-82 12:06")
 (* creates a window that displays
 a pretty printed expression.)

(PROG (WINDOW)
 (* ask the user for a piece of the
 screen and make it into a window.)

(SETQ WINDOW (CREATEW NIL "PP window")
 (* put the expression on the
 property list of the window so that
 the repaint and reshape functions
 can access it.)

(WINDOWPROP WINDOW (QUOTE PPEXPR) EXPR)
 (* set the repaint and reshape
 functions.)

(WINDOWPROP WINDOW (QUOTE REPAINTFN)
 (FUNCTION REPAINT.PPWINDOW))

(WINDOWPROP WINDOW (QUOTE RESHAPEFN)
 (FUNCTION RESHAPE.PPWINDOW))
 (* make the scroll function
 SCROLLBYREPAINTFN, a system
 function that uses the repaint
 function to do scrolling.)

(WINDOWPROP WINDOW (QUOTE SCROLLFN)
 (FUNCTION SCROLLBYREPAINTFN))
 (* call the reshape function to
 initially print the expression and
 calculate its extent.)

(RESHAPE.PPWINDOW WINDOW)
 (RETURN WINDOW))

(REPAINT.PPWINDOW
 [LAMBDA (WINDOW REGION) (* rrb "4-OCT-82 11:52")

 (* the repainting function for a window with a pretty printed expression. This repainting function ignores the region to be repainted and repaints the entire window.)

 (* set the window position to the beginning of the pretty printing of the expression.)
(MOVETO (WINDOWPROP WINDOW (QUOTE PPORIGX)))
(WINDOWPROP WINDOW (QUOTE PPORIGY))
WINDOW)
(PRINTDEF (WINDOWPROP WINDOW (QUOTE PPEXPR))
0 NIL NIL NIL WINDOW))

(RESHAPE.PPWINDOW
[LAMBDA (WINDOW) (* rrb "4-OCT-82 12:01")
(* the reshape function for a window with a pretty printed expression.)

(PROG (BTM)

(* set the position of the window so that the first character appears in the upper left corner and save the X and Y for the repaint function.)

(DSPRESET WINDOW)
(WINDOWPROP WINDOW (QUOTE PPORIGX)
(DSPXPOSITION NIL WINDOW))
(WINDOWPROP WINDOW (QUOTE PPORIGY)
(DSPYPOSITION NIL WINDOW))
(* call the repaint function to pretty print the expression in the newly cleared window.)

(REPAINT.PPWINDOW WINDOW)

(* save the region actually covered by the pretty printed expression so that the scrolling routines will know where to stop. The pretty printing of the expression does a carriage return after the last piece of the expression printed so that the current position is the base line of the next line of text. Hence the last visible piece of the expression (BTM) is the ending position plus the height of the font above the base line (its ASCENT).)

(WINDOWPROP WINDOW (QUOTE EXTENT)
(create REGION
LEFT ← 0
BOTTOM ← [SETQ BTM (ILPLUS
(DSPYPOSITION NIL WINDOW)
(FONTPROP WINDOW (QUOTE ASCENT]
WIDTH ← (WINDOWPROP WINDOW (QUOTE WIDTH))
HEIGHT ← (IDIFFERENCE
(WINDOWPROP WINDOW (QUOTE HEIGHT))
BTM])

28 36 WINDOWS AND MENUS
28.5 Menus

A menu is basically a means of selecting from a list of items. The system provides common layout and interactive user selection mechanisms, then calls a user-supplied function when a selection has been confirmed. The two major constituents of a menu are a list of items and a "when selected function." The label that appears for each item is the item itself for non-lists, or its CAR if the item is a list. In addition, there are a multitude of different formatting parameters for specifying font, size, and layout. When a menu is created, its unspecified fields are filled with defaults and its screen image is computed and saved.

Menus can be either pop up or fixed. If fixed menus are used, the menu must be included in a window.

(MENU MENU POSITION RELEASECONTROLFLG —) [Function]

This function provides menus that pop up when they are used. It displays MENU at POSITION (in screen coordinates) and waits for the user to select an item with a mouse key. Before any mouse key is pressed, the item the mouse is over is boxed. After any key is down, the selected menu item is video reversed. When all keys are released, MENU's WHENSELECTEDFN field is called with four arguments: (1) the item selected, (2) the menu, (3) the last mouse key released (LEFT, MIDDLE, or RIGHT), and (4) the reverse list of superitems rolled through when selecting the item and MENU returns its value. If no item is selected, MENU returns NIL. If POSITION is NIL, the menu is brought up at the value from MENU's MENUPOSITION field, if it is a POSITION, or at the current cursor position. The orientation of MENU with respect to the specified position is determined by its MENUOFFSET field.

If RELEASECONTROLFLG is NIL, this process will retain control of the mouse. In this case, if the user lets the mouse key up outside of the menu, MENU return NIL. (Note: this is the standard way of allowing the user to indicate that they do not want to make the offered choice.) If RELEASECONTROLFLG is non-NIL, this process will give up control of the mouse when it is outside of the menu so that other processes can be run. In this case, clicking outside the menu has no effect on the call to MENU. If the menu is closed (for example, by right buttoning in it and selecting "Close" from the window menu), MENU returns NIL.

Programmers are encouraged to provide a menu item such as
"cancel" or "abort" which gives users a positive way of indicating "no choice".

Note: A "released" menu will stay visible (on top of the window stack) until it is closed or an item is selected.

(ADDMENU MENU WINDOW POSITION DONTOPENFLG) [Function]
This function provides menus that remain active in windows. ADDMENU displays MENU at POSITION (in window coordinates) in WINDOW. If the window is too small to display the entire menu, the window is made scrollable. When an item is selected, the value of the WHENSELECTEDFN field of MENU is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). More than one menu can be put in a window, but a menu can only be added to one window at a time. ADDMENU returns the window into which MENU is placed.

If WINDOW is NIL, a window is created at the position specified by POSITION (in screen coordinates) that is the size of MENU. If a window is created, it will be opened unless DONTOPENFLG is non-NIL. If POSITION is NIL, the menu is brought up at the value of MENU's MENUPOSITION field (in window coordinates), if it is a position, or else in the lower left corner of WINDOW. If both WINDOW and POSITION are NIL, a window is created at the current cursor position.

Warning: ADDMENU resets several of the window properties of WINDOW. The CURSORINFN, CURSORMOVEDFN, and BUTTONEVENTFN window properties are replaced with MENUBUTTONFN, so that MENU will be active. MENUREPAINTFN is added to the REPAINTFN window property to update the menu image if the window is redisplayed. The SCROLLFN window property is changed to SCROLLBYREPAINTFN if the window is too small for the menu, to make the window scroll.

(DELETEMENU MENU CLOSEFLG FROMWINDOW) [Function]
This function removes MENU from the window FROMWINDOW. If MENU is the only menu in the window and CLOSEFLG is non-NIL, its window will be closed (by CLOSEW).

If FROMWINDOW is NIL, the list of currently open windows is searched for one that contains MENU. If none is found, DELETEMENU does nothing.

28.5.1 Menu Fields
A menu is a datatype with the following fields:
ITEMS

[Menu Field]
The list of items to appear in the menu. If an item is a list, its CAR will appear in the menu. If the item (or its CAR) is a bitmap, the bitmap will be displayed in the menu. The default selection functions interpret each item as a list of three elements: a label, a form whose value is returned upon selection, and a help string that is printed in the prompt window when the user presses a mouse key with the cursor pointing to this item. The default subitem function interprets the fourth element of the list. If it is a list whose CAR is the litatom SUBITEMS, the CDR is taken as a list of subitems.

SUBITEMFN

[Menu Field]
A function to be called to determine if an item has any subitems. If an item has subitems and the user rolls the cursor out the right of that item, a submenu with that item’s subitems in it pops up. If the user selects one of the items from the submenu, the selected subitem is handled as if it were selected from the main menu. If the user rolls out of the submenu to the left, the submenu is taken down and selection resumes from the main menu.

An item with subitems is marked in the menu by a grey, right pointing triangle following the label.

The function is called with two arguments: (1) the menu and (2) the item. It should return a list of the subitems of this item if any. (Note: it is called twice to compute the menu image and each time the user rolls out of the item box so it should be moderately efficient. The default SUBITEMFN, DEFAULTSUBITEMFN, checks to see if the item is a list whose fourth element is a list whose CAR is the litatom SUBITEMS and if so, returns the CDR of it.

For example:

(create MENU
  ITEMS ← '(AAAA (BBBB 'BBBB "help string for BBBB"
              (SUBITEMS BBBB1 BBBB2 BBBB3)))))

will create a menu with items A and B in which B will have subitems B1, B2 and B3. The following picture below shows this menu as it first appears:

```
AAAA
BBBB
```

The following picture shows the submenu, with the item BBBB3 selected by the cursor (•):

```
AAAA
BBBB1
BBBB2
BBBB3
```
WHENSElectedFN
A function to be called when an item is selected. The function is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). The default function DEFAULTWHENSElectedFN evaluates and returns the value of the second element of the item if the item is a list of at least length 2. If the item is not a list of at least length 2, DEFAULTWHENSElectedFN returns the item.

Note: If the menu is added to a window with ADDMENU, the default WHENSElectedFN is BACKGROUNDWHENSElectedFN, which is the same as DEFAULTWHENSElectedFN except that EVALASPROCESS (page 23.17) is used to evaluate the second element of the item, instead of tying up the mouse process.

WHENHEldFN
The function which is called when the user has held a mouse key on an item for MENUHEldWAIT milliseconds (initially 1200). The function is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). WHENHEldFN is intended for prompting users. The default is DEFAULTMENUHEldFN which prints (in the prompt window) the third element of the item or, if there is not a third element, the string "This item will be selected when the button is released."

WHENUNHEldFN
If WHENHEldFN was called, WHENUNHEldFN will be called: (1) when the cursor leaves the item, (2) when a mouse key is released, or (3) when another key is pressed. The function is called with the same three argument values used to call WHENHEldFN. The default WHENUNHEldFN is the function CLRPROMPT (page 28.3), which just clears the prompt window.

MENUPOSITION
The position of the menu to be used if the call to MENU or ADDMENU does not specify a position. For popup menus, this is in screen coordinates. For fixed menus, it is in the coordinates of the window the menu is in. The point within the menu image that is placed at this position is determined by MENUOFFSET. If MENUPOSITION is NIL, the menu will be brought up at the cursor position.

MENUOFFSET
The position in the menu image that is to be located at MENUPOSITION. The default offset is (0,0). For example, to bring up a menu with the cursor over a particular menu item, set
its MENUOFFSET to a position within that item and set its MENUPosition to NIL.

**MENUFONT**  
(Menu Field)  
The font in which the items will be appear in the menu. Default is the value of MENUFONT.

**TITLE**  
(Menu Field)  
If non-NIL, the value of this field will appear as a title in a line above the menu.

**MENUTITLEFONT**  
(Menu Field)  
The font in which the title of the menu will be appear. If this is NIL, the title will be in the same font as window titles. If it is T, it will be in the same font as the menu items.

**CENTERFLG**  
(Menu Field)  
If non-NIL, the menu items are centered; otherwise they are left-justified.

**MENUROWS**  
(Menu Field)  

**MENUCOLUMNS**  
(Menu Field)  
These fields control the shape of the menu in terms of rows and columns. If MENUROWS is given, the menu will have that number of rows. If MENUCOLUMNS is given, the menu will have that number of columns. If only one is given, the other one will be calculated to generate the minimal rectangular menu. (Normally only one of MENUROWS or MENUCOLUMNS is given.) If neither is given, the items will be in one column.

**ITEMHEIGHT**  
(Menu Field)  
The height of each item box in the menu. If not specified, it will be the maximum of the height of the MENUFONT and the heights of any bitmaps appearing as labels.

**ITEMWIDTH**  
(Menu Field)  
The width of each item box in the menu. If not specified, it will be the width of the largest item in the menu.

**MENUBORDERSIZE**  
(Menu Field)  
The size of the border around each item box. If not specified, 0 (no border) is used.
MENUS

MENOUTLINESIZE [Menu Field]
The size of the outline around the entire menu. If not specified, a maximum of 1 and the MENUBORDERSIZE is used.

CHANGEOFFSETFLG [Menu Field]
(popup menus only) If CHANGEOFFSETFLG is non-NIL, the position of the menu offset is set each time a selection is confirmed so that the menu will come up next time in the same position relative to the cursor. This will cause the menu to reappear in the same place on the screen if the cursor has not moved since the last selection. This is implemented by changing the MENUOFFSET field on each use. If CHANGEOFFSETFLG is the atom X or the atom Y, only the X or the Y coordinate of the MENUOFFSET field will be changed. For example, by setting the MENUOFFSET position to (-1,0) and setting CHANGEOFFSETFLG to Y, the menu will pop up so that the cursor is just to the left of the last item selected. This is the setting of the window command menus.

The following fields are read only.

IMAGEHEIGHT [Menu Field]
Returns the height of the entire menu.

IMAGEWIDTH [Menu Field]
Returns the width of the entire menu.

28.5.2 Miscellaneous Menu Functions

(MAXMENUITEMWIDTH MENU) [Function]
Returns the width of the largest menu item label in the menu MENU.

(MAXMENUITEMHEIGHT MENU) [Function]
Returns the height of the largest menu item label in the menu MENU.

(MENUREGION MENU) [Function]
Returns the region covered by the image of MENU in its window.

(WFROMMENU MENU) [Function]
Returns the window MENU is located in, if it is in one; NIL otherwise.
**DOSELECTEDITEM** (MENU ITEM BUTTON)

Calls MENU's WHENSELECTEDFN on ITEM and BUTTON. It provides a programmatic way of making a selection. It does not change the display.

**MENUITEMREGION** (ITEM MENU)

Returns the region occupied by ITEM in MENU.

**SHADEITEM** (ITEM MENU SHADE DS/W)

Shades the region occupied by ITEM in MENU. If DS/W is a display stream or a window, it is assumed to be where MENU is displayed. Otherwise, WFROMMENU is called to locate the window MENU is in. Shading is persistent, and is reapplied when the window the menu is in gets redisplayed. To unshade an item, call with a SHADE of 0.

**PUTMENUPROP** (MENU PROPERTY VALUE)

Stores the property PROPERTY with the value VALUE on a property list in the menu MENU. The user can use this property list for associating arbitrary data with a menu object.

**GETMENUPROP** (MENU PROPERTY)

Returns the value of the PROPERTY property of the menu MENU.

### 28.5.3 Examples of Menu Use

**Example:** A simple menu:

```
(MENU (create MENU ITEMS → '((YES T) (NO (QUOTE NIL)))))
```

Creates a menu with items YES and NO in a single vertical column:

```
YES
NO
```

If YES is selected, T will be returned. Otherwise, NIL will be returned.

**Example:** A simple menu, with centering:

```
(MENU (create MENU TITLE → "Foo?"
       ITEMS → '((YES T "Adds the Foo feature.")
                     (NO 'NO "Removes the Foo feature.")
                     CENTERFLG ← T))
```

Creates a menu with a title Foo? and items YES and NO centered in a single vertical column:

```
Foo?
YES
NO
```
The strings following the YES and NO are help strings and will be printed if the cursor remains over one of the items for a period of time. This menu differs from the one above in that it distinguishes the NO case from the case where the user clicked outside of the menu. If the user clicks outside of the menu, NIL is returned.

**Example:**

A multi-column menu:

```lisp
(create MENU ITEMS ← '(1 2 3 4 5 6 7 8 9 * 0 #)
CENTERFLG ← T
MENUCOLUMNS ← 3
MENUFONT ← (FONTCREATE 'MODERN 10 'BOLD)
ITEMHEIGHT ← 15
ITEMWIDTH ← 15
CHANGEOFFSETFLG ← T)
```

Creates a touch-tone-phone number pad with the items in 15 by 15 boxes printed in Modern 10 bold font:

```
1 2 3
4 5 6
7 8 9
* 0 #
```

If used in pop up mode, its first use will have the cursor in the middle. Subsequent use will have the cursor in the same relative location as the previous selection.

**Example:**

A program using a previously-saved menu:

```lisp
(SELECTQ (MENU
(COND ((type? MENU FOOMENU)
  (* use previously computed menu.)
  FOOMENU)
  (T (* create and save the menu)
  (SETQ FOOMENU
  (create MENU
  ITEMS ← '((A 'A-SELECTED "prompt string for A")
  (B 'B-SELECTED "prompt string for B")
  (A-SELECTED (* if A is selected) (DOATHING))
  (B-SELECTED (* if B is selected) (DOSTHING))
  (PROGN (* user selected outside the menu) NIL)))
```

This expression displays a pop up menu with two items, A and B, and waits for the user to select one. If A is selected, DOATHING is called. If B is selected, DOSTHING is called. If neither of these is selected, the form returns NIL.

The purpose of this example is to show some good practices to follow when using menus. First, the menu is only created once, and saved in the variable FOOMENU. This is more efficient if the menu is used more than once. Second, all of the information about the menu is kept in one place, which makes it easy to
understand and edit. Third, the forms evaluated as a result of selecting something from the menu are part of the code and hence will be known to masterscope (as opposed to the situation if the forms were stored as part of the items). Fourth, the items in the menu have help strings for the user. Finally, the code is commented (always worth the trouble).

28.6 Attached Windows

The attached window facility makes it easy to manipulate a group of windows as a unit. Standard window operations like moving, reshaping, opening, and closing can be done so that it appears to the user as if the windows are a single entity. Each collection of attached windows has one main window and any number of other windows that are "attached" to it. Moving or reshaping the main window causes all of the attached windows to be moved or reshaped as well. Moving or reshaping an attached window does not affect the main window.

Attached windows can have other windows attached to them. Thus, it is possible to attach window A to window B when B is already attached to window C. Similarly, if A has other windows attached to it, it can still be attached to B.

(ATTACHWINDOW WINDOWTOATTACH MAINWINDOW EDGE POSITIONONEDGE)

[Function]

Associates WINDOWTOATTACH with MAINWINDOW so that window operations done to MAINWINDOW are also done to WINDOWTOATTACH (the exact set of window operations passed between main windows and attached windows is described on page 28.51). ATTACHWINDOW moves WINDOWTOATTACH to the correct position relative to MAINWINDOW.

Note: A window can be attached to only one other window. Attaching a window to a second window will detach it from the first. Attachments can not form loops. That is, a window cannot be attached to itself or to a window that is attached to it. ATTACHWINDOW will generate an error if this is attempted.

EDGE determines which edge of MAINWINDOW the attached window is positioned along: it should be one of TOP, BOTTOM, LEFT, or RIGHT. If EDGE is NIL, it defaults to TOP.

POSITIONONEDGE determines where along EDGE the attached window is positioned. It should be one of the following:

LEFT The attached window is placed on the left (of a TOP or BOTTOM edge).
ATTACHED WINDOWS

RIGHT The attached window is placed on the right (of a TOP or BOTTOM edge).

BOTTOM The attached window is placed on the bottom (of a LEFT or RIGHT edge).

TOP The attached window is placed on the top (of a LEFT or RIGHT edge).

CENTER The attached window is placed in the center of the edge.

JUSTIFY or NIL The attached window is placed to fill the entire edge. ATTACHWINDOW reshapes the window if necessary.

Note: The width or height used to justify an attached window includes any other windows that have already been attached to MAINWINDOW. Thus (ATTACHWINDOW BBB AAA 'RIGHT 'JUSTIFY) followed by (ATTACHWINDOW CCC AAA 'TOP 'JUSTIFY) will put CCC across the top of both BBB and AAA:

```
   CCC
   
   AAA  BBB
```

WINDOWCOMACTION provides a convenient way of specifying how WINDOWTOATTACH responds to right button menu commands. The window property PASSTOMAINCOMS determines which right button menu commands are directly applied to the attached window, and which are passed to the main window (see page 28.51). Depending on the value of WINDOWCOMACTION, the PASSTOMAINCOMS window property of WINDOWTOATTACH is set as follows:

NIL PASSTOMAINCOMS is set to (CLOSEW MOVEW SHAPEW SHRINKW BURYW), so right button menu commands to close, move, shape, shrink, and bury are passed to the main window, and all others are applied to the attached window.

LOCALCLOSE PASSTOMAINCOMS is set to (MOVEW SHAPEW SHRINKW BURYW), which is the same as when WINDOWCOMACTION is NIL, except that the attached window can be closed independently.

HERE PASSTOMAINCOMS is set to NIL, so all right button menu commands are applied to the attached window.

MAIN PASSTOMAINCOMS is set to T, so all right button menu commands are passed to the main window.

Note: If the user wants to set the PASSTOMAINCOMS window property of an attached window to something else, it must be done after the window is attached, since ATTACHWINDOW modifies this window property.
(DETACHWINDOW WINDOWTODETACH) [Function]
Detaches WINDOWTODETACH from its main window. Returns a dotted pair (EDGE . POSITIONONEDGE) if WINDOWTODETACH was an attached window, NIL otherwise. This does not close WINDOWTODETACH.

(DETACHALLWINDOWS MAINWINDOW) [Function]
Detaches and closes all windows attached to MAINWINDOW.

(FREEATTACHEDWINDOW WINDOW) [Function]
Detaches the attached window WINDOW. In addition, other attached windows above (in the case of a TOP attached window) or below (in the case of a BOTTOM attached window) are moved closer to the main window to fill the gap.

Note: Attached windows that "reject" the move operation (see REJECTMAINCOMS, page 28.51) are not moved.

Note: FREEATTACHEDWINDOW currently doesn't handle LEFT or RIGHT attached windows.

(REMOVEWINDOW WINDOW) [Function]
Closes WINDOW, and calls FREEATTACHEDWINDOW to move other attached windows to fill any gaps.

(REPOSITIONATTACHEDWINDOWS WINDOW) [Function]
Repositions every window attached to WINDOW, in the order that they were attached. This is useful as a RESHAPEFN for main windows with attached window that don't want to be reshaped, but do want to keep their position relative to the main window when the main window is reshaped.

Note: Attached windows that "reject" the move operation (see REJECTMAINCOMS, page 28.51) are not moved.

(MAINWINDOW WINDOW RECURSEFLG) [Function]
If WINDOW is not a window, it generates an error. If WINDOW is closed, it returns WINDOW. If WINDOW is not attached to another window, it returns WINDOW itself. If RECURSEFLG is NIL and WINDOW is attached to a window, it returns that window. If RECURSEFLG is T, it returns the first window up the "main window" chain starting at WINDOW that is not attached to any other window.

(ATTACHEDWINDOWS WINDOW COM) [Function]
Returns the list of windows attached to WINDOW.
If COM is non-NIL, only those windows attached to WINDOW that do not reject the window operation COM are returned (see REJECTMAINCOMS, page 28.51).

(ALLATTACHEDWINDOWS WINDOW)  [Function]
Returns a list of all of the windows attached to WINDOW or attached to a window attached to it.

(WINDOWREGION WINDOW COM)  [Function]
Returns the screen region occupied by WINDOW and its attached windows, if it has any.
If COM is non-NIL, only those windows attached to WINDOW that do not reject the window operation COM are considered in the calculation (see REJECTMAINCOMS, page 28.51).

(WINDOWSIZE WINDOW)  [Function]
Returns the size of WINDOW and its attached windows (if any), as a dotted pair (WIDTH, HEIGHT).

(MINATTACHEDWINDOWEXTENT WINDOW)  [Function]
Returns the minimum size that WINDOW and its attached windows (if any) will accept, as a dotted pair (WIDTH, HEIGHT).

28.6.1 Attaching Menus To Windows
The following functions are provided to associate menus to windows.

(MENUWINDOW MENU VERTFLG)  [Function]
Returns a closed window that has the menu MENU in it. If MENU is a list, a menu is created with MENU as its ITEMS menu field (see page 28.39). Otherwise, MENU should be a menu. The returned window has the appropriate RESHAPEFN, MINSIZE and MAXSIZE window properties to allow its use in a window group.
If both the MENUROWS and MENUCOLUMNS fields of MENU are NIL, VERTFLG is used to set the default menu shape. If VERTFLG is non-NIL, the MENUCOLUMNS field of MENU will be set to 1 (the menu items will be listed vertically); otherwise the MENUROWS field of MENU will be set to 1 (the menu items will be listed horizontally).

(ATTACHMENU MENU MAINWINDOW EDGE POSITIONONEDGE NOOPENFLG)  [Function]
Creates a window that contains the menu MENU (by calling MENUWINDOW) and attaches it to the window MAINWINDOW.
on edge EDGE at position POSITIONONEDGE. The menu window is opened unless MAINWINDOW is closed, or NOOPENFLG is T.

If EDGE is either LEFT or RIGHT, MENUWINDOW will be called with VERTFLG = T, so the menu items will be listed vertically; otherwise the menu items will be listed horizontally. These defaults can be overridden by specifying the MENUROWS or MENUCOLUMNS fields in MENU.

(CREATEMENUEDWINDOW MENU WINDOWTITLE LOCATION WINDOWSPEC) [Function]
Creates a window with an attached menu and returns the main window. MENU is the only required argument, and may be a menu or a list of menu items. WINDOWTITLE is a string specifying the title of the main window. LOCATION specifies the edge on which to place the menu; the default is TOP. WINDOWSPEC is a region specifying a region for the aggregate window; if NIL, the user is prompted for a region.

Examples:
(Setq MENUW (CREATEMENUEDWINDOW MENU WINDOWTITLE LOCATION WINDOWSPEC))
creates (but does not open) a menu window that contains the two items "smaller" and "LARGER" with the title "zoom controls" and that calls the function ZOOMMAINWINDOW when an item is selected. Note that the menu items will be listed horizontally, because MENUWINDOW is called with VERTFLG = NIL, and the menu does not specify either a MENUROWS or MENUCOLUMNS field.

(ATTACHWINDOW MENUW (CREATEW '(50 50 150 50)) 'TOP 'JUSTIFY)
creates a window on the screen and attaches the above created menu window to its top:

<table>
<thead>
<tr>
<th>zoom controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>smaller</td>
</tr>
<tr>
<td>LARGER</td>
</tr>
</tbody>
</table>

(CREATEMENUEDWINDOW
(create MENU
ITEMS ← '(smaller LARGER)
MENUFONT ← '(MODERN 12)
TITLE ← "zoom controls"
CENTERFLG ← T
WHENSELECTEDFN ← (FUNCTION ZOOMMAINWINDOW)))

creates the same sort of window in one step, prompting the user
for a region.

28.6.2 Attached Prompt Windows

Many packages have a need to display status information or
prompt for small amounts of user input in a place outside their
standard window. A convenient way to do this is to attach a
small window to the top of the program's main window. The
following functions do so in a uniform way that can be
depended on among diverse applications.

(GETPROMPTWINDOW MAINWINDOW #LINES FONT DONTCREATE) [Function]
Returns the attached prompt window associated with
MAINWINDOW, creating it if necessary. The window is always
attached to the top of MAINWINDOW, has DSPSCROLL set to T,
and has a PAGEFULLFN of NILL to inhibit page holding. The
window is at least #LINES lines high (default 1); if a pre-existing
window is shorter than that, it is reshaped to make it large
enough. FONT is the font to give the prompt window (defaults
to the font of MAINWINDOW), and applies only when the
window is first created. If DONTCREATE is true, returns the
window if it exists, otherwise NILL without creating any prompt
window.

(REMOVEPROMPTWINDOW MAINWINDOW) [Function]
Detaches the attached prompt window associated with
MAINWINDOW (if any), and closes it.

28.6.3 Window Operations And Attached Windows

When a window operation, such as moving or clearing, is
performed on a window, there is a question about whether or
not that operation should also be performed on the windows
attached to it or performed on the window it is attached to. The
"right" thing to do depends on the window operation: it makes
sense to independently redisplay a single window in a collection
of windows, whereas moving a single window usually implies
moving the whole group of windows. The interpretation of
window operations also depends on the application that the
window group is used for. For some applications, it may be desirable to have a window group where individual windows can be moved away from the group, but still be conceptually attached to the group for other operations. The attached window facility is flexible enough to allow all of these possibilities.

The operation of window operations can be specified by each attached window, by setting the following two window properties:

| PASSTOMAINCOMS | Value is a list of window commands (e.g. CLOSEW, MOVEW) which, when selected from the attached window's right-button menu, are actually applied to the central window in the group, instead of being applied to the attached window itself. The "central window" is the first window up the "main window" chain that is not attached to any other window. If PASSTOMAINCOMS is NIL, all window operations are directly applied to the attached window. If PASSTOMAINCOMS is T, all window operations are passed to the central window.

Note: ATTACHWINDOW (page 28.45) allows this window property to be set to commonly-used values by using its WINDOWCOMACTION argument. ATTACHWINDOW always sets this window property, so users must modify it directly only after attaching the window to another window.

| REJECTMAINCOMS | Value is a list of window commands that the attached window will not allow the main window to apply to it. This is how a window can say "leave me out of this group operation." If REJECTMAINCOMS is NIL, all window commands may be applied to this attached window. If REJECTMAINCOMS is T, no window commands may be applied to this attached window.

Note: The PASSTOMAINCOMS and REJECTMAINCOMS window properties affect right-button menu operations applied to main windows or attached windows, and the action of programmatic window functions (SHAPEW, MOVEW, etc.) applied to main windows. However, these window properties do not affect the action of window functions applied to attached windows.

The following list describes the behavior of main and attached windows under the window operations, assuming that all attached windows have their REJECTMAINCOMS window property set to NIL and PASSTOMAINCOMS set to (CLOSEW MOVEW SHAPEW SHRINKW BURYW) (the default if
ATTACHWINDOW is called with WINDOWCOMACTION = NIL, see page 28.45).

The behavior for any particular operation can be changed for particular attached windows by setting the standard window properties (e.g., MOVEFN or CLOSEFN) of the attached window. An exception is the TOTOPFN property of an attached window, that is set to bring the whole window group to the top and should not be set by the user (although users can add functions to the TOTOPFN window property).

Move
If the main window moves, all attached windows move with it, and the relative positioning between the main window and the attached windows is maintained. If the region is determined interactively, the prompt region for the move is the union of the extent of the main window and all attached windows (excluding those with MOVEW in their REJECTMAINCOMS window property).

If an attached window is moved by calling the function MOVEW, it is moved without affecting the main window. If the right-button window menu command Move is called on an attached window, it is passed on to the main window, so that all windows in the group move.

Reshape
If the main window is reshaped, the minimum size of it and all of its attached windows is used as the minimum of the space for the result. Any space greater than the minimum is distributed among the main window and its attached windows. Attached windows with SHAPEW on their REJECTMAINCOMS window property are ignored when finding the minimum size, creating a "ghost" region, or distributing space after a reshape.

If an attached window is reshaped by calling the function SHAPEW, it is reshaped independently. If the right-button window menu command Shape is called on an attached window, it is passed on to the main window, so the whole group is reshaped.

Note: Reshaping the main window will restore the conditions established by the call to ATTACHWINDOW, whereas moving the main window does not. Thus, if A is attached to the top of B and then moved by the user, its new position relative to B will be maintained if B is moved. If B is reshaped, A will be reshaped to the top of B. Additionally, if, while A is moved away from the top of B, C is attached to the top of B, C will position itself above where A used to be.

Close
If the main window is closed, all of the attached windows are closed also and the links from the attached windows to the main window are broken. This is necessary for the windows to be garbage collected.
If an attached window is closed by calling the function CLOSEW, it is closed without affecting the main window. If the right-button window menu command Close is called on an attached window, it is passed on to the main window. Note that closing an attached window detaches it.

Open
If the main window is opened, it opens all attached windows and reestablishes links from them to the main window.

Attached windows can be opened independently and this does not affect the main window. Note that it is possible to reopen a closed attached window and not have it linked to its main window.

Shrink
The collection of windows shrinks as a group. The SHRINKFNs of the attached windows are evaluated but the only icon displayed is the one for the main window.

Redisplay
The main or attached windows can be redisplayed independently.

Totop
If any main or attached window is brought to the top, all of the other windows are brought to the top also.

Expand
Expanding any of the windows expands the whole collection.

Scrolling
All of the windows involved in the group scroll independently.

Clear
All windows clear independently of each other.

28.6.4 Window Properties Of Attached Windows

Windows that are involved in a collection either as a main window or as an attached window have properties stored on them. The only properties that are intended to be set be set by the user are the MINSIZE, MAXSIZE, PASSTOMAINCOMS, and REJECTMAINCOMS window properties. The other properties should be considered read only.

MINSIZE [Window Property]

MAXSIZE [Window Property]

Each of these window properties should be a dotted pair (WIDTH, HEIGHT) or a function to apply to the window that returns a dotted pair. The numbers are used when the main window is reshaped. The MINSIZE is used to determine the size of the smallest region acceptable during reshaping. Any amount greater than the collective minimum is spread evenly among the windows until each reaches MAXSIZE. Any excess is given to the main window.

Note: If you give the main window of an attached window group a MINSIZE or MAXSIZE property, its value is moved to the
MAINWINDOWMINSIZE or MAINWINDOWMAXSIZE property, so that the main window can be given a size function that computes the minimum or maximum size of the entire group. Thus, if you want to change the main window's minimum or maximum size after attaching windows to it, you should change the MAINWINDOWMINSIZE or MAINWINDOWMAXSIZE property instead.

Note: This doesn’t address the hard problem of overlapping attached windows side to side, for example if window A was attached as [TOP, LEFT] and B as [TOP, RIGHT]. Currently, the attached window functions do not worry about the overlap.

The default MAXSIZE is NIL, which will let the region grow indefinitely.

---

**MAINWINDOW**  
[Window Property]  
Pointer from attached windows to the main window of the group. This link is not available if the main window is closed. The function MAINWINDOW (page 28.47) is the preferred way to access this property.

---

**ATTACHEDWINDOWS**  
[Window Property]  
Pointer from a window to its attached windows. The function ATTACHEDWINDOWS (page 28.47) is the preferred way to access this property.

---

**WHEREATTACHED**  
[Window Property]  
For attached windows, a dotted pair (EDGE. POSITIONONEDGE) giving the edge and position on the edge that determine how the attached window is placed relative to its main window.

---

The TOTOPFN window property on attached windows and the properties TOTOPFN, DOSHAPEFN, MOVEFN, CLOSEFN, OPENFN, SHRINKFN, EXPANDFN and CALCULATEREGIONFN on main windows contain functions that implement the attached window manipulation facilities. Care should be used in modifying or replacing these properties.
# 29. Hardcopy Facilities

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>29.1. Low-level Hardcopy Variables</td>
<td>29.5</td>
</tr>
</tbody>
</table>
Interlisp-D includes facilities for generating hardcopy in "Interpress" format and "Press" format. Interpress is a file format used for communicating documents to Xerox Network System printers such as the Xerox 8044 and Xerox 5700. Press is a file format used for communicating documents to Xerox laser Xerographic printers known by the names "Dover", "Spruce", "Penguin", and "Raven". There are also library packages available for supporting other types of printer formats (4045, FX-80, C150, etc.). The hardcopy facilities are designed to allow the user to support new types of printers with minimal changes to the user interface.

Files can be in a number of formats, including Interpress files, plain text files, and formatted Tedit files. In order to print a file on a given printer, it is necessary to identify the format of the file, convert the file to a format that the printer can accept, and transmit it. Rather than require that the user explicitly determine file types and do the conversion, the Interlisp-D hardcopy functions generate Interpress or other format output depending on the appropriate choice for the designated printer. The hardcopy functions use the variables PRINTERTYPES and PRINTFILETYPES (described below) to determine the type of a file, how to convert it for a given printer, and how to send it. By changing these variables, the user can define other kinds of printers and print to them using the normal hardcopy functions.

```
(SEND.FILE.TO.PRINTER FILE HOST PRINTOPTIONS)
```

The function SEND.FILE.TO.PRINTER causes the file FILE to be sent to the printer HOST. If HOST is NIL, the first host in the list DEFAULTPRINTINGHOST which can print FILE is used.

PRINTOPTIONS is a property list of the form (PROP1 VALUE1 PROP2 VALUE2 ...). The properties accepted depends on the type of printer. For Interpress printers, the following properties are accepted:

- DOCUMENT.NAME The document name to appear on the header page (a string). Default is the full name of the file.
- DOCUMENT.CREATION.DATE The creation date to appear on the header page (a Lisp integer date, such as returned by IDATE). The default value is the creation date of the file.
- SENDER.NAME The name of the sender to appear on the header page (a string). The default value is the name of the user.
RECIPIENT.NAME
The name of the recipient to appear on the header page (a string). The default is none.

MESSAGE
An additional message to appear on the header page (a string). The default is none.

#COPIES
The number of copies to be printed. The default value is 1.

PAGES.TO.PRINT
The pages of the document that should be printed, represented as a list (FIRSTPAGE # LASTPAGE #). For example, if this option is (3 5), this specifies that pages 3 through 5, inclusive, should be printed. Note that the page numbering used for this purpose has no connection to any page numbers that may be printed on the document. The default is to print all of the pages in the document.

MEDIUM
The medium on which the master is to be printed. If omitted, this defaults to the value of NSPRINT.DEFAULT.MEDIUM, as follows: NIL means to use the printer’s default; T means to use the first medium reported available by the printer; any other value must be a Courier value of type MEDIUM. The format of this type is a list (PAPER (KNOWN.SIZE TYPE)) or (PAPER (OTHER.SIZE (WIDTH LENGTH))). The paper TYPE is one of US.LETTER, US.LEGAL, A0 through A10, ISO.B0 through ISO.B10, and JIS.B0 through JIS.B10. For users who use A4 paper exclusively, it should be sufficient to set NSPRINT.DEFAULT.MEDIUM to (PAPER (KNOWN.SIZE "A4").)

When using different paper sizes, it may be necessary to reset the variable DEFAULTPAGEREGION, the region on the page used for printing (measured in micas from the lower-left corner).

STAPLE?
True if the document should be stapled.

#SIDES
1 or 2 to indicate that the document should be printed on one or two sides, respectively. The default is the value of EMPRESS#SIDES.

PRIORITY
The priority of this print request, one of LOW, NORMAL, or HIGH. The default is the printer’s default.

Note: Press printers only recognize the options #COPIES, #SIDES, DOCUMENT.CREATION.DATE, and DOCUMENT.NAME.

For example,

(SEND.FILE.TO.PRINTER 'FOO NIL
 '('#COPIES 3 #SIDES 2 DOCUMENT.NAME "For John")

SEND.FILE.TO.PRINTER calls PRINTERTYPE and PRINTFILETYPE to determine the printer type of HOST and the file format of FILE. If FILE is a formatted file already in a form that the printer can print, it is transmitted directly. Otherwise, CONVERT.FILE.TO.TYPE.FOR.PRINTER is called to do the conversion. [Note: If the file is converted, PRINTOPTIONS is passed to the formatting function, so it can include properties such as HEADING, REGION, and FONTS.] All of these functions
HARDCOPY FACILITIES

use the lists PRINTERTYPES and PRINTFILETYPES to actually determine how to do the conversion.

LISTFILES (page 17.14) calls the function LISTFILES1 to send a single file to a hardcopy printing device. Interlisp-D is initialized with LISTFILES1 defined to call SEND.FILE.TO.PRINTER.

(HARDCOPYW WINDOW/BITMAP/REGION FILE HOST SCALEFACTOR ROTATION PRINTER TYPE) [Function]

Creates a hardcopy file from a bitmap and optionally sends it to a printer. Note that some printers may have limitations concerning how big or how "complicated" the bitmap may be printed.

WINDOW/BITMAP/REGION can either be a WINDOW (open or closed), a BITMAP, or a REGION (interpreted as a region of the screen). If WINDOW/BITMAP/REGION is NIL, the user is prompted for a screen region using GETREGION.

If FILE is non-NIL, it is used as the name of the file for output. If HOST = NIL, this file is not printed. If FILE is NIL, a temporary file is created, and sent to HOST.

To save an image on a file without printing it, perform (HARDCOPYW IMAGE FILE). To print an image to the printer PRINTER without saving the file, perform (HARDCOPYW IMAGE NIL PRINTER).

If both FILE and HOST are NIL, the default action is to print the image, without saving the file. The printer used is determined by the argument PRINTERTYPE and the value of the variable DEFAULTPRINTINGHOST. If PRINTERTYPE is non-NIL, the first host on DEFAULTPRINTINGHOST of the type PRINTERTYPE is used. If PRINTERTYPE is NIL, the first printer on DEFAULTPRINTINGHOST that implements the BITMAPSCALE (as determined by PRINTERTYPES, page 29.5) operation is used, if any. Otherwise, the first printer on DEFAULTPRINTINGHOST is used.

The type of hardcopy file produced is determined by HOST if non-NIL, else by PRINTERTYPE if non-NIL, else by the value of DEFAULTPRINTINGHOST, as described above.

SCALEFACTOR is a reduction factor. If not given, it is computed automatically based on the size of the bitmap and the capabilities of the printer type. This may not be supported for some printers.

ROTATION specifies how the bitmap image should be rotated on the printed page. Most printers (including Interpress printers) only support a ROTATION of multiples of 90.

PRINTERTYPE specifies what type of printer to use when HOST is NIL. HARDCOPYW uses this information to select which printer
to use or what print file format to convert the output into, as described above.

The background menu contains a "Hardcopy" command (page 28.6) that prompts the user for a region on the screen, and sends the image to the default printer.

Hardcopy output may also be obtained by writing a file on the printer device LPT, e.g. (COPYFILE 'FOO (LPT)). When a file on this device is closed, it is converted to Interpress or some other format (if necessary) and sent to the default printer (the first host on DEFAULTPRINTINGHOST). One can include the printer name directly in the file name, e.g. (COPYFILE 'FOO (LPT)TREMOR:) will send the file to the printer TREMOR:

(PRINTERSTATUS PRINTER) [Function]
Returns a list describing the current status of the printer named PRINTER. The exact form of the value returned depends on the type of printer. For Interpress printers, the status describes whether the printer is available or busy or needs attention, and what type of paper is loaded in the printer.
Returns NIL if the printer does not respond in a reasonable time, which can occur if the printer is very busy, or does not implement the printer status service.

DEFAULTPRINTINGHOST [Variable]
The variable DEFAULTPRINTINGHOST is used to designate the default printer to be used as the output of printing operations. It should be a list of the known printer host names, for example, (QUAKE LISP:PRINT:). If an element of DEFAULTPRINTINGHOST is a list, it is interpreted as (PRINTERTYPE HOST), specifying both the host type and the host name. The type of the printer, which determines the protocol used to send to it and the file format it requires, is determined by the function PRINTERTYPE.
If DEFAULTPRINTINGHOST is a single printer name, it is treated as if it were a list of one element.

(PRINTFILETYPE FILE —) [Function]
Returns the format of the file FILE. Possible values include INTERPRESS, TEDIT, etc. If it cannot determine the file type, it returns NIL. Uses the global variable PRINTFILETYPES.

(PRINTERTYPE HOST) [Function]
Returns the type of the printer HOST. Currently uses the following heuristic: (1) If HOST is a list, the CAR is assumed to be the printer type and CADR the name of the printer; (2) If HOST is a litatom with a non-NIL PRINTERTYPE property, the property
value is returned as the printer type; (3) If HOST contains a colon (e.g., \texttt{PRINTER:PARC:XEROX}) it is assumed to be an \textsc{Interpress} printer; (4) if HOST is the \textsc{Cadr} of a list on \textsc{DEFAULTPRINTINGHOST}, the \textsc{Car} is returned as the printer type; (5) otherwise, the value of \textsc{DEFAULTPRINTERTYPE} is returned as the printer type.

29.1 Low-level Hardcopy Variables

The following variables are used to define how \textsc{Interlisp} should generate hardcopy of different types. The user should only need to change these variables when it is necessary to access a new type of printer, or define a new hardcopy document type (not often).

\begin{itemize}
  \item \textbf{PRINTERTYPES} [Variable]
  \begin{itemize}
    \item The characteristics of a given printer are determined by the value of the list \textsc{PRINTERTYPES}. Each element is a list of the form \texttt{(TYPES (PROPERTY1 VALUE1) (PROPERTY2 VALUE2) ...)}. \texttt{TYPES} is a list of the printer types that this entry addresses. The \texttt{(PROPERTYn VALUEn)} pairs define properties associated with each printer type.
    \begin{itemize}
      \item \textbf{CANPRINT} Value is a list of the file types that the printer can print directly.
      \item \textbf{STATUS} Value is a function that knows how to find out the status of the printer, used by \textsc{PRINTERSTATUS} (page 29.4).
      \item \textbf{PROPERTIES} Value is a function which returns a list of known printer properties.
      \item \textbf{SEND} Value is a function which invokes the appropriate protocol to send a file to the printer.
      \item \textbf{BITMAPSCALE} Value is a function of arguments \textit{WIDTH} and \textit{HEIGHT} in bits which returns a scale factor for scaling a bitmap.
      \item \textbf{BITMAPFILE} Value is a form which, when evaluated, converts a bitmap to a file format that the printer will accept.
    \end{itemize}
  \end{itemize}
\end{itemize}

Note: The name \texttt{8044} is defined on \textsc{PRINTERTYPES} as a synonym for the \textsc{Interpress} printer type. The names \texttt{SPRUCE}, \texttt{PENGUIN}, and \texttt{DOVER} are defined on \textsc{PRINTERTYPES} as synonyms for the \texttt{PRESS} printer type. The printer types \texttt{FULLPRESS} and \texttt{RAVEN} are also defined the same as \texttt{PRESS}, except that these printer types indicate that the printer is a "Full Press" printer that is able to scale bitmap images, in addition to the normal Press printer facilities.
The variable `PRINTFILETYPES` contains information about various file formats, such as Tedit files and Interpress files. The format is similar to `PRINTERTYPES`. The properties that can be specified include:

- **TEST**: Value is a function which tests a file if it is of the given type. Note that this function is passed an open stream.
- **CONVERSION**: Value is a property list of other file types and functions that convert from the specified type to the file format.
- **EXTENSION**: Value is a list of possible file extensions for files of this type.
30. Terminal Input/Output

30.1. Interrupt Characters

30.2. Terminal Tables

30.2.1. Terminal Syntax Classes
30.2.2. Terminal Control Functions
30.2.3. Line-Buffering

30.3. Dribble Files

30.4. Cursor and Mouse

30.4.1. Changing the Cursor Image
30.4.2. Flashing Bars on the Cursor
30.4.3. Cursor Position
30.4.4. Mouse Button Testing
30.4.5. Low Level Mouse Functions

30.5. Keyboard Interpretation

30.6. Display Screen

30.7. Miscellaneous Terminal I/O
[This page intentionally left blank]
Most input/output operations in Interlisp can be simply modeled as reading or writing on a linear stream of bytes. However, the situation is much more complex when it comes to controlling the user's "terminal," which includes the keyboard, the mouse, and the display screen. For example, Interlisp coordinates the operation of these separate I/O devices so that the cursor on the screen moves as the mouse moves, and any characters typed by the user appear in the window currently containing a flashing cursor. Most of the time, this system works correctly without need for user modification.

The purpose of this chapter is to describe how to access the low-level controls for the terminal I/O devices. It documents the use of interrupt characters, the keyboard characters that generate interrupts. Then, it describes terminal tables, used to determine the meaning of the different editing characters (character delete, line delete, etc.). Then, the "dribble file" facility that allows terminal I/O to be saved onto a file is presented (page 30.12). Finally, the low-level functions that control the mouse and cursor, the keyboard, and the screen are documented.

30.1 Interrupt Characters

Errors and breaks can be caused by errors within functions, or by explicitly breaking a function. The user can also indicate his desire to go into a break while a program is running by typing certain control characters known as "interrupt characters". The following interrupt characters are currently enabled in Interlisp-D:

Note: In Interlisp-D with multiple processes, it is not sufficient to say that "the computation" is broken, aborted, etc; it is necessary to specify which process is being acted upon. Usually, the user wants interrupts to occur in the TTY process, which is the one currently receiving keyboard input. However, sometimes the user wants to interrupt the mouse process, if it is currently busy executing a menu command or waiting for the user to specify a region on the screen. Most of the interrupt characters below take place in the mouse process if it is busy, otherwise the
INTERRUPT CHARACTERS

TTY process. Control-G can be used to break arbitrary processes. For more information, see page 23.14.

Control-B Causes a break within the mouse process (if busy) or the TTY process. Use control-G to break a particular process.

Control-D Aborts the mouse process (if busy) or the TTY process, and unwinds its stack to the top level. Calls RESET (page 14.20).

Control-E Aborts the mouse process (if busy) or the TTY process, and unwinds its stack to the last ERRORSET. Calls ERROR! (page 14.20).

Control-G Pops up a menu listing all of the currently-running processes. Selecting one of the processes will cause a break to take place in that process.

Control-P Changes the PRINTLEVEL setting of PRINTLEVEL (see page 25.11) in the TTY process. This allows the PRINTLEVEL setting to be changed dynamically, even while Interlisp is printing.

When control-P is typed, Interlisp rings the bell, prints "set printlevel to:," and waits for the user to type a series of digits. Input is terminated by a non-digit, after which the program continues.

If the input is terminated by a period or an exclamation point, the CAR printlevel is immediately set to this number, and printing continues with the (possibly new) printlevel. If the print routine is currently deeper than the new level, all unfinished lists above that level will be terminated by "-->. Thus, if a circular or long list of atoms, is being printed out, typing "control-P0." will cause the list to be terminated immediately.

If the input is terminated by a comma, another number may be typed terminated by a period or exclamation point. The CAR printlevel will then be set to the first number, the CDR printlevel to the second number.

In either case, if a period is used to terminate the printlevel setting, the printlevel will be returned to its previous setting after the current printout has finished. If an exclamation point is used, the change is permanent and the printlevel is not restored (until it is changed again).

Control-T Prints status information for the TTY process. First it prints "IO wait," "Waiting," or "Running," depending on whether the TTY process is currently in waiting for characters to be typed, waiting for some other reason, or running. Next, it prints the names of the top three frames on the stack, to show what is running. Then, it prints a line describing the percentage of time (since the last control-T) that has been spent running a program, swapping, garbage collecting, doing local disk i/o, etc. For example:

Running in TTWAITFORINPUT in TTBIN in TTYIN1
95% Util, 0% Swap, 4% GC
DELETE Clears typeahead in all processes.

The user can disable and/or redefine Interlisp interrupt characters, as well as define new interrupt characters. Interlisp-D is initialized with the following interrupt channels: RESET (control-D), ERROR (control-E), BREAK (control-B), HELP (control-G), PRINTLEVEL (control-P), RUBOUT (DELETE), and RAID. Each of these channels independently can be disabled, or have a new interrupt character assigned to it via the function INTERRUPTCHAR described below. In addition, the user can enable new interrupt channels, and associate with each channel an interrupt character and an expression to be evaluated when that character is typed.

(function)

(interruptchar char typiform hardflg —) Defines CHAR as an interrupt character. If CHAR was previously defined as an interrupt character, that interpretation is disabled.

CHAR is either a character or a character code (page 2.12). Note that full sixteen-bit NS characters can be specified as interrupt characters (see page 2.12). CHAR can also be a value returned from INTERRUPTCHAR, as described below.

If TYPIFORM = NIL, CHAR is disabled.

If TYPIFORM = T, the current state of CHAR is returned without changing or disabling it.

If TYPIFORM is one of the literal atoms RESET, ERROR, BREAK, HELP, PRINTLEVEL, RUBOUT, or RAID, then INTERRUPTCHAR assigns CHAR to the indicated Interlisp interrupt channel, (reenabling the channel if previously disabled).

If TYPIFORM is any other literal atom, CHAR is enabled as an interrupt character that when typed causes the atom TYPIFORM to be immediately set to T.

If TYPIFORM is a list, CHAR is enabled as a user interrupt character, and TYPIFORM is the form that is evaluated when CHAR is typed. The interrupt will be hard if HARDFLG = T, otherwise soft.

(interruptchar t) restores all Interlisp channels to their original state, and disables all user interrupts.

HARDFLG determines what process the interrupt should run in. If HARDFLG is NIL, the interrupt will run in the TTY process, which is the process currently receiving keyboard input. If HARDFLG is T, the interrupt will occur in whichever process happens to be running. If HARDFLG is MOUSE, the interrupt will happen in the mouse process, if the mouse is busy, otherwise in the TTY process.

INTERRUPTCHAR returns a value which, when given as the CHAR argument to INTERRUPTCHAR, will restore things as they were before the call to INTERRUPTCHAR. Therefore, INTERRUPTCHAR
can be used in conjunction with RESETFORM or RESETLST (page 14.26).

INTERRUPTCHAR is undoable.

(RESET.INTERRUPTS PERMITTEDINTERRUPTS SAVECURRENT?)  [Function]
PERMITTEDINTERRUPTS is a list of interrupt character settings to
be performed, each of the form (CHAR TYPFORM HARDFLG).
The effect of RESET.INTERRUPTS is as if (INTERRUPTCHAR CHAR
TYPFORM HARDFLG) were performed for each item on
PERMITTEDINTERRUPTS, and (INTERRUPTCHAR OTHERCHAR NIL)
were performed on every other existing interrupt character.

If SAVECURRENT? is non-NIL, then RESET.INTERRUPTS returns
the current state of the interrupts in a form that could be passed
to RESET.INTERRUPTS, otherwise it returns NIL. This can be used
with a RESET.INTERRUPTS that appears in a RESETFORM, so that
the list is built at "entry", but not upon "exit".

(LISPINTERRUPTS)  [Function]
Returns the initial default interrupt character settings for
Interlisp-D, as a list that RESET.INTERRUPTS would accept.

(INTERRUPTABLE FLAG)  [Function]
if FLAG = NIL, turns interrupts off. If FLAG = T, turns interrupts
on. Value is previous setting. INTERRUPTABLE compiles open.

Any interrupt character typed while interrupts are off is treated
the same as any other character, i.e. placed in the input buffer,
and will not cause an interrupt when interrupts are turned back
on.

30.2 Terminal Tables

A read table (page 25.33) contains input/output information that
is media-independent. For example, the action of parentheses is
the same regardless of the device from which the input is being
performed. A terminal table is an object that contains
information that pertains to terminal input/output operations
only, such as the character to type to delete the last character or
to delete the last line. In addition, terminal tables contain such
information as how line-buffering is to be performed, how
control characters are to be echoed/printed, whether lower case
input is to be converted to upper case, etc.

Using the functions below, the user may change, reset, or copy
terminal tables, or create a new terminal table and install it as
the primary terminal table via SETTERMTABLE. However, unlike
read tables, terminal tables cannot be passed as arguments to input/output functions.

**GETTERMTABLE TTBL**  
If TTBL = NIL, returns the primary (i.e., current) terminal table. If TTBL is a terminal table, return TTBL. Otherwise, generates an **ILLEGAL TERMINAL TABLE** error.

**COPYTERMTABLE TTBL**  
Returns a copy of TTBL. TTBL can be a real terminal table, NIL (copies the primary terminal table), or ORIG (returns a copy of the original system terminal table). Note that COPYTERMTABLE is the only function that creates a terminal table.

**SETTERMTABLE TTBL**  
Sets the primary terminal table to be TTBL. Returns the previous primary terminal table. Generates an **ILLEGAL TERMINAL TABLE** error if TTBL is not a real terminal table.

**RESETTERMTABLE TTBL FROM**  
Copies (smashes) FROM into TTBL. FROM and TTBL can be NIL or a real terminal table. In addition, FROM can be ORIG, meaning to use the system's original terminal table.

**TERMTABLEP TTBL**  
Returns TTBL, if TTBL is a real terminal table, NIL otherwise.

### 30.2.1 Terminal Syntax Classes

A terminal table associates with each character a single "terminal syntax class", one of CHARDELETE, LINEDELETE, WORDDELETE, RETYPE, CTRLV, EOL, and NONE. Unlike read table classes, only one character in a particular terminal table can belong to each of the classes (except for the default class NONE). When a new character is assigned one of these syntax classes by SETSYNTAX (page 25.37), the previous character is disabled (i.e., reassigned the syntax class NONE), and the value of SETSYNTAX is the code for the previous character of that class, if any, otherwise NIL.

The terminal syntax classes are interpreted as follows:

- **CHARDELETE** (Initially BackSpace and control-A in Interlisp-D) Typing this character deletes the previous character typed. Repeated use of this character deletes successive characters back to the beginning of the line.

- **LINEDELETE** (Initially control-Q in Interlisp-D) Typing this character deletes the whole line; it cannot be used repeatedly.
WORDDELETE

(Initially control-W in Interlisp-D) Typing this character deletes the previous "word", i.e., sequence of non-separator characters.

RETYPE

(Initially control-R) Causes the line to be retyped as Interlisp sees it (useful when repeated deletions make it difficult to see what remains).

CTRLV

(Initially control-V) When followed by A, B, ..., Z, inputs the corresponding control character control-A, control-B, ..., control-Z. This allows interrupt characters to be input without causing an interrupt.

EOL

On input from a terminal, the EOL character signals to the line buffering routine to pass the input back to the calling function. It also is used to terminate inputs to READLINE (page 13.36). In general, whenever the phrase carriage-return linefeed is used, what is meant is the character with terminal syntax class EOL.

NONE

The terminal syntax class of all other characters.

GETSYNTAX, SETSYNTAX, and SYNTAXP all work on terminal tables as well as read tables (see page 25.36). As with read tables, full sixteen-bit NS characters can be specified in terminal tables (see page 2.12). When given NIL as a TABLE argument, GETSYNTAX and SYNTAXP use the primary read table or primary terminal table depending on which table contains the indicated CLASS argument. For example, (SETSYNTAX CH 'BREAK) refers to the primary read table, and (SETSYNTAX CH 'CHARDELETE) refers to the primary terminal table. In the absence of such information, all three functions default to the primary read table; e.g., (SETSNTAX ' [ ']) refers to the primary read table. If given incompatible CLASS and table arguments, all three functions generate errors. For example, (SETSNTAX CH 'BREAK TTBL), where TTBL is a terminal table, generates an ILLEGAL READTABLE error, and (GETSYNTAX 'CHARDELETE RDTBL) generates an ILLEGAL TERMINAL TABLE error.

30.2.2 Terminal Control Functions

(ECHOCHAR CHARCODE MODE TTBL)

[Function]

ECHOCHAR sets the "echo mode" of the character CHARCODE to MODE in the terminal table TTBL. The "echo mode" determines how the character is to be echoed or printed. Note that although the name of this function suggests echoing only, it affects all output of the character, both echoing of input and printing of output.

CHARCODE should be a character code. CHARCODE can also be a list of characters, in which case ECHOCHAR is applied to each of
them with arguments MODE and TTBL. Note that echo modes can be specified for full sixteen-bit NS characters (see page 2.12).

MODE should be one of the litatoms IGNORE, REAL, SIMULATE, or INDICATE which specify how the character should be echoed or printed:

- **IGNORE** CHARCODE is never printed.
- **REAL** CHARCODE itself is printed. Some terminals may respond to certain control and meta characters in interesting ways.
- **SIMULATE** Output of CHARCODE is simulated. For example, control-I (tab) may be simulated by printing spaces. The simulation is machine-specific and beyond the control of the user.
- **INDICATE** For control or meta characters, CHARCODE is printed as # and/or ↑ followed by the corresponding alphabetic character. For example, control-A would echo as ↑A, and meta-control-W would echo as #↑W.

The value of ECHOCHAR is the previous echo mode for CHARCODE. If MODE = NIL, ECHOCHAR returns the current echo mode without changing it.

Warning: In some fonts, control and meta characters may be used for printable characters. If the echomode is set to INDICATE for these characters, they will not print out correctly.

### ECHOCONTROL CHAR MODE TTBL

ECHOCONTROL is an old, limited version of ECHOCHAR, that can only specify the echo mode of control characters. CHAR is a character or character code. If CHAR is an alphabetic character (or code), it refers to the corresponding control character, e.g., (ECHOCONTROL 'Z 'INDICATE) if equivalent to (ECHOCHAR (CHARCODE ↑ Z) 'INDICATE).

### ECHOMODE FLG TTBL

If FLG = T, turns echoing for terminal table TTBL on. If FLG = NIL, turns echoing off. Returns the previous setting.

Note: Unlike ECHOCHAR, this only affects echoing of typed-in characters, not printing of characters.

### GETECHOMODE TTBL

Returns the current echo mode for TTBL.

The following functions manipulate the "raise mode," which determines whether lower case characters are converted to upper case when input from the terminal. There is no "raise mode" for input from files.
(RAISE FLG TTBL)  [Function]
Sets the RAISE mode for terminal table TTBL. If FLG = NIL, all characters are passed as typed. If FLG = T, input is echoed as typed, but lowercase letters are converted to upper case. If FLG = 0, input is converted to upper case before it is echoed. Returns the previous setting.

(GETRAISE TTBL)  [Function]
Returns the current RAISE mode for TTBL.

(DELETECONTROL TYPE MESSAGE TTBL)  [Function]
Specifies the output protocol when a CHARDELETE or LNEDELET is typed, by specifying character strings to print when characters are deleted.

Interlisp-10 (designed for use on hardcopy terminals) echoes the characters being deleted, preceding the first by a \ and following the last by a \, so that it is easy to see exactly what was deleted. Note: Interlisp-D is initially set up to physically erase the deleted characters from the display, so the DELETECONTROL strings are initialized to the null string.

The various values of TYPE specify different phases of the deletion, as follows:

1STCHDEL MESSAGE is the message printed the first time CHARDELETE is typed. Initially "\" in Interlisp-10.

NTHCHDEL MESSAGE is the message printed when the second and subsequent CHARDELETE characters are typed (without intervening characters). Initially "" in Interlisp-10.

POSTCHDEL MESSAGE is the message printed when input is resumed following a sequence of one or more CHARDELETE characters. Initially "\" in Interlisp-10.

EMPTYCHDEL MESSAGE is the message printed when a CHARDELETE is typed and there are no characters in the buffer. Initially "##cr" in Interlisp-10.

ECHO If TYPE = ECHO, the characters deleted by CHARDELETE are echoed. MESSAGE is ignored.

NOECHO If TYPE = NOECHO, the characters deleted by CHARDELETE are not echoed. MESSAGE is ignored.

LNEDELETE MESSAGE is the message printed when the LNEDELETE character is typed. Initially "##cr".

Note: In Interlisp-10, the LNEDELETE, 1STCHDEL, NTHCHDEL, POSTCHDEL, and EMPTYCHDEL messages must be 4 characters or fewer in length.

DELETECONTROL returns the previous message as a string. If MESSAGE = NIL, the value returned is the previous message.
without changing it. For $\text{TYPE} = \text{ECHO}$ and $\text{NOECHO}$, the value of $\text{DELETECONTROL}$ is the previous echo mode, i.e., $\text{ECHO}$ or $\text{NOECHO}$.

### 30.2.3 Line-Buffering

Characters typed at the terminal are stored in two buffers before they are passed to an input function. All characters typed in are put into the low-level "system buffer", which allows type-ahead. When an input function is entered, characters are transferred to the "line buffer" until a character with terminal syntax class $\text{EOL}$ appears (or, for calls from $\text{READ}$, when the count of unbalanced open parentheses reaches 0). Note that $\text{PEEKC}$ is an exception; it returns the character immediately when its second argument is $\text{NIL}$. Until this time, the user can delete characters one at a time from the line buffer by typing the current $\text{CHARDELETE}$ character, or delete the entire line buffer back to the last carriage-return by typing the current $\text{LINEDELETE}$.

Note that this line editing is not performed by $\text{READ}$ or $\text{RATOM}$, but by Interlisp, i.e., it does not matter (nor is it necessarily known) which function will ultimately process the characters, only that they are still in the Interlisp line buffer. However, the function that is requesting input at the time the buffering starts does determine whether parentheses counting is observed. For example, if a program performs $(\text{PROGN} \ (\text{RATOM}) \ (\text{READ}))$ and the user types in "$\text{A (B C D)}$", the user must type in the carriage-return following the right parenthesis before any action is taken, because the line buffering is happening under $\text{RATOM}$. If the program had performed $(\text{PROGN} \ (\text{READ}) \ (\text{READ}))$, the line-buffering would be under $\text{READ}$, so that the right parenthesis would terminate line buffering, and no terminating carriage-return would be required.

Once a carriage-return has been typed, the entire line is "available" even if not all of it is processed by the function initiating the request for input. If any characters are "left over", they are returned immediately on the next request for input. For example, $(\text{LIST} \ (\text{RATOM}) \ (\text{READC}) \ (\text{RATOM}))$ when the input is "$\text{ABr}$" returns the three-element list $\text{(A % B)}$ and leaves the carriage-return in the buffer.

If a carriage-return is typed when the input under $\text{READ}$ is not "complete" (the parentheses are not balanced or a string is in progress), line buffering continues, but the lines completed so
far are not available for editing with CHARDELETE or LINEDELETE.
The function CONTROL is available to defeat line-buffering:

(Control mode TTLB) [Function]
If MODE = T, eliminates Interlisp's normal line-buffering for the
terminal table TTLB. If MODE = NIL, restores line-buffering
(normal). When operating with a terminal table in which
(CONTROL T) has been performed, characters are returned to the
calling function without line-buffering as described below.
CONTROL returns its previous setting.

(GETCONTROL TTLB) [Function]
Returns the current control mode for TTLB.

The function that initiates the request for input determines how
the line is treated when (CONTROL T) is in effect:

READ
If the expression being typed is a list, the effect is the same as
though done with (CONTROL NIL), i.e., line-buffering continues
until a carriage-return or matching parentheses. If the
expression being typed is not a list, it is returned as soon as a
break or separator character is encountered, e.g., (READ) when
the input is "ABC<space>" immediately returns ABC.
CHARDELETE and LINEDELETE are available on those characters
still in the buffer. Thus, if a program is performing several reads
under (CONTROL T), and the user types "NOW IS THE TIME"
followed by control-Q, only TIME is deleted, since the rest of the
line has already been transmitted to READ and processed.

An exception to the above occurs when the break or separator
character is an opening parenthesis, bracket or double-quote,
since returning at this point would leave the line buffer in a
"funny" state. Thus if the input to (READ) is "ABC (", the ABC is
not read until a carriage-return or matching parentheses is
encountered. In this case the user could LINEDELETE the entire
line, since all of the characters are still in the buffer.

RATOM
Characters are returned as soon as a break or separator character
is encountered. Until then, LINEDELETE and CHARDELETE may
be used as with READ. For example, (RATOM) followed by
"ABC<control-A><space>" returns AB (RATOM) followed by
"(<control-A>)" returns ( and types ## indicating that
control-A was attempted with nothing in the buffer, since the ( is
a break character and would therefore already have been read.

READC
PEEKC
The character is returned immediately; no line editing is possible.
In particular, (READC) is perfectly happy to return the
CHARDELETE or LINEDELETE characters, or the ESCAPE character (%).

The system buffer and line buffer can be directly manipulated using the following functions.

(CLEARBUF FILE FLG)  [Function]
Clears the input buffer for FILE. If FILE is T and FLG is T, the contents of Interlisp's system buffer and line buffer are saved (and can be obtained via SYSBUF and LINBUF described below).

When control-D or control-E is typed, or any of the interrupt characters that require terminal interaction is typed (control-G, or control-P), Interlisp automatically performs (CLEARBUF T T).

For control-P and, when the break is exited normally, control-H, Interlisp restores the buffer after the interaction.

The action of (CLEARBUF T), i.e., clearing of typeahead, is also available as the RUBOUT interrupt character, initially assigned to the delete key in Interlisp-O. Note that this interrupt clears both buffers at the time it is typed, whereas the action of the CHARDELETE and LINEDELETE character occur at the time they are read.

(SYSBUF FLG)  [Function]
If FLG = T, returns the contents of the system buffer (as a string) that was saved at the last (CLEARBUF T T). If FLG = NIL, clears this internal buffer.

(LINBUF FLG)  [Function]
Same as SYSBUF for the line buffer.

If both the system buffer and Interlisp's line buffer are empty, the internal buffers associated with LINBUF and SYSBUF are not changed by a (CLEARBUF T T).

(BKSYSBUF X FLG RDTBL)  [Function]
BKSYSBUF sets the system buffer to the PRIN1-name of X. The effect is the same as though the user typed X. Some implementations have a limit on the length of X, in which case characters in X beyond the limit are ignored. Returns X.

If FLG is T, then the PRIN2-name of X is used, computed with respect to the read table RDTBL.

Note that if the user is typing at the same time as the BKSYSBUF is being performed, the relative order of the type-in and the characters of X is unpredictable.

Compatibility note: Some implementations of BKSYSBUF (Interlisp-10) use a "system" buffer, from which keyboard
interrupts are also processed. In this case, BKSYSBUF of an interrupt character actually invokes the interrupt at some (asynchronous) time after the BKSYSBUF is initiated. In other implementations (Interlisp-D), the characters are not processed for interrupts, and it is possible to BKSYSBUF characters which would otherwise be impossible to type.

(BKLINBUF STR) [Function]

STR is a string. BKLINBUF sets Interlisp's line buffer to STR. Some implementations have a limit on the length of STR, in which case characters in STR beyond the limit are ignored. Returns STR.

BKLINBUF, BKSYSBUF, LINBUF, and SYSBUF provide a way of "undoing" a CLEARBUF. Thus to "peek" at various characters in the buffer, one could perform (CLEARBUF T T), examine the buffers via LINBUF and SYSBUF, and then put them back.

The more common use of these functions is in saving and restoring typeahead when a program requires some unanticipated (from the user's standpoint) input. The function RESETBUFS provides a convenient way of simply clearing the input buffer, performing an interaction with the user, and then restoring the input buffer.

(RESETBUFS FORM₁ FORM₂ ... FORMₙ) [Nlambda NoSpread Function]

Clears any typeahead (ringing the terminal's bell if there was, indeed, typeahead), evaluates FORM₁, FORM₂,... FORMₙ, then restores the typeahead. Returns the value of FORMₙ. Compiles open.

30.3 Dribble Files

A dribble file is a "transcript" of all of the input and output on a terminal. In Interlisp-D, DRIBBLE opens a dribble file for the current process, recording the terminal input and output for that process. Multiple processes can have separate dribble files open at the same time.

(DRIBBLE FILE APPENDFLG THAWEDFLG) [Function]

Opens FILE and begins recording the typescript. Returns the old dribble file if any, otherwise NIL. If APPENDFLG = T, the typescript will be appended to the end of FILE. If THAWEDFLG = T, the file will be opened in "thawed" mode, for those implementations that support it. (DRIBBLE) closes the dribble file for the current process. Only one dribble file can be
active for each process at any one time, so (DRIBBLE FILE1) followed by (DRIBBLE FILE2) will cause FILE1 to be closed.

(DRIBBLEFILE) [Function]
Returns the name of the current dribble file for the current process, if any, otherwise NIL.

Terminal input is echoed to the dribble file a line buffer at a time. Thus, the typescript produced is somewhat neater than that appearing on the user's terminal, because it does not show characters that were erased via control-A or control-Q. Note that the typescript file is not included in the list of files returned by (OPENP), nor will it be closed by a call to CLOSEALL or CLOSEF. Only (DRIBBLE) closes the typescript file.

30.4 Cursor and Mouse

A mouse is a small box connected to the computer keyboard by a long wire. On the top of the mouse are two or three buttons. On the bottom is a rolling ball or a set of photoreceptors, to detect when the mouse is moved. As the mouse is moved on a surface, a small image on the screen, called the cursor, moves to follow the movement of the mouse. By moving the mouse, the user can cause the cursor to point to any part of the display screen.

The mouse and cursor are an important part of the Interlisp-D user interface. The Interlisp-D window system allows the user to create, move, and reshape windows, and to select items from displayed menus, all by moving the mouse and clicking the mouse buttons. This section describes the low-level functions used to control the mouse and cursor.

30.4.1 Changing the Cursor Image

Interlisp-D maintains the image of the cursor on the screen, moving it as the mouse is moved. The bitmap that becomes visible as the cursor can be accessed by the following function:

(CURSORBITMAP) [Function]
Returns the cursor bitmap.
CURSORWIDTH  [Variable]

CURSORHEIGHT  [Variable]

Value is the width and height of the cursor bitmap, respectively.

The cursor bitmap can be changed like any other bitmap by BITBLTing into it or pointing a display stream at it and printing or drawing curves. However, for some applications it is necessary to save and restore the cursor, which can be most easily done using CURSOR record objects. A CURSOR record contains fields CURSORBITMAP and CURSORHOTSPOT. The value of the CURSORBITMAP field is a bitmap that is CURSORWIDTH bits wide by CURSORHEIGHT high. The value of the CURSORHOTSPOT field is the "hot spot" of the cursor, a position in the bitmap interpreted as the point that the cursor is pointing to. CURSOR objects can be saved on a file using the file package command CURSORS, or the UGLYVARS file package command.

(CURSORCREATE BITMAP X Y)  [Function]

Returns a cursor object which has BITMAP as its image and the location (X,Y) as the hot spot. If X is a POSITION, it is used as the hot spot. If BITMAP has dimensions different from CURSORWIDTH by CURSORHEIGHT, the lesser of the widths and the lesser of the heights are used to determine the bits that actually get copied into the lower left corner of the cursor. If X is NIL, 0 is used. If Y is NIL, CURSORHEIGHT-1 is used. The default cursor is an uparrow with its tip in the upper left corner and its hot spot at (0,CURSORHEIGHT-1).

(CURSOR NEWCURSOR —)  [Function]

Returns a CURSOR record instance that contains (a copy of) the current cursor specification. If NEWCURSOR is a CURSOR record instance, the cursor will be set to the values in NEWCURSOR. If NEWCURSOR is T, the cursor will be set to the default cursor DEFAULTCURSOR, an upward left pointing arrow:

(SETCURSOR NEWCURSOR —)  [Function]

If NEWCURSOR is a CURSOR record instance, the cursor will be set to the values in NEWCURSOR. This does not return the old cursor, and therefore, provides a way of changing the cursor without using storage.

(FLIPCURSOR)  [Function]

Inverts the cursor.
The following list describes the cursors used by the Interlisp-D system. Most of them are stored as the values of various variables.

In variable **DEFAULTCURSOR**. This is the default cursor.

In variable **WAITINGCURSOR**. Represents an hourglass. Used during long computations.

In variable **MOUSECONFIRMCURSOR**. Indicates that the system is waiting for the user to confirm an action by pressing the left mouse button, or aborting the action by pressing any other button. Used by the function **MOUSECONFIRM** (page 28.11).

In variable **SYSOUTCURSOR**. Indicates that the system is saving the virtual memory in a sysout file. See **SYSOUT**, page 12.8.

In variable **SAVINGCURSOR**. Indicates that **SAVEVM** has been called automatically to save the virtual memory state after the system is idle for long enough. See **SAVEVMWAIT**, page 12.7.

In variable **CROSSHAIRS**. Used by **GETPOSITION** (page 28.9) to indicate a position.

In variable **BOXCURSOR**. Used by **GETBOXPOSITION** (page 28.9) to indicate where to place the corner of a box.

In variable **FORCEPS**. Used by **GETREGION** (page 28.10) when the user switches corners.

In variable **EXPANDINGBOX**. Used by **GETREGION** (page 28.10) when a box is first displayed.

In variable **UpperRightCursor**.

In variable **LowerRightCursor**.

In variable **UpperLeftCursor**.

In variable **LowerLeftCursor**.

The previous four cursors are used by **GETREGION** (page 28.10) to indicate the four corners of a region.

In variable **VertThumbCursor**. Used during scrolling to indicate thumbing in a vertical scroll bar.

In variable **VertScrollCursor**.

In variable **ScrollUpCursor**.

In variable **ScrollDownCursor**.

The previous four cursors are used by **SCROLL.HANDLER** (page 28.24) during vertical scrolling.
In variable HorizThumbCursor. Used during scrolling to indicate thumbing in a horizontal scroll bar.

In variable HorizScrollCursor.

In variable ScrollLeftCursor.

In variable ScrollRightCursor.

The previous four cursors are used by SCROLL.HANDLER (page 28.24) during horizontal scrolling.

These cursors are used by the Teleraid low-level debugger. These cursors are not accessible as standard Interlisp-D cursors.

### 30.4.2 Flashing Bars on the Cursor

The low-level Interlisp-D system uses the cursor to display certain system status information, such as garbage collection or swapping. This is done because changing the cursor image is very quick, and does not require interacting with the window system. Interlisp inverts horizontal bars on the cursor when the system is swapping pages, or doing certain stack operations. Normally, these bars are only inverted for a very short time, so they look like they are flashing. These cursor changes are interpreted as follows:

- **Inverted cursor:** Whatever image is being displayed as the cursor, whenever Interlisp does a garbage collection, the whole cursor is inverted.

- **Top bar:** Swap read. On when Interlisp is swapping in a page from the virtual memory file into the real memory. It is also on when Interlisp allocates a new virtual memory page, even though that doesn't involve a disk read. If this is flashing a lot, the system is doing a lot of swapping. This is an indication that the virtual memory working set is fragmented (see page 22.1). Performance may be improved by reloading a clean Interlisp system.

- **Upper middle bar:** Stack operations. If this is flashing a lot, it suggests that some process is neglecting to release stack pointers in a timely fashion (see page 11.9).

- **Lower middle bar:** Stack operations. On when Interlisp is moving frames on the stack. If the system is slow, and this is flashing a lot, HARDRESET (page 23.1) sometimes helps.

- **Bottom bar:** Swap write. On when Interlisp writes a dirty virtual memory page from the real memory back into the virtual memory file.
30.4.3 Cursor Position

The position at which the cursor bitmap is being displayed can be read or set using the following functions:

\[(\text{CURSORPOSITION NEWPOSITION DISPLAYSTREAM OLDPOSITION})\]

[Function]

Returns the location of the cursor in the coordinate system of DISPLAYSTREAM (or the current display stream, if DISPLAYSTREAM is NIL). If NEWPOSITION is non-NIL, it should be a position and the cursor will be positioned at NEWPOSITION. If NEWPOSITION is NIL, the current position is simple returned.

Note: The current position of the cursor is the position of the "hot spot" of the cursor, not the position of the cursor bitmap.

If OLDPOSITION is a POSITION object, this object will be changed to point to the location of the cursor and returned, rather than allocating a new POSITION. This can improve performance if CURSORPOSITION is called repeatedly to track the cursor.

Note: To get the location of the cursor in absolute screen coordinates, use the variables LASTMOUSEX and LASTMOUSEY (page 30.18).

\[(\text{ADJUSTCURSORPOSITION DELTAX DELTAY})\]

[Function]

Moves the cursor DELTAX points in the X direction and DELTAY points in the Y direction. DELTAX and DELTAY default to 0.

30.4.4 Mouse Button Testing

There are two or three keys on the mouse. These keys (also called buttons) are referred to by their location: LEFT, MIDDLE, or RIGHT. The following macros are provided to test the state of the mouse buttons:

\[(\text{MOUSESTATE BUTTONFORM})\]

[Macro]

Reads the state of the mouse buttons, and returns T if that state is described by BUTTONFORM. BUTTONFORM can be one of the key indicators LEFT, MIDDLE, or RIGHT; the atom UP (indicating all keys are up); the form (ONLY KEY); or a form of AND, OR, or NOT applied to any valid button form.

For example: (MOUSESTATE LEFT) will be true if the left mouse button is down. (MOUSESTATE (ONLY LEFT)) will be true if the left mouse button is the only one down. (MOUSESTATE (OR (NOT LEFT) MIDDLE)) will be true if either the left mouse button is up or the middle mouse button is down.
(LASTMOUSESTATE BUTTONFORM) [Macro]
Similar to MOUSESTATE, but tests the value of LASTMOUSEBUTTONS (below) rather than getting the current state. This is useful for determining which keys caused MOUSESTATE to be true.

(UNTILMOUSESTATE BUTTONFORM INTERVAL) [Macro]
BUTTONFORM is as described in MOUSESTATE. Waits until BUTTONFORM is true or until INTERVAL milliseconds have elapsed. The value of UNTILMOUSESTATE is T if BUTTONFORM was satisfied before it timed out, otherwise NIL. If INTERVAL is NIL, it waits indefinitely. This compiles into an open loop that calls the TTY wait background function. This form should not be used inside the TTY wait background function. UNTILMOUSESTATE does not use any storage during its wait loop.

30.4.5 Low Level Mouse Functions
This section describes the functions and variables that provide low level access to the mouse and cursor.

(LASTMOUSEX DISPLAYSTREAM) [Function]
Returns the value of the cursor's X position in the coordinates of DISPLAYSTREAM (as of the last call to GETMOUSESTATE, below).

(LASTMOUSEY DISPLAYSTREAM) [Function]
Returns the value of the cursor's Y position in the coordinates of DISPLAYSTREAM (as of the last call to GETMOUSESTATE, below).

LASTMOUSEX [Variable]
Value is the X position of the cursor in absolute screen coordinates (as of the last call to GETMOUSESTATE, below).

LASTMOUSEY [Variable]
Value is the Y position of the cursor in absolute screen coordinates (as of the last call to GETMOUSESTATE, below).

LASTMOUSEBUTTONS [Variable]
Value is an integer that has bits on corresponding to the mouse buttons that are down (as of the last call to GETMOUSESTATE, below). Bit 4Q is the left mouse button, 2Q is the right button, 1Q is the middle button.
LASTKEYBOARD

Value is an integer encoding the state of certain keys on the keyboard (as of the last call to GETMOUSESTATE, below). Bit 200Q = lock, 100Q = left shift, 40Q = ctrl, 10Q = right shift, 4Q = blank Bottom, 2Q = blank Middle, 1Q = blank Top. If the key is down, the corresponding bit is on.

(GETMOUSESTATE)

Reads the current state of the mouse and sets the variables LASTMOUSEX, LASTMOUSEY, and LASTMOUSEBUTTONS. In polling mode, the program must remember the previous state and look for changes, such as a key going up or down, or the cursor moving outside a region of interest.

(DECODEBUTTONS BUTTONSTATE)

Returns a list of the mouse buttons that are down in the state BUTTONSTATE. If BUTTONSTATE is not a small integer, the value of LASTMOUSEBUTTONS (above) is used. The button names that can be returned are: LEFT, MIDDLE, RIGHT (the three mouse keys).

30.5 Keyboard Interpretation

For each key on the keyboard and mouse there is a corresponding bit in memory that the hardware turns on and off as the key moves up and down. System-level routines decode the meaning of key transitions according to a table of "key actions", which may be to put particular character codes in the sysbuffer, cause interrupts, change the internal shift/control status, or create events to be placed in the mouse buffer.

(KEYDOWNP KEYNAME)

Used to read the instantaneous state of any key, independent of any buffering or pre-assigned key action. Returns T if the key named KEYNAME is down at the moment the function is executed.

Most keys are named by the characters on the key-top. Therefore, (KEYDOWNP 'a) or (KEYDOWNP 'A) returns T if the "A" key is down.

There are a number of keys that do not have standard names printed on them. These can be accessed by special names as follows:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Special Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>SPACE</td>
</tr>
<tr>
<td>Carriage return</td>
<td>CR</td>
</tr>
</tbody>
</table>
KEYBOARD INTERPRETATION

Line-feed LF
Backspace BS
Tab TAB
Blank keys on 1132 The 1132 keyboard has three unmarked keys on the right of the normal keyboard. These can be accessed by BLANK-BOTTOM, BLANK-MIDDLE, and BLANK-TOP.
Escape ESCAPE
Shift keys LSHIFT for the left shift key, RSHIFT for the right shift key.
Shift lock key LOCK
Control key CTRL
Mouse buttons The state of the mouse buttons can be accessed using LEFT, MIDDLE, and RIGHT.

*(SHIFTDOWNP SHIFT)*

[Function]

Returns T if the internal "shift" flag specified by *SHIFT* is on; NIL otherwise.

If *SHIFT* = 1SHIFT, 2SHIFT, LOCK, META, or CTRL, SHIFTDOWNP returns the state of the left shift, right shift, shift lock, control, and meta flags, respectively.

If *SHIFT* = SHIFT, SHIFTDOWNP returns T if either the left or right shift flag is on.

If *SHIFT* = USERMODE1, USERMODE2, or USERMODE3, SHIFTDOWNP returns the state of one of three user-settable flags that have no other effect on key interpretation. These flags can be set or cleared on character transitions by using KEYACTION (below).

*(KEYACTION KEYNAME ACTIONS —)*

[Function]

Changes the internal tables that define the action to be taken when a key transition is detected by the system keyboard handler. *KEYNAME* is specified as for KEYDOWNP. *ACTIONS* is a dotted pair of the form (DOWN-ACTION , UP-ACTION), where the acceptable transition actions and their interpretations are:

**NIL**
Take no action on this transition (the default for up-transitions on all ordinary characters).

**IGNORE**
Take no action on this transition (the default for up-transitions on all ordinary characters).

*(CHAR SHIFTEDECHAR LOCKFLAG)*

If a transition action is a three-element list, CHAR and SHIFTEDECHAR are either character codes or (non-numeric) single-character litatoms standing for their character codes. Note that CHAR and SHIFTEDECHAR can be full sixteen-bit NS characters (see page 2.12). When the transition occurs, CHAR or SHIFTEDECHAR is transmitted to the system buffer, depending on whether either of the two shift keys are down.
LOCKFLAG is optional, and may be LOCKSHIFT or NOLOCKSHIFT. If LOCKFLAG is LOCKSHIFT, then SHIFTEDCHAR will also be transmitted when the LOCK shift is down (the alphabetic keys initially specify LOCKSHIFT, but the digit keys specify NOLOCKSHIFT). For example, (a A LOCKSHIFT) and (61Q ! NOLOCKSHIFT) are the initial settings for the down transitions of the "a" and "1" keys respectively.

1SHIFTUP, 1SHIFTDOWN
2SHIFTUP, 2SHIFTDOWN
CTRLUP, CTRLDOWN
METAUP, METADOWN

Change the status of the internal "shift" flags for the left shift, right shift, control, and meta keys, respectively. These shifts affect the interpretation of ordinary key actions. If either of the shifts is down, then SHIFTEDCHARs are transmitted. If the control flag is on, then the seventh bit of the character code is cleared as characters are transmitted. If the meta flag is on, then the eighth bit of the character code is set (normallly cleared) as characters are transmitted. For example, the initial keyactions for the left shift key is (1SHIFTDOWN. 1SHIFTUP).

LOCKUP, LOCKDOWN, LOCKTOGGLE

Change the status of the internal "shift" flags for the shift lock key. If the lock flag is down, then SHIFTEDCHARs are transmitted if the key action specified LOCKSHIFT. LOCKUP and LOCKDOWN clear and set the shift lock flag, respectively. LOCKTOGGLE complements the flag (turning it off if the flag is on; on if the flag is off).

USERMODE1UP, USERMODE1DOWN, USERMODE1TOGGLE
USERMODE2UP, USERMODE2DOWN, USERMODE2TOGGLE
USERMODE3UP, USERMODE3DOWN, USERMODE3TOGGLE

Change the status of the three user flags USERMODE1, USERMODE2, and USERMODE3, whose status can be determined by calling SHIFTDOWNP (above). These flags have no other effect on key interpretation.

EVENT

An encoding of the current state of the mouse and selected keys is placed in the mouse-event buffer when this transition is detected.

KEYACTION returns the previous setting for KEYNAME. If ACTIONS is NIL, returns the previous setting without changing the tables.

(MODIFY.KEYACTIONS KEYACTIONS SAVECURRENT?) [Function]

KEYACTIONS is a list of key actions to be set, each of the form (KEYNAME . ACTIONS). The effect of MODIFY.KEYACTIONS is as if (KEYACTION KEYNAME ACTIONS) were performed for each item on KEYACTIONS.

If SAVECURRENT? is non-NIL, then MODIFY.KEYACTIONS returns a list of all the results from KEYACTION, otherwise it returns NIL.
This can be used with a MODIFY.KEYACTIONS that appears in a RESETFORM, so that the list is built at "entry", but not upon "exit".

(METASHIFT FLG) [NoSpread Function]  
If FLG is T, changes the keyboard handler (via KEYACTION) so as to interpret the "stop" key on the 1108 as a metashift: if a key is struck while the meta is down, it is read with the 200Q bit set. For CHAT users this is a way of getting an "Edit" key on your simulated Datamedia.

If FLG is other than NIL or T, it is passed as the ACTIONS argument to KEYACTION. The reason for this is that if someone has set the "STOP" key to some random behavior, then (RESETFORM (METASHIFT T) _) will correctly restore that random behavior.

30.6 Display Screen

Interlisp-D supports a high-resolution bitmap display screen. All printing and drawing operations to the screen are actually performed on a bitmap in memory, which is read by the computer hardware to become visible as the screen. This section describes the functions used to control the appearance of the display screen.

(SCREENBITMAP)  [Function]  
Returns the screen bitmap.

SCREENWIDTH  [Variable]  
Value is the width and height of the screen bitmap, respectively.

SCREENHEIGHT  [Variable]  
Value is the width and height of the screen bitmap, respectively.

WHOLEDISPLAY  [Variable]  
Value is a region that is the size of the screen bitmap.

The background shade of the display window can be changed using the following function:

(CHANGEBACKGROUND SHADE _)  [Function]  
Changes the background shade of the window system. SHADE determines the pattern of the background. If SHADE is a texture, then the background is simply painted with it. If SHADE is a BITMAP, the background is tesselated (tiled) with it to cover the
screen. If SHADE is T, it changes to the original shade, the value of WINDOWBACKGROUND SHADE. It returns the previous value of the background.

(CHANGEBACKGROUND BORDER SHADE —) [Function]
On the Xerox 1108, changes the shade of the border of the display to SHADE, which should be a texture. It returns the previous texture of the background border.
CHANGEBACKGROUND BORDER is a no-op on the Xerox 1132.

WINDOWBACKGROUND SHADE [Variable]
Value is the default background shade for the display.

(VIDEOCOLOR BLACKFLG) [NoSpread Function]
Sets the interpretation of the bits in the screen bitmap. If BLACKFLG is NIL, a 0 bit will be displayed as white, otherwise a 0 bit will be displayed as black. VIDEOCOLOR returns the previous setting. If BLACKFLG is not given, VIDEOCOLOR will return the current setting without changing anything.

Note: This function only works on the Xerox 1100 and Xerox 1108.

(VIDEORATE TYPE) [Function]
Sets the rate at which the screen is refreshed. TYPE is one of NORMAL or TAPE. If TYPE is TAPE, the screen will be refreshed at the same rate as TV (60 cycles per second). This makes the picture look better when video taping the screen. Note: Changing the rate may change the dimensions of the display on the picture tube.

Maintaining the video image on the screen uses cpu cycles, so turning off the display can improve the speed of compute-bound tasks. When the display is off, the screen will be white but any printing or displaying that the program does will be visible when the display is turned back on. Note: Breaks and PAGEFULLFN waiting (page 28.30) turn the display on, but users should be aware that it is possible to have the system waiting for a response to a question printed or a menu displayed on a non-visible part of the screen. The functions below are provided to turn the display off.

Note: These functions have no effect on the Xerox 1108 display.

(SETDISPLAYHEIGHT NSCANLINES) [Function]
Sets the display to only show the top NSCANLINES of the screen. If NSCANLINES is T, resets the display to show the full screen. Returns the previous setting.
(DISPLAY SCREEN)

**DISPLAYDOWN FORM NSCANLINES**

Evaluates FORM (with the display set to only show the top NSCANLINES of the screen), and returns the value of FORM. It restores the screen to its previous setting. If NSCANLINES is not given, it defaults to 0.

### 30.7 Miscellaneous Terminal I/O

(RINGBELLS N)

Flashes (reverse-videos) the screen N times (default 1). On the Xerox 1108, this also beeps through the keyboard speaker.

(PLAYTUNE Frequency/Duration.pairlist)

On the Xerox 1108, PLAYTUNE plays a sequence of notes through the keyboard speaker. Frequency/Duration.pairlist should be a list of dotted pairs (FREQUENCY . DURATION). PLAYTUNE maps down its argument, beeping the 1108 keyboard buzzer at each frequency for the specified amount of time. Specifying NIL for a frequency means to turn the beeper off the specified amount of time. The units of time are TICKS (page 12.16), which last about 28.78 microseconds on the Xerox 1108. PLAYTUNE makes no sound on a Xerox 1132. The default "simulate" entry for control-G (ASCII BEL) on the 1108 uses PLAYTUNE to make a short beep.

PLAYTUNE is implemented using BEEPON and BEEPOFF:

(BEEPON FREQ)

On the Xerox 1108, turns on the keyboard speaker playing a note with frequency FREQ, measured in TICKS (page 12.16). The speaker will continue to play the note until BEEPOFF is called.

(BEEPOFF)

Turns off the keyboard speaker on the Xerox 1108.

(SETMAINTPANEL N)

On the Xerox 1108, this sets the four-digit "maintanance panel" display on the front of the computer to display the number N.
31. Ethernet

31.1. Ethernet Protocols

31.1.1. Protocol Layering

31.1.2. Level Zero Protocols

31.1.3. Level One Protocols

31.1.4. Higher Level Protocols

31.1.5. Connecting Networks: Routers and Gateways

31.1.6. Addressing Conflicts with Level Zero Mediums

31.1.7. References

31.2. Higher-level PUP Protocol Functions

31.3. Higher-level NS Protocol Functions

31.3.1. Name and Address Conventions

31.3.2. Clearinghouse Functions

31.3.3. NS Printing

31.3.4. SPP Stream Interface

31.3.5. Courier Remote Procedure Call Protocol

31.3.5.1. Defining Courier Programs

31.3.5.2. Courier Type Definitions

31.3.5.2.1. Pre-defined Types

31.3.5.2.2. Constructed Types

31.3.5.2.3. User Extensions to the Type Language

31.3.5.3. Performing Courier Transactions

31.3.5.3.1. Expedited Procedure Call

31.3.5.3.2. Expanding Ring Broadcast

31.3.5.3.3. Using Bulk Data Transfer

31.3.5.3.4. Courier Subfunctions for Data Transfer

31.4. Level One Ether Packet Format

31.5. PUP Level One Functions

31.5.1. Creating and Managing Pups
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.5.2</td>
<td>Sockets</td>
<td>31.28</td>
</tr>
<tr>
<td>31.5.3</td>
<td>Sending and Receiving Pups</td>
<td>31.29</td>
</tr>
<tr>
<td>31.5.4</td>
<td>Pup Routing Information</td>
<td>31.30</td>
</tr>
<tr>
<td>31.5.5</td>
<td>Miscellaneous PUP Utilities</td>
<td>31.31</td>
</tr>
<tr>
<td>31.5.6</td>
<td>PUP Debugging Aids</td>
<td>31.32</td>
</tr>
<tr>
<td>31.6</td>
<td>NS Level One Functions</td>
<td>31.36</td>
</tr>
<tr>
<td>31.6.1</td>
<td>Creating and Managing XIPs</td>
<td>31.36</td>
</tr>
<tr>
<td>31.6.2</td>
<td>NS Sockets</td>
<td>31.37</td>
</tr>
<tr>
<td>31.6.3</td>
<td>Sending and Receiving XIPs</td>
<td>31.37</td>
</tr>
<tr>
<td>31.6.4</td>
<td>NS Debugging Aids</td>
<td>31.38</td>
</tr>
<tr>
<td>31.7</td>
<td>Support for Other Level One Protocols</td>
<td>31.38</td>
</tr>
<tr>
<td>31.8</td>
<td>The SYSQUEUE mechanism</td>
<td>31.41</td>
</tr>
</tbody>
</table>
Interlisp was first developed on large timesharing machines which provided each user with access to large amounts of disk storage, printers, mail systems, etc. Interlisp-D, however, was designed to run on smaller, single-user machines without these facilities. In order to provide Interlisp-D users with access to all of these services, Interlisp-D supports the Ethernet communications network, which allows multiple Interlisp-D machines to share common printers, file servers, etc.

Interlisp-D supports the Experimental Ethernet (3 Megabits per second) and the Ethernet (10 Megabits per second) local communications networks. These networks may be used for accessing file servers, remote printers, mail servers, or other machines. This chapter is divided into three sections: First, an overview of the various Ethernet and Experimental Ethernet protocols is presented. Then follow sections documenting the functions used for implementing PUP and NS protocols at various levels.

### 31.1 Ethernet Protocols

The members of the Xerox 1100 family (1108, 1132), Xerox file servers and laser xerographic printers, along with machines made by other manufacturers (most notably DEC) have the capability of communicating over 3 Megabit per second Experimental Ethernets, 10 Megabit per second Ethernets and telephone lines.

Xerox pioneered its work with Ethernet using a set of protocols known as PARC Universal Packet (PUP) computer communication protocols. The architecture has evolved into the newer Network Systems (NS) protocols developed for use in Xerox office products. All of the members of the Xerox 1100 family can use both NS and PUP protocols.

### 31.1.1 Protocol Layering

The communication protocols used by the members of the Xerox 1100 family are implemented in a "layered" fashion, which means that different levels of communication are implemented
as different protocol layers. Protocol Layering allows implementations of specific layers to be changed without requiring changes to any other layers. The layering also allows use of the same higher level software with different lower levels of protocols. Protocol designers can implement new types of protocols at the correct protocol level for their specific application in a layered system.

At the bottom level, level zero, there is a need to physically transmit data from one point to another. This level is highly dependent on the particular transmission medium involved. There are many different level zero protocols, and some of them may contain several internal levels. At level one, there is a need to decide where the data should go. This level is concerned with how to address a source and destination, and how to choose the correct transmission medium to use in order to route the packet towards its destination. A level one packet is transmitted by encapsulating it in the level zero packet appropriate for the transmission medium selected. For each independent communication protocol system, a single level one protocol is defined. The rule for delivery of a level one packet is that the communication system must only make a best effort to deliver the packet. There is no guarantee that the packet is delivered, that the packet is not duplicated and delivered twice, or that the packets will be delivered in the same order as they were sent.

The addresses used in level zero and level one packets are not necessarily the same. Level zero packets are specific to a particular transmission medium. For example, the destination address of a level zero packet transmitted on one of the two kinds of Ethernet is the Ethernet address (host number) of a machine on the particular network. Level one packets specify addresses meaningful to the particular class of protocols being implemented. For the PUP and NS protocols, the destination address comprises a network number, host number (not necessarily the same as the level zero host number), and a socket number. The socket number is a higher-level protocol concept, used to multiplex packets arriving at a single machine destined for separate logical processes on the machine.

Protocols in level two add order and reliability to the level one facilities. They suppress duplicate packets, and are responsible for retransmission of packets for which acknowledgement has not been received. The protocol layers above level two add conventions for data structuring, and implement application specific protocols.

31.1.2 Level Zero Protocols

Level zero protocols are used to physically connect computers. The addresses used in level zero protocols are protocol specific.
The Ethernet and Experimental Ethernet level zero protocols use host numbers, but level zero phone line protocols contain less addressing information since there are only two hosts connected to the telephone line, one at each end. As noted above, a level zero protocol does not include network numbers.

The 3MB Experimental Ethernet [1] was developed at PARC. Each Experimental Ethernet packet includes a source and destination host address of eight bits. The Experimental Ethernet standard is used by any machine attached to an Experimental Ethernet.

The 10MB Ethernet [2] was jointly developed and standardized by Digital, Intel, and Xerox. Each Ethernet level zero packet includes a source and destination host address that is 48 bits long. The Ethernet standard is used by any machine attached to an Ethernet.

Both of the level one protocols described later (PUP and NS) can be transported on any of the level zero protocols described above.

The Ethernet and Experimental Ethernet protocols are broadcast mediums. Data packets can be sent on these networks to every host attached to the net. A packet directed at every host on a network is a broadcast packet.

Other Level 0 protocols in use in industry include X.25, broadband networks, and Chaosnet. In addition, by using the notion of "mutual encapsulation", it is possible to treat a higher-level protocol (e.g. ARPANET) as if it were a Level Zero Protocol.

31.1.3 Level One Protocols

Two Level One Protocols are used in the Xerox 1100 Family, the PUP and the NS protocols. With the proper software, computers attached to Ethernets or Experimental Ethernets can send PUPs and NS packets to other computers on the same network, and to computers attached to other Ethernets or Experimental Ethernets.

The PUP protocols [3] were designed by Xerox computer scientists at the Palo Alto Research Center. The destination and source addresses in a PUP packet are specified using an 8-bit network number, an 8-bit host number, and a 32-bit socket number. The 8-bit network number allows an absolute maximum of 256 PUP networks in an internet. The 8-bit host number is network relative. That is, there may be many host number "1"s, but only one per network. 8 bits for the host number limits the number of hosts per network to 256. The
socket number is used for further levels of addressing within a specific machine.

The Network Systems (NS) protocols [4, 5] were developed by the Xerox Office Products Division. Each NS packet address includes a 32-bit network number, a 48-bit host number, and a 16-bit socket number. The NS host and network numbers are unique through all space and time. A specific NS host number is generally assigned to a machine when it is manufactured, and is never changed. In the same fashion, all networks (including those sold by Xerox and those used within Xerox) use the same network numbering space—there is only one network "74".

### 31.1.4 Higher Level Protocols

The higher level PUP protocols include the File Transfer Protocol (FTP) and the Leaf Protocol used to send and retrieve files from Interim File Servers (IFSs) and DEC File Servers, the Telnet protocol implemented by "Chat" windows and servers, and the EFTP protocol used to communicate with the laser xerographic printers developed by PARC ("Dovers" and "Penguins").

The higher level NS protocols include the Filing Protocol which allows workstations to access the product File Services sold by Xerox, the Clearinghouse Protocol used to access product Clearinghouse Services, and the TelePress Protocol used to communicate with the Xerox model 8044 Print Server.

### 31.1.5 Connecting Networks: Routers and Gateways

When a level one packet is sent from one machine to another, and the two machines are not on the same network, the packet must be passed between networks. Computers that are connected to two or more level zero mediums are used for this function. In the PUP world, these machines have been historically called "Gateways." In the NS world these machines are called Internetwork Routers (Routers), and the function is packaged and sold by Xerox as the Internetwork Routing Service (IRS).

Every host that uses the PUP protocols requires a PUP address; NS Hosts require NS addresses. An address consists of two parts: the host number and the network number. A computer learns its network number by communicating with a Router or Gateway that is attached to the same network. Host number determination is dependent on the hardware and the type of host number, PUP or NS.
Note that there is absolutely no relationship between a host's NS host and net numbers and the same host's PUP host and net numbers.

31.1.6 Addressing Conflicts with Level Zero Mediums

For convenience in the respective protocols, a level one PUP (8-bit) host number is the same as a level zero Experimental Ethernet host number; i.e., when a PUP level one packet is transported by an Experimental Ethernet to another host on the same network, the level zero packet specifies the same host number as the level one packet. Similarly, a level one NS (48-bit) host number is the same as a level zero Ethernet host number.

When a PUP level one packet is transported by an Ethernet, or an NS level one packet is sent on Experimental Ethernet, the level one host number cannot be used as the level zero address, but rather some means must be provided to determine the correct level zero address. Xerox solved this problem by specifying another level-one protocol called translation to allow hosts on an Experimental Ethernet to announce their NS host numbers, or hosts on an Ethernet to announce their PUP host numbers. Thus, both the Ethernet and Experimental Ethernet Level Zero Protocols totally support both families of higher level protocols.

31.1.7 References

31.2 Higher-level PUP Protocol Functions

This section describes some of the functions provided in Interlisp-D to perform protocols above Level One. Level One functions are described in a later section, for the benefit of those users who wish to program new protocols.

(ETHERHOSTNUMBER NAME)  [Function]
Returns the number of the named host. The number is 16-bit quantity, the high 8 bits designating the net and the low 8 bits the host. If NAME is NIL, returns the number of the local host.

(ETHERPORT NAME ERRORFLG MULTFLG)  [Function]
Returns a port corresponding to NAME. A "port" is a network address that represents (potentially) one end of a network connection, and includes a socket number in addition to the network and host numbers. Most network functions that take a port as argument allow the socket to be zero, in which case a well-known socket is supplied. A port is currently represented as a dotted pair (NETHOST. SOCKET).

NAME may be a litatom, in which case its address is looked up, or a port, which is just returned directly. If ERRORFLG is true, generates an error "host not found" if the address lookup fails, else it returns NIL. If MULTFLG is true, returns a list of alternative port specifications for NAME, rather than a single port (this is provided because it is possible for a single name in the name database to have multiple addresses). If MULTFLG is NIL and NAME has more than one address, the currently nearest one is returned. ETHERPORT caches its results.

The SOCKET of a port is usually zero, unless the name explicitly contains a socket designation, a number or symbolic name following a + in NAME, e.g., PHYLUM + LEAF. A port can also be specified in the form "NET#HOST#SOCKET", where each of NET, HOST and SOCKET is a sequence of octal digits; the socket, but not the terminating #, can be omitted, in which case the socket is zero.

(ETHERHOSTNAME PORT USE. OCTAL. DEFAULT)  [Function]
Looks up the name of the host at address PORT. PORT may be a numeric address, a (NETHOST. SOCKET) pair returned from ETHERPORT, or a numeric designation in string form, "NET#HOST#SOCKET", as described above. In the first case, the net defaults to the local net. If PORT is NIL, returns the name of the local host. If there is no name for the given port, but USE. OCTAL. DEFAULT is true, the function returns a string specifying the port in octal digits, in the form "NET#HOST#SOCKET", with SOCKET omitted if it is zero. Most
functions that take a port argument will also accept ports in this octal format.

(EFTP HOST FILE PRINTOPTIONS) [Function]
Transmits FILE to HOST using the EFTP protocol. The FILE need not be open on entry, but in any case is closed on exit. EFTP returns only on success; if HOST does not respond, it keeps trying.

The principal use of the EFTP protocol is for transmitting Press files to a printer. If PRINTOPTIONS is non-NIL, EFTP assumes that HOST is a printer and FILE is a Press file, and takes additional action: it calls PRINTERSTATUS (page 29.4) for HOST and prints this information to the prompt window; and it fills in the "printed-by" field on the last page of the press file with the value of USERNAME (page 24.40). Also, PRINTOPTIONS is interpreted as a list in property list format that controls details of the printing. Possible properties are as follows:

#COPIES Value is the number copies of the file to print. Default is one.
#SIDES If the value is 2, select two-sided printing (if the printer can print two-sided copies).
DOCUMENT.CREATION.DATE Value is the document creation date to appear on the header page (an integer date as returned by IDATE).
DOCUMENT.NAME Value is the document name to appear on the header page (as a string). Default is the full name of the file.

31.3 Higher-level NS Protocol Functions

The following is a description of the Interlisp-D facilities for using Xerox SPP and Courier protocols and the services based on them. The sections on naming conventions, Printing, and Filing are of general interest to users of Network Systems servers. The remaining sections describe interfaces of interest to those who wish to program other applications on top of either Courier or SPP.

31.3.1 Name and Address Conventions

Addresses of hosts in the NS world consist of three parts, a network number, a machine number, and a socket number. These three parts are embodied in the Interlisp-D data type NSADDRESS. Objects of type NSADDRESS print as "net#h1.h2.h3#socket", where all the numbers are printed in octal radix, and the 48-bit host number is broken into three
16-bit fields. Most functions that accept an address argument will accept either an NSADDRESS object or a string that is the printed representation of the address.

Higher-level functions accept host arguments in the form of a symbolic name for the host. The NS world has a hierarchical name space. Each object name is in three parts: the Organization, the Domain, and the Object parts. There can be many domains in a single organization, and many objects in a single domain. The name space is maintained by the Clearinghouse, a distributed network database service.

A Clearinghouse name is standardly notated as object:domain:organization. The parts organization or domain:organization may be omitted if they are the default (see below). Alphabetic case is not significant. Internally, names are represented as objects of data type NSNAME, but most functions accept the textual representation as well, either as a litatom or a string. Objects of type NSNAME print as object:domain:organization, with fields omitted when they are equal to the default. A Domain is standardly represented as an NSNAME in which the object part is null. If frequent use is to be made of an NS name, it is generally preferable to convert it to an NSNAME once, by calling PARSE.NSNAME, then passing the resultant object to all functions desiring it.

CH.DEFAULT.ORGANIZATION [Variable]

This is a string specifying the default Clearinghouse organization.

CH.DEFAULT.DOMAIN [Variable]

This is a string specifying the default Clearinghouse domain. If it or the variable CH.DEFAULT.ORGANIZATION is NIL, they are set by Lisp system code (when they are needed) to be the first domain served by the nearest Clearinghouse server.

In small organizations with just one domain, it is reasonable to just leave these variables NIL and have the system set them appropriately. In organizations with more than one domain, it is wise to set them in the site initialization file, so as not to be dependent on exactly which Clearinghouse servers are up at any time.

(PARSE.NSNAME NAME #PARTS DEFAULTDOMAIN) [Function]

When #PARTS is 3 (or NIL), parses NAME, a litatom or string, into its three parts, returning an object of type NSNAME. If the domain or organization is omitted, defaults are supplied, either from DEFAULTDOMAIN (an NSNAME whose domain and
organization fields only are used) or from the variables CH.DEFAULT.DOMAIN and CH.DEFAULT.ORGANIZATION.

If #PARTS is 2, NAME is interpreted as a domain name, and an NSNAME with null object is returned. In this case, if NAME is a full 3-part name, the object part is stripped off.

If #PARTS is 1, NAME is interpreted as an organization name, and a simple string is returned. In this case, if NAME is a 2- or 3-part name, the organization is extracted from it.

If NAME is already an object of type NSNAME, then it is returned as is (if #PARTS is 3), or its domain and/or organization parts are extracted (if #PARTS is 1 or 2).

\[(\text{NSNAME}.\text{TO}.\text{STRING} \text{ NSNAME} \text{ FULLNAMEFLG})\]  

[Function]

Converts NSNAME, an object of type NSNAME, to its string representation. If FULLNAMEFLG is true, the full printed name is returned; otherwise, fields that are equal to the default are omitted.

Programmers who wish to manipulate NSADDRESS and NSNAME objects directly should load the Library package ETHERRECORDS.

### 31.3.2 Clearinghouse Functions

This section describes functions that may be used to access information in the Clearinghouse.

\[(\text{START}.\text{CLEARINGHOUSE} \text{ RESTARTFLG})\]  

[Function]

Performs an expanding ring broadcast in order to find the nearest Clearinghouse server, whose address it returns. If a Clearinghouse has already been located, this function simply returns its address immediately, unless RESTARTFLG is true, in which case the cache of Clearinghouse information is invalidated and a new broadcast is performed. \text{START}.\text{CLEARINGHOUSE} is normally performed automatically by the system the first time it needs Clearinghouse information; however, it may be necessary to call it explicitly (with RESTARTFLG set) if the local Clearinghouse server goes down.

\[\text{CH.NET.HINT}\]  

[Variable]

A number or list of numbers, giving a hint as to which network the nearest Clearinghouse server is on. When \text{START}.\text{CLEARINGHOUSE} looks for a Clearinghouse server, it probes the network(s) given by \text{CH.NET.HINT} first, performing the expanding ring broadcast only if it fails there. If the nearest Clearinghouse server is not on the directly connected network,
setting CH.NET.HINT to the proper network number in the local site init file (page 12.1) can speed up START.CLEARINGHOUSE considerably.

### SHOW.CLEARINGHOUSE ENTIRE.CLEARINGHOUSE? DONT.GRAPH)

This function displays the structure of the cached Clearinghouse information in a window. Once created, it will be redisplayed whenever the cache is updated, until the window is closed. The structure is shown using the library package GRAPHER.

If ENTIRE.CLEARINGHOUSE? is true, then this function probes the Clearinghouse to discover the entire domain:organization structure of the Internet, and graphs the result. If DONT.GRAPH is true, the structure is not graphed, but rather the results are returned as a nested list indicating the structure.

### LOOKUP.NS.SERVER NAME TYPE FULLFLG)

Returns the address, as an NSADDRESS, for the object, NAME. TYPE is the property under which the address is stored, which defaults to ADDRESS.LIST. The information is cached so that it need not be recomputed on each call; the cache is cleared by restarting the Clearinghouse. If FULLFLG is true, returns a list whose first element is the canonical name of NAME and whose tail is the address list.

The following functions perform various sorts of retrieval operations on database entries in the Clearinghouse. Here, "The Clearinghouse" refers to the collective service offered by all the Clearinghouse servers on an internet; Lisp internally deals with which actual server(s) it needs to contact to obtain the desired information. The argument(s) describing the objects under consideration can be strings or NSNAME's, and in most cases can contain the wild card "*", which matches a subsequence of zero or more characters. Wildcards are permitted only in the most specific field of a name (e.g., in the object part of a full three-part name). When an operation intended for a single object is instead given a pattern, the operation is usually performed on the first matching object in the database, which may or may not be interesting.

### (CH.LOOKUP.OBJECT OBJECTPATTERN)

Looks up OBJECTPATTERN in the Clearinghouse database, returning its canonical name (as an NSNAME) if found, NIL otherwise. If OBJECTPATTERN contains a "*", returns the first matching name.
**ETHERNET**

**HIGHER-LEVEL NS PROTOCOL FUNCTIONS**

**CH.LIST.ORGANIZATIONS ORGANIZATIONPATTERN**

Returns a list of organization names in the Clearinghouse database matching ORGANIZATIONPATTERN. The default pattern is "*", which matches anything.

**CH.LIST.DOMAINS DOMAINPATTERN**

Returns a list of domain names (two-part NSNAME's) in the Clearinghouse database matching DOMAINPATTERN. The default pattern is "*", which matches anything in the default organization.

**CH.LIST.OBJECTS OBJECTPATTERN PROPERTY**

Returns a list of object names matching OBJECTPATTERN and having the property PROPERTY. PROPERTY is a number or a symbolic name for a Clearinghouse property; the latter include USER, PRINT.SERVICE, FILE.SERVICE, MEMBERS, ADDRESS.LIST and ALL.

For example,

(CH.LIST.OBJECTS "*:PARC:Xerox" (QUOTE USER))

returns a list of the names of users in the domain PARC:Xerox.

(CH.LIST.OBJECTS "*:lisp*:PARC:Xerox" (QUOTE MEMBERS))

returns a list of all group names in PARC:Xerox containing the substring "lisp".

**CH.LIST.ALIASES OBJECTNAMEPATTERN**

Returns a list of all objects in the Clearinghouse database that are aliases and match OBJECTNAMEPATTERN.

**CH.LIST.ALIASES.OF OBJECTPATTERN**

Returns a list of all objects in the Clearinghouse database that are aliases of OBJECTPATTERN.

**CH.RETRIEVE.ITEM OBJECTPATTERN PROPERTY INTERPRETATION**

Retrieves the value of the PROPERTY property of OBJECTPATTERN. Returns a list of two elements, the canonical name of the object and the value. If INTERPRETATION is given, it is a Clearinghouse type (see page 31.19) with which to interpret the bits that come back; otherwise, the value is simply of the form (SEQUENCE UNSPECIFIED), a list of 16-bit integers representing the value.

**CH.RETRIEVE.MEMBERS OBJECTPATTERN PROPERTY**

Retrieves the members of the group OBJECTPATTERN, as a list of NSNAMES. PROPERTY is the Clearinghouse Group property
under which the members are stored; the usual property used for this purpose is `MEMBERS`.

(CH.ISMEMBER GROUPNAME PROPERTY SECONDARYPROPERTY NAME) [Function]
Tests whether `NAME` is a member of `GROUPNAME`'s `PROPERTY` property. This is a potentially complex operation; see the description of procedure `IsMember` in the Clearinghouse Protocol documentation for details.

31.3.3 NS Printing
This section describes the facilities that are available for printing Interpress masters on NS Print servers.

(NSPRINT PRINTER FILE OPTIONS) [Function]
This function prints an Interpress master on `PRINTER`, which is a Clearinghouse name represented as a string or `NSNAME`. If `PRINTER` is `NIL`, `NSPRINT` uses the first print server registered in the default domain. `FILE` is the name of an Interpress file to be printed. `OPTIONS` is a list in property list format that controls details of the printing (see `SEND.FILE.TO.PRINTER`, page 29.1).

(NSPRINTER.STATUS PRINTER) [Function]
This function returns a list describing the printer's current status; whether it is available or busy, and what kind of paper is loaded.

(NSPRINTER.PROPERTIES PRINTER) [Function]
This function returns a list describing the printer's capabilities at the moment; the type of paper loaded, whether it can print two-sided, etc.

31.3.4 SPP Stream Interface
This section describes the stream interface to the Sequenced Packet Protocol. SPP is the transport protocol for Courier, which in turn is the transport layer for Filing and Printing.

(SPP.OPEN HOST SOCKET PROBEP NAME PROPS) [Function]
This function is used to open a bidirectional SPP stream. There are two cases: user and server.
User: If `HOST` is specified, an SPP connection is initiated to `HOST`, an `NSADDRESS` or string representing an NS address. If the socket part of the address is null (zero), it is defaulted to `SOCKET`. If both `HOST` and `PROBEP` are specified, then the connection is
probed for a response before returning the stream; \texttt{NIL} is returned if \texttt{HOST} doesn't respond.

Server: If \texttt{HOST} is \texttt{NIL}, a passive connection is created which listens for an incoming connection to local socket \texttt{SOCKET}.

\textbf{SPP\textunderscore OPEN} returns the input side of the bidirectional stream; the function \texttt{SPOUTPUTSTREAM} is used to obtain the output side. The standard stream operations \texttt{BIN}, \texttt{READP}, \texttt{EOFP} (on the input side), and \texttt{BOUT}, \texttt{FORCEOUTPUT} (on the output side), are defined on these streams, as is \texttt{CLOSEF}, which can be applied to either stream to close the connection.

\texttt{NAME} is a mnemonic name for the connection process, mainly useful for debugging.

\texttt{PROPS} is an optional property list, used to set the properties that determine the behavior of the SPP stream when certain events occur. The following properties can be specified:

- **CLOSEFN**
  A function or list of functions called (with the stream as argument) when an SPP connection is closed.

- **ATTENTIONFN**
  A function called (with the stream as argument) when an ATTENTION packet is received on the SPP connection.

- **ERRORHANDLER**
  A function called (with the stream as argument) when an error (such as end-of-stream) occurs on the SPP connection.

- **OTHERXIPHANDLER**
  A function called (with the stream as argument) when a non-SPP, non-error packet is received on the socket associated with the SPP connection.

- **EOM\textunderscore ON\textunderscore FORCEOUTPUT**
  The value of this property should be either \texttt{T} or \texttt{NIL} (the default). If \texttt{T}, then the end-of-message bit is set when the current collection of bytes buffered for transmission is forcibly sent (e.g. by \texttt{FORCEOUTPUT}, page 25.10).

- **SERVER\textunderscore FUNCTION**
  This property can be used for creating SPP servers. Normally, when a connection is opened with the \texttt{HOST} argument set to \texttt{NIL}, a passive "listener" connection is created. \texttt{SPP\textunderscore OPEN} will not return until some other host attempts to connect to socket specified in the \texttt{SPP\textunderscore OPEN} call.

If the \texttt{SERVER\textunderscore FUNCTION} property is specified, a new listener (and listener process) is created. \texttt{SPP\textunderscore OPEN} will return immediately. Whenever another host attempts to connect to the specified socket, a new process and unique SPP connection are created. The function specified by the \texttt{SERVER\textunderscore FUNCTION} property is run in the top level of the new process. The server function should be a function of two arguments: the first argument is the SPP input stream associated with the connection; the second argument is the SPP output stream associated with the connection.
(SPPOUTPUTSTREAM STREAM)  [Function]
Applied to the input stream of an SPP connection, this function returns the corresponding output stream.

SPP.USER.TIMEOUT  [Variable]
Specifies the time, in milliseconds, to wait before deciding that a host isn't responding.

(SPP.DSTYPE STREAM DSTYPE)  [Function]
Accesses the current datastream type of the connection. If DSTYPE is NIL, returns the datastream type of the current packet being read. If DSTYPE is non-NIL, sets the datastream type of all subsequent packets sent on this connection, until the next call to SPP.DSTYPE. Since this affects the current partially-filled packet, the stream should probably be flushed (via FORCEOUTPUT) before this function is called.

(SPP.SENDEOM STREAM)  [Function]
Transmits the data buffered so far on the output stream STREAM, if any, with the End of Message bit set. If there is nothing buffered, sends a zero-length packet with the End of Message bit set.

(SPP.SENDATTENTION STREAM ATTENTIONBYTE —)  [Function]
Sends an SPP "attention" packet on the output stream STREAM, with the Attention bit set and containing the single byte of data ATTENTIONBYTE.

Note: The appropriate way to determine whether an SPP stream is open, or whether an End of Message or Attention indication has been reached (for input streams) is to use the EOFP function (page 25.6). When EOFP is applied to an SPP stream, it returns one of the following values:

NIL  The connection is open and readable or writable.
T  The connection is closed.
EOM  (Input streams only) The End of Message bit was set in the last packet received, and all bytes from the packet have been read. The function SPP.CLEAREOM (below) must be called to clear this condition.
ATTENTION  (Input streams only) An attention packet is waiting. SPP.CLEARATTENTION (below) must be called before the single byte of data associated with the attention packet can be read.
(SPP.CLEAREOM STREAM NOERRORFLG)  
[Function]
Clears the End of Message indication on STREAM. This is necessary in order to read beyond the EOM. Causes an error if the stream is not currently at the End of Message, unless NOERRORFLG is non-NIL.

(SPP.CLEARATTENTION STREAM NOERRORFLG)  
[Function]
Clears the Attention packet indication on STREAM. This must be called before the single byte of data associated with the attention packet can be read. Causes an error if the stream does not have an attention packet waiting, unless NOERRORFLG is non-NIL.

31.3.5 Courier Remote Procedure Call Protocol

Courier is the Xerox Network Systems Remote Procedure Call protocol. It uses the Sequenced Packet Protocol for reliable transport. Courier uses procedure call as a metaphor for the exchange of a request from a user process and its positive reply from a server process; exceptions or error conditions are the metaphor for a negative reply. A family of remote procedures and the errors they can raise constitute a remote program. A remote program generally represents a complete service, such as the Filing or Printing programs described earlier in this chapter.

For more detail about Courier, the reader is referred to the published specification of the Courier protocol. The following documentation assumes some familiarity with the protocol. It describes how to define a Courier program and use it to communicate with a remote system element that implements a server for that program. This section does not discuss how to construct such a server.

31.3.5.1 Defining Courier Programs

A Courier program definition is accessed using the file package type COURIERPROGRAMS, so GETDEF, PUTDEF, and EDITDEF can be used to manipulate Courier programs. The file package command COURIERPROGRAMS (page 17.39) can be used to save Courier programs on files. Courier program are initially defined using the following function:

(COURIERPROGRAM NAME ...)  
[NLambda NoSpread Function]
This function is used to define Courier programs. The syntax is

(COURIERPROGRAM NAME

(PROGRAMNUMBER VERSIONNUMBER)

.DEFINITIONS)
The tail _DEFINITIONS_ is a property list where the properties are selected from _TYPES_, _PROCEDURES_, _ERRORS_ and _INHERITS_; the values are lists of pairs of the form (_LABEL_ . _DEFINITION_). These are described in more detail as follows:

The _TYPES_ section lists the symbolically-defined types used to represent the arguments and results of procedures and errors in this Courier program. Each element in this section is of the form (_TYPENAME_ _TYPEDEFINITION_), e.g., (_PRIORITY_ INTEGER). The _TYPEDEFINITION_ can be a predefined type (see next section), another type defined in this _TYPES_ section, or a qualified typename taken from another Courier program; these latter are written as a dotted pair (_PROGRAMNAME_ . _TYPENAME_).

The _PROCEDURES_ section lists the remote procedures defined by this Courier program. A procedure definition is a stylized reduction of the Courier definition syntax defined in the Courier Protocol specification:

(_PROCEDURENAME_ _NUMBER_ _ARGUMENTS_ _RETURNS_ _RESULTTYPES_ _REPORTS_ _ERRORNAMES_)

_ArgUMENTS_ is a list of type names, one per argument to the remote procedure, or NIL if the procedure takes no arguments. _RESULTTYPES_ is a list of type names, one for each value to be returned. _ERRORNAMES_ is a list of names of errors that can be raised by this procedure; each such error must be listed in the program's _ERRORS_ section. The atoms _RETURNS_ and _REPORTS_ are noise words to aid readability.

The _ERRORS_ section lists the errors that can be raised by procedures in this program. An error definition is of the form

(_ERRORNAME_ _NUMBER_ _ARGUMENTS_),

where _ARGUMENTS_ is a list of type names, one for each argument, if any, reported by the error.

The _INHERITS_ section is an optional list of other Courier programs, some of whose definitions are "inherited" by this program. More specifically, if a type, procedure or error referenced in the current program definition is not defined in this program, the system searches for a definition of it in each of the inherited programs in turn, and uses the first such definition found.

The _INHERITS_ section is useful when defining variants of a given Courier program. For example, if one wanted to try out version 4 of Courier program _BAR_, and version 4 differed from version 3 of program _BAR_ only in a small number of procedure or type definitions, one could define a program _NEWBAR_ with an _INHERITS_ section of (_BAR_) and only need to list the few changed definitions inside _NEWBAR_.

---

ETHERNET
31.3.5.2 Courier Type Definitions

This section describes how the Courier types described in the Courier Protocol document are expressed in a Lisp Courier program definition, and how values of each type are represented. Each type in a Courier program's TYPES section must ultimately be defined in terms of one of the following "base" types, although the definition can be indirect through arbitrarily many levels. That is, a type can be defined in terms of any other type known by an extant Courier definition. The names of the base types are "global"; they need no qualification, nor do type names mentioned in the same Courier program. To refer to a type not defined in the same Courier program (or to any non-base type when there is no program context), one writes a Qualified name, in the form (PROGRAM . TYPE). In general, a Qualified name is legal in any place that calls for a Courier type.

31.3.5.2.1 Pre-defined Types

Pre-defined (atomic) types are expressed as uppercase litatoms from the following set:

**BOOLEAN**
Values are represented by T and NIL.

**INTEGER**
Values are represented as small integers in the range [-32768..32767].

**CARDINAL**
Values are represented as small integers in the range [0..65535].

**UNSPECIFIED**
Same as CARDINAL.

**LONGINTEGER**
Same as LONGINTEGER. Note that Interlisp-D does not (currently) have a datatype that truly represents a 32-bit unsigned integer.

**LONGCARDINAL**
Values are represented as FIXP's.

**STRING**
Values are represented as Lisp strings.

In addition, the following types not in the document have been added for convenience:

**TIME**
Represents a date and time in accordance with the Network Time Standard. The value is a FIXP such as returned by the function IDATE, and is encoded as a LONGCARDINAL.

**NSADDRESS**
Represents a network address. The value is an object of type NSADDRESS (page 31.7), and is encoded as six items of type UNSPECIFIED.

**NSNAME**
Represents a three-part Clearinghouse name. The value is an object of type NSNAME (page 31.8), and is encoded as three items of type STRING.
NSNAME2 Represents a two-part Clearinghouse name, i.e., a domain. The value is an object of type NSNAME (page 31.8), and is encoded as two items of type STRING.

31.3.5.2.2 Constructed Types

Constructed Types are composite objects made up of elements of other types. They are all expressed as a list whose CAR names the type and whose remaining elements give details. The following are available:

(ENUMERATION (NAME INDEX) ... (NAME INDEX))

Each NAME is an arbitrary litatom or string; the corresponding INDEX is its Courier encoding (a CARDINAL). Values of type ENUMERATION are represented as a NAME from the list of choices. For example, a value of type (ENUMERATION (UNKNOWN 0) (RED 1) (BLUE 2)) might be the litatom RED.

(SEQUENCE TYPE)

A SEQUENCE value is represented as a list, each element being of type TYPE. A SEQUENCE of length zero is represented as NIL. Note that there is no maximum length for a SEQUENCE in the Lisp implementation of Courier.

(ARRAY LENGTH TYPE)

An ARRAY value is represented as a list of LENGTH elements, each of type TYPE.

(CHOICE (NAME INDEX TYPE) ... (NAME INDEX TYPE))

The CHOICE type allows one to select among several different types at runtime; the INDEX is used in the encoding to distinguish the value types. A value of type CHOICE is represented in Lisp as a list of two elements, (NAME VALUE). For example, a value of type (CHOICE (STATUS 0 (ENUMERATION (BUSY 0) (COMPLETE 1))) (MESSAGE 1 STRING))

could be (STATUS COMPLETE) or (MESSAGE "Out of paper.").

(RECORD (FIELDNAME TYPE) ... (FIELDNAME TYPE))

Values of type RECORD are represented as lists, with one element for each field of the record. The field names are not part of the value, but are included for documentation purposes.

For programmer convenience, there are two macros that allow Courier records to be constructed and dissected in a manner similar to Lisp records. These compile into the appropriate composites of CONS, CAR and CDR.

(COURIER.CREATE TYPE FIELDNAME ← VALUE ... FIELDNAME ← VALUE) [Macro]

Creates a value of type TYPE, which should be a fully-qualified type name that designates a RECORD type, e.g., (MAILTRANSPORT . POSTMARK). Each FIELDNAME should correspond to a field of the record, and all fields must be
included. Each VALUE is evaluated; all other arguments are not. The assignment arrows are for readability, and are optional.

(COURIER.FETCH TYPE FIELD OBJECT) [Macro]
Analagous to the Record Package operator fetch. Argument TYPE is as with COURIER.CREATE; FIELD is the name of one of its fields. COURIER.FETCH extracts the indicated field from OBJECT. For readability, the noiseword "of" may be inserted between FIELD and OBJECT. Only the argument OBJECT is evaluated.

For example, if the program CLEARINGHOUSE has a type declaration

(USERTA.VALUE (RECORD (LAST.NAME.INDEX CARDINAL)
  (FILE.SERVICE STRING)),

then the expression

(SETQ INFO (COURIER.CREATE
  (CLEARINGHOUSE . USERTA.VALUE)
  LAST.NAME.INDEX ← 12
  FILE.SERVICE ← "Phylex:PARC:Xerox")

would set the variable INFO to the list (12 "Phylex:PARC:Xerox"). The expression

(COURIER.FETCH (CLEARINGHOUSE . USERTA.VALUE)
  FILE.SERVICE of INFO)

would produce "Phylex:PARC:Xerox".

31.3.5.2.3 User Extensions to the Type Language

The programmer can add new base types to the Courier language by telling the system how to read and write values of that type. The programmer chooses a name for the type, and gives the name a COURIERDEF property. The new name can then be used anywhere that the type names listed in the previous sections, such as CARDINAL, can be used. Such extensions are useful for user-defined objects, such as datatypes, that are not naturally represented by any predefined or constructed type. The NSADDRESS and NSNAME Courier types are defined by this mechanism.

COURIERDEF [Property Name]
The format of the COURIERDEF property is a list of up to four elements, (READFN WRITEFN LENGTHFN WRITEREPFN). The first two elements are required; if the latter two are omitted, the system will simulate them as needed. The elements are as follows:
31.3.5.3 Performing Courier Transactions

The normal use of Courier is to open a connection with a remote system element using COURIER.OPEN, perform one or more remote procedure calls using COURIER.CALL, then close the connection with CLOSEF.

(COURIER.OPEN HOSTNAME SERVERTYPE NOERRORFLG NAME WHENCLOSEDFN OTHERPROPS) [Function]

Opens a Courier connection to the Courier socket on HOST, and returns an SPP stream that can be passed to COURIER.CALL. HOSTNAME can be an NS address, or a symbolic Clearinghouse name in the form of a string, litatom or NSNAME. In the case of a symbolic name, SERVERTYPE specifies the Clearinghouse.
property under which the server's address may be found; normally, this is NIL, in which case the ADDRESS.LIST property is used.

Normally, if a connection cannot be made, or the server supports the wrong version of Courier, an error occurs. If NOERRORFLG is non-NIL, COURIER.OPEN returns NIL in these cases.

If NAME is non-NIL, it is used as the name of the Courier connection process.

WHENCLOSEDFN is a function (or list of functions) of one argument, the Courier stream, that will be called when the connection is closed, either by user or server.

If OTHERPROPS is non-NIL, it should be a property list of SPP stream properties, as accepted by SPP.OPEN (page 31.12). Any CLOSEFN property on this list is overridden by the value of WHENCLOSEDFN.

(COURIER.CALL STREAM PROGRAM PROCEDURE ARG1 ... ARGN NOERRORFLG)  [NoSpread Function]

This function calls the remote procedure PROCEDURE of the Courier program PROGRAM. STREAM is the stream returned by COURIER.OPEN. The arguments should be Lisp values appropriate for the Courier types of the corresponding formal parameters of the procedure. There must be the same number of actual and formal arguments. If the procedure call is successful, Courier returns the result(s) of the call as specified in the RETURNS section of the procedure definition. If there is only a single result, it is returned directly, otherwise a list of results is returned.

Procedures that take a Bulk Data argument (source or sink) are treated specially; see page 31.24.

If the procedure call results in an error, one of three possible courses is available. The default behavior is to cause a Lisp error. To suppress the error, an optional keyword can be appended to the argument list, as if an extra argument. This NOERRORFLG argument can be the atom NOERROR, in which case NIL is returned as the result of the call. If NOERRORFLG is RETURNERRORS, the result of the call is a list (ERROR ERRORNAME . ERRORARGS). If the failure was a Courier Reject, rather than Error, then ERRORNAME is the atom REJECT.

Examples:

(COURIER.PROGRAM PERSONNEL (17 1)
  TYPES
  ((PERSON.NAME (RECORD (FIRST.NAME STRING)
      (MIDDLE MIDDLE.PART)
      (LAST.NAME STRING)))

31.21
(MIDDLE.PART (CHOICE (NAME 0 STRING) 
  (INITIAL 1 STRING))) 
(BIRTHDAY (RECORD (YEAR CARDINAL) 
  (MONTH STRING) 
  (DAY CARDINAL))) 

PROCEDURES 
  ((GETBIRTHDAY 3 (PERSON.NAME) 
    RETURNS (BIRTHDAY) REPORTS (NO.SUCH.PERSON))) 

  ERRORS 
  ((NO.SUCH.PERSON 1)) 
)

This expression defines PERSONNEL to be Courier program number 17, version number 1. The example defines three types, PERSON.NAME, MIDDLE.PART and BIRTHDAY, and one procedure, GETBIRTHDAY, whose procedure number is 3. The following code could be used to call the remote GETBIRTHDAY procedure on the host with address HOSTADDRESS.

(SETQ STREAM (COURIER.OPEN HOSTADDRESS) 
(PROG1 (COURIER.CALL STREAM 'PERSONNEL 'GETBIRTHDAY 
  (COURIER.CREATE (PERSONNEL . PERSON.NAME) 
    FIRST.NAME ← "Eric" 
    MIDDLE ← '(INITIAL "e") 
    LAST.NAME ← "Cooper") 
  (CLOSEF STREAM)) 

COURIER.CALL in this example might return a value such as (1959 "January" 10).

31.3.5.3.1 Expedited Procedure Call

Some Courier servers support "Expedited Procedure Call", which is a way of performing a single Courier transaction by a Packet Exchange protocol, rather than going to the expense of setting up a full Courier connection. Expedited calls must have no bulk data arguments, and their arguments and results must each fit into a single packet.

(COURIER.EXPEDITED.CALL ADDRESS SOCKET# PROGRAM PROCEDURE ARG1 ... ARGN 
  NOERRORFLG) [NoSpread Function]

Attempts to perform a Courier call using the Expedited Procedure Call. ADDRESS is the NS address of the remote host and SOCKET# is the socket on which it is known to listen for expedited calls. The remaining arguments are exactly as with COURIER.CALL. If the arguments to the procedure do not fit in one packet, or if there is no response to the call, or if the call returns the error USE.COURIER (which must be defined by
31.3.5.3.2 Expanding Ring Broadcast

"Expanding Ring Broadcast" is a method of locating a server of a particular type whose address is not known in advance. The system broadcasts some sort of request packet on the directly-connected network, then on networks one hop away, then on networks two hops away, etc., until a positive response is received.

For use in locating a server for a particular Courier program, a stylized form of Expanding Ring Broadcast is defined. The request packet is essentially the call portion of an Expedited Procedure Call for some procedure defined in the program. The response packet is a Courier response, and typically contains at least the server's address as the result of the call. The designer of the protocol must, of course, specify which procedure to use in the broadcast (usually it is procedure number zero) and on what socket the server should listen for broadcasts.

START.CLEARINGHOUSE uses this procedure to locate the nearest Clearinghouse server.

(COURIER.BROADCAST.CALL DESTSOCKET# PROGRAM PROCEDURE ARGS RESULTFN NETHINT MESSAGE) [Function]

Performs an expanding ring broadcast for servers willing to implement PROCEDURE in Courier program PROGRAM. DESTSOCKET# is the socket on which such servers of this type are known to listen for broadcasts, typically the same socket on which they listen for expedited calls. ARGs is the argument list, if any, to the procedure (note that it is not spread, unlike with COURIER.CALL).

If a host responds positively, then the function RESULTFN is called with one argument, the Courier results of the procedure call. If RESULTFN returns a non-null value, the value is returned as the value of COURIER.BROADCAST.CALL and the search stops there; otherwise, the search for a responsive host continues. If RESULTFN is not supplied (or is NIL), then the results of the procedure call are returned directly from COURIER.BROADCAST.CALL; i.e., RESULTFN defaults to the identity function.

NETHINT, if supplied, is a net number or list of net numbers as a hint concerning which net(s) to try first before performing a pure expanding-ring broadcast. If MESSAGE is non-NIL, it is a description (string) of what the broadcast is looking for, to be
When a Courier program needs to transfer an arbitrary amount of information as an argument or result of a Courier procedure, the procedure is usually defined to have one argument of type "Bulk Data". The argument is a "source" if it is information transferred from caller to server (as though a procedure argument), a "sink" if it is information transferred from server to caller (as though a procedure result). These two "types" are indicated in a Courier procedure's formal argument list as BULK.DATA.SOURCE and BULK.DATA.SINK, respectively. A Courier procedure may have at most one such argument.

In a Courier call, the bulk data is transmitted in a special way, between the arguments and the results. There are two basic ways to handle this in the call. The caller can specify how the bulk data is to be interpreted (how to read or write it), or the caller can request to be given a bulk data stream as the result of the Courier call. The former is the preferred way; both are described below.

In the first method, the caller passes as the actual argument to the Courier call (i.e., in the position in the argument list occupied by BULK.DATA.SOURCE or BULK.DATA.SINK) a function to perform the transfer. Courier sets up the transaction, then calls the supplied function with one argument, a stream on which to write (if a source argument) or read (if a sink) the bulk data. If the function returns normally, the Courier transaction proceeds as usual; if it errors out, Courier sends a Bulk Data Abort to abort the transaction.

In the case of a sink argument, if the value returned from the sink function is non-NIL, it is returned as the result of COURIER.CALL; otherwise, the result of COURIER.CALL is the usual procedure result, as declared in the Courier program.

For convenience, a Bulk Data sink argument to a Courier call can be specified as a fully qualified Courier type, e.g., (CLEARINGHOUSE . NAME), in which case the Bulk Data stream is read as a "stream of" that type (see COURIER.READ.BULKDATA, below).

The second method for handling bulk data is to pass NIL as the bulk data "argument" to COURIER.CALL. In this case, Courier sets up the call, then returns a stream that is open for OUTPUT (if a source argument) or INPUT (if a sink). The caller is responsible for transferring the bulk data on the stream, then closing the stream to complete the transaction. The value returned from...
CLOSEF is the Courier result. This method is required if the caller's control structure is open-ended in a way such that the bulk data cannot be transferred within the scope of the call to COURIER.CALL.

In either method, the stream on which the bulk data is transferred is a standard Interlisp stream, so BIN, BOUT, COPYBYTES are all appropriate.

Many Courier programs define a "Stream of <type>" as a means of transferring an arbitrary number of objects, all of the same type. Although this is typically specified formally in the printed Courier documentation as a recursive definition, the recursion is in practice unnecessary and unwieldy; instead, the following function should be used.

(COURIER.READ.BULKDATA STREAM PROGRAM TYPE DONTCLOSE) [Function]
Reads from STREAM a "Stream of TYPE" for Courier program PROGRAM, and returns a list of the objects read. STREAM is closed on exit, unless DONTCLOSE is non-NIL.

Passing (X . Y) as the bulk argument to a Courier call is thus equivalent to passing the function (LAMBDA (STREAM) (COURIER.READ.BULKDATA STREAM X Y)).

31.3.5.3.4 Courier Subfunctions for Data Transfer

The following functions are of interest to those who transfer data in Courier representations, e.g., as part of a function to implement a user-defined Courier type.

(COURIER.READ STREAM PROGRAM TYPE) [Function]
Reads from the stream STREAM a Courier value of type TYPE for program PROGRAM. If TYPE is a predefined type, then PROGRAM is irrelevant; otherwise, it is required in order to qualify TYPE.

(COURIER.WRITE STREAM ITEM PROGRAM TYPE) [Function]
Writes ITEM to the stream STREAM as a Courier value of type TYPE for program PROGRAM.

(COURIER.READ.SEQUENCE STREAM PROGRAM TYPE) [Function]
Reads from the stream STREAM a Courier value SEQUENCE of values of type TYPE for program PROGRAM. Equivalent to (COURIER.READ STREAM PROGRAM (SEQUENCE TYPE)).
Higher-Level NS Protocol Functions

(COURIER.WRITE.SEQUENCE STREAM ITEM PROGRAM TYPE) [Function]

Equivalent to (COURIER.WRITE STREAM ITEM PROGRAM (SEQUENCE TYPE)).

Some Courier programs traffic in values whose interpretation is left up to the clients of the program; the values are transferred in Courier transactions as values of type (SEQUENCE UNSPECIFIED). For example, the Clearinghouse program transfers the value of a database property as an uninterpreted sequence, leaving it up to the caller, who knows what type of value the particular property takes, to interpret the sequence of raw bits as some other Courier representation. The following functions are useful when dealing with such values.

(COURIER.WRITE.REP VALUE PROGRAM TYPE) [Function]

Produces a list of 16-bit integers, i.e., a value of type (SEQUENCE UNSPECIFIED), that represents VALUE when interpreted as a Courier value of type TYPE in PROGRAM. Examples:

(COURIER.WRITE.REP T NIL 'BOOLEAN) = > (1)

(COURIER.WRITE.REP "Thing" NIL 'STRING) = >
  (5 52150Q 64556Q 63400Q)

(COURIER.WRITE.REP '(10 25) NIL '(SEQUENCE INTEGER)) = >
  (2 10 25)

(COURIER.READ.REP LIST.OF.WORDS PROGRAM TYPE) [Function]

Interprets LIST.OF.WORDS, a list of 16-bit integers, as a Courier object of type TYPE in the Courier program PROGRAM.

(COURIER.WRITE.SEQUENCE.UNSPECIFIED STREAM ITEM PROGRAM TYPE) [Function]

Writes to the stream STREAM in the form (SEQUENCE UNSPECIFIED) the object ITEM, whose value is really a Courier value of type TYPE for program PROGRAM. Equivalent to, but usually much more efficient than, (COURIER.WRITE STREAM (COURIER.WRITE.REP ITEM PROGRAM TYPE) NIL '(SEQUENCE UNSPECIFIED)).

31.4 Level One Ether Packet Format

The data type ETHERPACKET is the vehicle for all kinds of packets transmitted on an Ethernet or Experimental Ethernet. An ETHERPACKET contains several fields for use by the Ethernet drivers and a large, contiguous data area making up the data of the level zero packet. The first several words of the area are
reserved for the level one to zero encapsulation, and the remainder (starting at field EPBODY) make up the level one packet. Typically, each level one protocol defines a BLOCKRECORD (page 8.11) that overlays the ETHERPACKET starting at the EPBODY field, describing the format of a packet for that particular protocol. For example, the records PUP and XIP define the format of level one packets in the PUP and NS protocols.

The extra fields in the beginning of an ETHERPACKET have mostly a fixed interpretation over all protocols. Among the interesting ones are:

**EPLINK**
A pointer used to link packets, used by the SYSQUEUE mechanism (page 31.41). Since this field is used by the system for maintaining the free packet queue and ether transmission queues, do not use this field unless you understand it.

**EPFLAGS**
A byte field that can be used for any purpose by the user.

**EPUSERFIELD**
A pointer field that can be used for any purpose by the user. It is set to NIL when a packet is released.

**EPTRANSMITTING**
A flag that is true while the packet is "being transmitted", i.e., from the time that the user instructs the system to transmit the packet until the packet is gathered up from the transmitter's finished queue. While this flag is true, the user must not modify the packet.

**EPREQUEUE**
A pointer field that specifies the desired disposition of the packet after transmission. The possible values are: NIL means no special treatment; FREE means the packet is to be released after transmission; an instance of a SYSQUEUE means the packet is to be enqueued on the specified queue (page 31.41).

The normal life of an outgoing Ether packet is that a program obtains a blank packet, fills it in according to protocol, then sends the packet over the Ethernet. If the packet needs to be retained for possible retransmission, the EPREQUEUE field is used to specify a queue to place the packet on after its transmission, or the caller hangs on to the packet explicitly.

There are redefinitions, or "overlays" of the ETHERPACKET record specifically for use with the PUP and NS protocols. The following sections describe those records and the handling of the PUP and NS level one protocols, how to add new level one protocols, and the queueing mechanism associated with the EPREQUEUE field.
31.5 PUP Level One Functions

The functions in this section are used to implement level two and higher PUP protocols. That is, they deal with sending and receiving PUP packets. It is assumed the reader is familiar with the format and use of pups, e.g., from reading reference [3] on page 31.5.

31.5.1 Creating and Managing Pups

There is a record PUP that overlays the data portion of an ETHER_PACKET and describes the format of a pup. This record defines the following numeric fields: PUPLENGTH (16 bits), TCONTROL (transmit control, 8 bits, cleared when a PUP is transmitted), PUPTYPE (8 bits), PUPID (32 bits), PUPIDHI and PUPIDLO (16 bits each overlaying PUPID), PUPDEST (16 bits overlayed by 8-bit fields PUPDESTNET and PUPDESTHOST), PUPDESTSOCKET (32 bits, overlayed by 16-bit fields PUPDESTSOCKETHI and PUPDESTSOCKETLO), and PUPSOURCE, PUPSOURCENET, PUPSOURCEHOST, PUPSOUCESOCKET, PUPSOUCESOCKETHI, and PUPSOUCESOCKETLO, analogously. The field PUPCONTENTS is a pointer to the start of the data portion of the pup.

(ALLOCATE.PUP) [Function]
Returns a (possibly used) pup. Keeps a free pool, creating new pups only when necessary. The pup header fields of the pup returned are guaranteed to be zero, but there may be garbage in the data portion if the pup had been recycled, so the caller should clear the data if desired.

(CLEARPUP PUP) [Function]
Clears all information from PUP, including the pointer fields of the ETHER_PACKET and the pup data portion.

(RELEASE.PUP PUP) [Function]
Releases PUP to the free pool.

31.5.2 Sockets

Pups are sent and received on a socket. Generally, for each "conversation" between one machine and another, there is a distinct socket. When a pup arrives at a machine, the low-level pup software examines the pup's destination socket number. If there is a socket on the machine with that number, the incoming pup is handed over to the socket; otherwise the incoming pup is
discarded. When a user process initiates a conversation, it generally selects a large, random socket number different from any other in use on the machine. A server process, on the other hand, provides a specific service at a "well-known" socket, usually a fairly small number. In the PUP world, advertised sockets are in the range 0 to 100Q.

(OPENPUPSOCKET SKT# IFCLASH)  [Function]
Opens a new pup socket. If SKT# is NIL (the normal case), a socket number is chosen automatically, guaranteed to be unique, and probably different from any socket opened this way in the last 18 hours (the low half of the time of day clock is sampled).

If a specific local socket is desired, as is typically the case when implementing a server, SKT# is given, and must be a (up to 32-bit) number. IFCLASH indicates what to do in the case that the designated socket is already in use: if NIL, an error is generated; if ACCEPT, the socket is quietly returned; if FAIL, then OPENPUPSOCKET returns NIL without causing an error. Note that "well-known" socket numbers should be avoided unless the caller is actually implementing one of the services advertised as provided at the socket.

(CLOSEPUPSOCKET PUPSOC NOERRORFLG)  [Function]
Closes and releases socket PUPSOC. If PUPSOC is T, closes all pup sockets (this must be used with caution, since it will also close system sockets!). If PUPSOC is already closed, an error is generated unless NOERRORFLG is true.

(PUPSOCKETNUMBER PUPSOC)  [Function]
Returns the socket number (a 32-bit integer) of PUPSOC.

(PUPSOCKETEVENT PUPSOC)  [Function]
Returns the EVENT of PUPSOC (page 23.7). This event is notified whenever a pup arrives on PUPSOC, so pup clients can perform an AWAIT.EVENT on this event if they have nothing else to do at the moment.

31.5.3 Sending and Receiving Pups

(SENDPUP PUPSOC PUP)  [Function]
Sends PUP on socket PUPSOC. If any of the PUPSOURCEHOST, PUPSOURCENET, or PUPSOUCESOCKET fields is zero, SENDPUP fills them in using the pup address of this machine and/or the socket number of PUPSOC, as needed.
(GETPUP PUPSOC WAIT) [Function]
Returns the next pup that has arrived addressed to socket PUPSOC. If there are no pups waiting on PUPSOC, then GETPUP returns NIL, or waits for a pup to arrive if WAIT is T. If WAIT is an integer, GETPUP interprets it as a number of milliseconds to wait, finally returning NIL if a pup does not arrive within that time.

(DISCARDPUPS SOC) [Function]
Discards without examination any pups that have arrived on SOC and not yet been read by a GETPUP.

(EXCHANGEPUPS SOC OUTPUP DUMMY IDFILTER TIMEOUT) [Function]
Sends OUTPUP on SOC, then waits for a responding pup, which it returns. If IDFILTER is true, ignores pups whose PUPID is different from that of OUTPUP. TIMEOUT is the length of time (msecs) to wait for a response before giving up and returning NIL. TIMEOUT defaults to \ETHERTIMEOUT. EXCHANGEPUPS discards without examination any pups that are currently waiting on SOC before OUTPUP gets sent. (DUMMY is ignored; it exists for compatibility with an earlier implementation).

31.5.4 Pup Routing Information
Ordinarily, a program calls SENDPUP and does not worry at all about the route taken to get the pup to its destination. There is an internet routing process in Lisp whose job it is to maintain information about the best routes to networks of interest. However, there are some algorithms for which routing information and/or the topology of the net are explicitly desired. To this end, the following functions are supplied:

(PUPNET.DISTANCE NET#) [Function]
Returns the "hop count" to network NET#, i.e., the number of gateways through which a pup must pass to reach NET#, according to the best routing information known at this point. The local (directly-connected) network is considered to be zero hops away. Current convention is that an inaccessible network is 16 hops away. PUPNET.DISTANCE may need to wait to obtain routing information from an Internetwork Router if NET# is not currently in its routing cache.

(SORT.PUPHOSTS.BY.DISTANCE HOSTLIST) [Function]
Sorts HOSTLIST by increasing distance, in the sense of PUPNET.DISTANCE. HOSTLIST is a list of lists, the CAR of each list being a 16-bit Net/Host address, such as returned by
ETHERHOSTNUMBER. In particular, a list of ports (nethost . socket) pairs is in this format.

(PRINTROUTINGTABLE TABLE SORT FILE)  [Function]
Prints to FILE the current routing cache. The table is sorted by network number if SORT is true. TABLE = PUP (the default) prints the PUP routing table; TABLE = NS prints the NS routing table.

31.5.5 Miscellaneous PUP Utilities

(SETUPPUP PUP DESTHOST DESTSOCKET TYPE ID SOC REQUEUE)  [Function]
Fills in various fields in PUP's header: its length (the header overhead length; assumes data length of zero), TYPE, ID (if ID is NIL, generates a new one itself from an internal 16-bit counter), destination host and socket (DESTHOST may be anything that ETHERPORT accepts; an explicit nonzero socket in DESTHOST overrides DESTSOCKET). If SOC is not supplied, a new socket is opened. REQUEUE fills the packets EPREQUEUE field (see above). Value of SETUPPUP is the socket.

(SWAPPUPPORTS PUP)  [Function]
Swaps the source and destination addresses in PUP. This is useful in simple packet exchange protocols, where you want to respond to an input packet by diddling the data portion and then sending the pup back whence it came.

(GETPUPWORD PUP WORD #)  [Function]
Returns as a 16-bit integer the contents of the WORD # th word of PUP's data portion, counting the first word as word zero.

(PUTPUPWORD PUP WORD # VALUE)  [Function]
Stores 16-bit integer VALUE in the WORD # th word of PUP's data portion.

(GETPUPBYTE PUP BYTE #)  [Function]
Returns as an integer the contents of the BYTE # th 8-bit byte of PUP's data portion, counting the first byte as byte zero.

(PUTPUPBYTE PUP BYTE # VALUE)  [Function]
Stores VALUE in the BYTE # th 8-bit byte of PUP's data portion.
PUP LEVEL ONE FUNCTIONS

(GETPUPSTRING PUP OFFSET) [Function]
Returns a string consisting of the characters in PUP's data portion starting at byte OFFSET (default zero) through the end of PUP.

(PUTPUPSTRING PUP STR) [Function]
Appends STR to the data portion of PUP, incrementing PUP's length appropriately.

31.5.6 PUP Debugging Aids
Tracing facilities are provided to allow the user to see the pup traffic that passes through SENDPUP and GETPUP. The tracing can be verbose, displaying much information about each packet, or terse, which shows a concise “picture” of the traffic.

PUPTRACEFLG [Variable]
Controls tracing information provided by SENDPUP and GETPUP. Legal values:
- NIL No tracing.
- T Every SENDPUP and every successful GETPUP call PRINTPUP of the pup at hand (see below).
- PEEK Allows a concise “picture” of the traffic. For normal, non-broadcast packets, SENDPUP prints "1", GETPUP prints "+". For broadcast packets, SENDPUP prints "↑", GETPUP prints "*". In addition, for packets that arrive not addressed to any socket on this machine (e.g., broadcast packets for a service not implemented on this machine), a "&" is printed.

PUPIGNORETYPES [Variable]
A list of pup types (small integers). If the type of a pup is on this list, then GETPUP and SENDPUP will not print the pup verbosey, but treat it as though PUPTRACEFLG were PEEK. This allows the user to filter out "uninteresting" pups, e.g., routine routing information pups (type 201Q).

PUPONLYTYPES [Variable]
A list of pup types. If this variable is non-NIL, then GETPUP and SENDPUP print verbosely only pups whose types appear on the list, treating others as though PUPTRACEFLG were PEEK. This lets the tracing be confined to only a certain class of pup traffic.

PUPTRACEFILE [Variable]
The file to which pup tracing output is sent by default. The file must be open. PUPTRACEFILE is initially T.
ETHERNET

PUPTRACETIME

[Variable]
If this variable is true, then each printout of a pup is accompanied by a relative timestamp (in seconds, with 2 decimal places) of the current time (i.e., when the SENDPUP or GETPUP was called; for incoming pups, this is not the same as when the pup actually arrived).

(PUPTRACE FLG REGION)

[Function]
Creates a window for puptracing, and sets PUPTRACEFILE to it. If PUPTRACEFILE is currently a window and FLG is NIL, closes the window. Sets PUPTRACEFLG to be FLG. If REGION is supplied, the window is created with that region. The window’s BUTTONEVENTFN is set to cycle PUPTRACEFLG through the values NIL, T, and PEEK when the mouse is clicked in the window.

(PRINTPUP PACKET CALLER FILE PRE.NOTE DOFILTER)

[Function]
Prints the information in the header and possibly data portions of pup PACKET to FILE. If CALLER is supplied, it identifies the direction of the pup (GET or PUT), and is printed in front of the header. FILE defaults to PUPTRACEFILE. If PRE.NOTE is non-NIL, it is PRINT’ed first. If DOFILTER is true, then if PUP’s type fails the filtering criteria of PUPIGNORETYPES or PUPONLYTYPES, then PUP is printed “tersely”, i.e., as a !, +, ↑, or *, as described above.

GETPUP and SENDPUP, when PUPTRACEFLG is non-NIL, call (PRINTPUP PUP ('GET or 'PUT) NIL NIL T).

The form of printing provided by PRINTPUP can be influenced by adding elements to PUPPRINTMACROS.

PUPPRINTMACROS

[Variable]
An association list of elements (PUPTYPE . MACRO) for printing pups. The MACRO (CDR of each element) tells how to print the information in a pup of type PUPTYPE (CAR of the element). If MACRO is a litatom, then it is a function of two arguments (PUP FILE) that is applied to the pup to do the printing. Otherwise, MACRO is a list describing how to print the data portion of the pup (the header is printed in a standard way).

The list form of MACRO consists of "commands" that specify a "datatype" to interpret the data, and an indication of how far that datatype extends in the packet. Each element of MACRO is one of the following: (a) a byte offset (positive integer), indicating the byte at which the next element, if any, takes effect; (b) a negative integer, the absolute value of which is the number of bytes until the next element, if any, takes effect; or (c) an atom giving the format in which to print the data, one of the following:
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BYTES</strong></td>
<td>Print the data as 8-bit bytes, enclosed in brackets. This is the default format to start with.</td>
</tr>
<tr>
<td><strong>CHARS</strong></td>
<td>Print the data as (8-bit) characters. Non-printing characters are printed as if the format were <strong>BYTES</strong>, except that the sequence 15Q, 12Q is printed specially as [crlf].</td>
</tr>
<tr>
<td><strong>WORDS</strong></td>
<td>Print the data as 16-bit integers, separated by commas (or the current <strong>SEPR</strong>).</td>
</tr>
<tr>
<td><strong>INTEGERS</strong></td>
<td>Print the data as 32-bit integers, separated by commas (or the current <strong>SEPR</strong>). Note: the singular <strong>BYTE</strong>, <strong>CHAR</strong>, <strong>WORD</strong>, <strong>INTEGER</strong> are accepted as synonyms for these four commands.</td>
</tr>
<tr>
<td><strong>SEPR</strong></td>
<td>Set the separator for <strong>WORDS</strong> and <strong>INTEGERS</strong> to be the next element of the macro. The separator is initially the two characters, comma, space.</td>
</tr>
<tr>
<td><strong>IFSSTRING</strong></td>
<td>Interprets the data as a 16-bit length followed by that many 8-bit bytes or characters. If the current datatype is <strong>BYTES</strong>, leaves it alone; otherwise, sets it to be <strong>CHARS</strong>.</td>
</tr>
<tr>
<td><strong>FINALLY</strong></td>
<td>The next element of the macro is printed when the end of the packet is reached (or printing stops because of a ...). This command does not alter the datatype, and can appear anywhere in the macro as long as it is encountered before the actual end of the packet.</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>Perform a TERPRI.</td>
</tr>
<tr>
<td><strong>REPEAT</strong></td>
<td>The remainder of the macro is itself treated as a macro to be applied over and over until the packet is exhausted. Note that the offsets specified in the macro must be in the relative form, i.e., negative integers. For example, the macro (<strong>INTEGERS</strong> 4 <strong>REPEAT</strong> <strong>BYTES</strong> -2 <strong>WORDS</strong> -4) says to print the first 4 bytes of the data as one 32-bit integer, then print the rest of the data as sets of 2 8-bit bytes and 2 16-bit words. Only as much of the macro is processed as is needed to print the data in the given packet. The default macro for printing a pup is (<strong>BYTES</strong> 12 ...), meaning to print the first up to 12 bytes as bytes, and then print &quot;...&quot; if there is anything left.</td>
</tr>
</tbody>
</table>

```plaintext
(PUP.ECHOUSER HOST ECHOSTREAM INTERVAL NTIMES) [Function]
```

Sends dummy packets to be echoed by the host **HOST**. Can be used as a simple test of the functioning of the Ethernet and the host. **HOST** is the pup host to send the packets to. **ECHOSTREAM** is the stream for printing status information. **INTERVAL** is the interval (in milliseconds) to wait for the packet to be echoed (default 1000). **NTIMES** is the number of packets to send (default 1000).
As each packet is sent and received, characters are printed to ECHOSTREAM as follows:

! Printed when a packet is sent.
+ Printed when an echo packet is successfully received.
. Printed when an echo packet has not been received after INTERVAL milliseconds.
? Printed when a packet is received, but it isn't an echo packet or an error packet.
(late) Printed when an error packet is received, after the echo request timed out.

The trace can be used to test the functioning of the ethernet and host. For example, if the trace is ! + + ! + + + , the host is listening and echoing correctly. ! + + . ! + + indicates that for some reason the host is not responding. ! + + . ! (late) . . (late) + indicates that the packets are being echoed, but not immediately.

The following functions are used by PRINTPUP and similar functions, and may be of interest in special cases.

(PORTSTRING NETHOST SOCKET) [Function]
Converts the pup address NETHOST, SOCKET into octal string format as follows: NET#HOST#SOCKET. NETHOST may be a port (dotted pair of nethost and socket), in which case SOCKET is ignored, and the socket portion of NETHOST is omitted from the string if it is zero.

(PRINTPUPROUTE PACKET CALLER FILE) [Function]
Prints the source and destination addresses of pup PACKET to FILE in the PORTSTRING format, preceded by CALLER (interpreted as with PRINTPUP).

(PRINTPACKETDATA BASE OFFSET MACRO LENGTH FILE) [Function]
Prints data according to MACRO, which is a list interpreted as described under PUPPRINTMACROS, to FILE. The data starts at BASE and extends for LENGTH bytes. The actual printing starts at the OFFSETth byte, which defaults to zero. For example, PRINTPUP ordinarily calls (PRINTPACKETDATA (fetch PUPCONTENTS of PUP) 0 MACRO (IDIFFERENCE (fetch PUPLENGTH of PUP) 20) FILE).

(PRINTCONSTANT VAR CONSTANTLIST FILE PREFIX) [Function]
CONSTANTLIST is a list of pairs (VARNAME VALUE), of the form given to the CONSTANTS File Package Command. PRINTCONSTANT prints VAR to FILE, followed in parentheses by
the VARNAME out of CONSTANTLIST whose VALUE is EQ to VAR, or ? if it finds no such element. If PREFIX is non-NIL and is an initial substring of the selected VARNAME, then VARNAME is printed without the prefix.

For example, if FOOCONSTANTS is ((FOO.REQUEST 1) (FOO.ANSWER 2) (FOO.ERROR 3)), then (PRINTCONSTANT 2 FOOCONSTANTS T "FOO.") produces "2 (ANSWER).

(OCTALSTRING N) [Function]
Returns a string of octal digits representing N in radix 8.

31.6 NS Level One Functions
The functions in this section are used to implement level two and higher NS protocols. The packets used in the NS protocol are termed Xerox Internet Packets (XIPs). The functions for manipulating XIPs are similar to those for managing PUPs, so will be described in less detail here. The major difference is that NS host addresses are 48-bit numbers. Since Interlisp-D cannot currently represent 48-bit numbers directly as integers, there is an interim form called NSHOSTNUMBER, which is defined as a TYPE RECORD of three fields, each of them being a 16-bit portion of the 48-bit number.

31.6.1 Creating and Managing XIPs
There is a record XIP that overlays the data portion of an ETHERPACKET and describes the format of a XIP. This record defines the following fields: XIPLENGTH (16 bits), XIPTCONTROL (transmit control, 8 bits, cleared when a XIP is transmitted), XIPTYPE (8 bits), XIPDESTNET (32 bits), XIPDESTHOST (an NSHOSTNUMBER), XIPDESTSOCKET (16 bits), and XIPSOURCENET, XIPSOURCEHOST, and XIPSOURCESOCKET, analogously. The field XIPCONTENTS is a pointer to the start of the data portion of the XIP.

(ALLOCATE.XIP) [Function]
Returns a (possibly used) XIP. As with ALLOCATE.PUP, the header fields are guaranteed to be zero, but there may be garbage in the data portion if the pup had been recycled.

(RELEASE.XIP XIP) [Function]
Releases XIP to the free pool.
31.6.2 NS Sockets

As with pups, XIPs are sent and received on a socket. The same comments apply as with pup sockets (page 31.29), except that NS socket numbers are only 16 bits.

**OPENNSOCKET SKT# IFCLASH**

[Function]

Opens a new NS socket. If SKT# is NIL (the normal case), a socket number is chosen automatically, guaranteed to be unique, and probably different from any socket opened this way in the last 18 hours. If a specific local socket is desired, as is typically the case when implementing a server, SKT# is given, and must be a (up to 16-bit) number. IFCLASH governs what to do if SKT# is already in use: if IFCLASH is NIL, an error is generated; if IFCLASH is ACCEPT, the socket is quietly returned; if IFCLASH is FAIL, then OPENNSOCKET returns NIL without causing an error.

**CLOSENSOCKET NSOC NOERRORFLG**

[Function]

Closes and releases socket NSOC. If NSOC is T, closes all NS sockets (this must be used with caution, since it will also close system sockets!). If NSOC is already closed, an error is generated unless NOERRORFLG is true.

**NSOCKETNUMBER NSOC**

[Function]

Returns the socket number (a 16-bit integer) of NSOC.

**NSOCKETEVENT NSOC**

[Function]

Returns the EVENT of NSOC. This event is notified whenever a XIP arrives on NSOC.

31.6.3 Sending and Receiving XIPs

**SENDXIP NSOC XIP**

[Function]

Sends XIP on socket NSOC. If any of the XIPSOURCESHOST, XIPSOURCENET, or XIPSOURCESOCKET fields is zero, SENDXIP fills them in using the NS address of this machine and/or the socket number of NSOC, as needed.

**GETXIP NSOC WAIT**

[Function]

Returns the next XIP that has arrived addressed to socket NSOC. If there are no XIPs waiting on NSOC, then GETXIP returns NIL, or waits for a XIP to arrive if WAIT is T. If WAIT is an integer, GETXIP interprets it as a number of milliseconds to wait, finally returning NIL if a XIP does not arrive within that time.
31.6.4 NS Debugging Aids

XIPs can be printed automatically by SENDXIP and GETXIP analogously to the way pups are. The following variables behave with respect to XIPs the same way that the corresponding PUP-named variables behave with respect to PUPs: XPTRACEFLG, XPTRACEFILE, XPIGNORETYPES, XIPONLYTYPES, XIPPRINTMACROS. In addition, the functions PRINTXIP, PRINTXIPROUTE, XPTRACE, and NS.ECHOUSER are directly analogous to PRINTPUP, PRINTPUPROUTE, PUPTRACE, and PUP.ECHOUSER. See page 31.32.

31.7 Support for Other Level One Protocols

Raw packets other than of type PUP or NS can also be sent and received. This section describes facilities to support such protocols. Many of these functions have a \ in their names to designate that they are system internal, not to be dealt with as casually as user-level functions.

(RESTART.ETHER) [Function]
This function is intended to be invoked from the executive on those rare occasions when the Ethernet appears completely unresponsive, due to Lisp having gotten into a bad state. RESTART.ETHER reinitializes Lisp's Ethernet driver(s), just as when the Lisp system is started up following a LOGOUT, SYSOUT, etc. This aborts any Ethernet activity and clears several internal caches, including the routing table.
(\ALLOCATE.ETHERPACKET) [Function]

Returns an ETHERPACKET datum. Enough of the packet is cleared so that if the packet represents a PUP or NS packet, that its header is all zeros; no guarantee is made about the remainder of the packet.

(RELEASE.ETHERPACKET EPKT) [Function]

Returns EPKT to the pool of free packets. This operation is dangerous if the caller actually is still holding on to EPKT, e.g., in some queue, since this packet could be returned to someone else (via \ALLOCATE.ETHERPACKET) and suffer the resulting contention.

From a logical standpoint, programs need never call \RELEASE.ETHERPACKET, since the packets are eventually garbage-collected after all pointers to them drop. However, since the packets are so large, normal garbage collections tend not to occur frequently enough. Thus, for best performance, a well-disciplined program should explicitly release packets when it knows it is finished with them.

A locally-connected network for the transmission and receipt of Ether packets is specified by a network descriptor block, an object of type NDB. There is one NDB for each directly-connected network; ordinarily there is only one. The NDB contains information specific to the network, e.g., its PUP and NS network numbers, and information about how to send and receive packets on it.

\LOCALNDBS [Variable]

The first NDB connected to this machine, or NIL if there is no network. Any other NDBs are linked to this first one via the NDBNEXT field of the NDB.

In order to transmit an Ether packet, a program must specify the packet's type and its immediate destination. The type is a 16-bit integer identifying the packet's protocol. There are preassigned types for PUP and NS. The destination is a host address on the local network, in whatever form the local network uses for addressing; it is not necessarily related to the logical ultimate destination of the packet. Determining the immediate destination of a packet is the task of routing. The functions SENDPUP and SENDXIP take care of this for the PUP and NS protocols, routing a packet directly to its destination if that host is on the local network, or routing it to a gateway if the host is on some other network accessible via the gateway. Of course, a gateway must know about the type (protocol) of a packet in order to be able to forward it.
SUPPORT FOR OTHER LEVEL ONE PROTOCOLS

**ENCAPSULATE.ETHERPACKET** *NDB Packet PDU NBYTES ETYPE*  
**[Function]**
Encapsulates *PACKET* for transmission on network *NDB*. *PDH* is the physical destination host (e.g., an 8-bit PUP host number or a 48-bit NS host number); *NBYTES* is the length of the packet in bytes; *ETYPE* is the packet's encapsulation type (an integer).

**TRANSMIT.ETHERPACKET** *NDB Packet*  
**[Function]**
Transmits *PACKET*, which must already have been encapsulated, on network *NDB*. Disposition of the packet after transmission is complete is determined by the value of *PACKET*'s *EPREQUEUE* field.

In order to receive Ether packets of type other than PUP or NS, the programmer must specify what to do with incoming packets. Lisp maintains a set of packet filters, functions whose job it is to appropriately dispose of incoming packets of the kind they want. When a packet arrives, the Ethernet driver calls each filter function in turn until it finds one that accepts the packet. The filter function is called with two arguments: (*PACKET TYPE*), where *PACKET* is the actual packet, and *TYPE* is its Ethernet encapsulation type (a number). If a filter function accepts the packet, it should do what it wants to with it, and return T; else it should return NIL, allowing other packet filters to see the packet. Since the filter function is run at interrupt level, it should keep its computation to a minimum. For example, if there is a lot to be done with the packet, the filter function can place it on a queue and notify another process of its arrival.

The system already supplies packet filters for packets of type PUP and NS; these filters enqueue the incoming packet on the input queue of the socket to which the packet is addressed, after checking that the packet is well-formed and indeed addressed to an existing socket on this machine.

Incoming packets have their *EPNETWORK* field filled in with the *NDB* of the network on which the packet arrived.

**ADD.PACKET.FILTER** *FILTER*  
**[Function]**
Adds function *FILTER* to the list of packet filters if it is not already there.

**DEL.PACKET.FILTER** *FILTER*  
**[Function]**
Removes *FILTER* from the list of packet filters.

**CHECKSUM** *BASE NWOROS INITSUM*  
**[Function]**
Computes the one's complement add and cycle checksum for the *NWOROS* words starting at address *BASE*. If *INITSUM* is supplied, it is treated as the accumulated checksum for some set of words.
SUPPORT FOR OTHER LEVEL ONE PROTOCOLS

preceding BASE; normally INITSUM is omitted (and thus treated as zero).

(PRINTPACKET PACKET CALLER FILE PRE.NOTE DOFILTER) [Function]
Prints PACKET by invoking a function appropriate to PACKET's type. See PRINTPUP for the intended meaning of the other arguments. In order for PRINTPACKET to work on a non-standard packet, there must be information on the list \PACKET.PRINTERS.

\PACKET.PRINTERS [Variable]
An association list mapping packet type into the name of a function for printing that type of packet.

31.8 The SYSQUEUE mechanism

The SYSQUEUE facility provides a low-level queueing facility. The functions described herein are all system internal: they can cause much confusion if misused.

A SYSQUEUE is a datum containing a pointer to the first element of the queue and a pointer to the last; each item in the queue points to the next via a pointer field located at offset 0 in the item (its QLINK field in the QABLEITEM record). A SYSQUEUE can be created by calling (NCREATE 'SYSQUEUE).

(\ENQUEUE Q ITEM) [Function]
Enqueues ITEM on Q, i.e., links it to the tail of the queue, updating Q's tail pointer appropriately.

(\DEQUEUE Q) [Function]
Removes the first item from Q and returns it, or returns NIL if Q is empty.

(\UNQUEUE Q ITEM NOERRORFLG) [Function]
Removes the ITEM from Q, wherever it is located in the queue, and returns it. If ITEM is not in Q, causes an error, unless NOERRORFLG is true, in which case it returns NIL.

(\QUEUELENGTH Q) [Function]
Returns the number of elements in Q.

(\ONQUEUE ITEM Q) [Function]
True if ITEM is an element of Q.
[This page intentionally left blank]
INDEX
ANTILOG X I: 7.13

*ANY* (in edit pattern) II: 16.18

APPEND (File access) III: 24.2

APPEND X₁ X₂ ... X_N I: 3.5

APPENDTOVAR VAR X₁ X₂ ... X_N II: 17.55; 17.36

APPENDVARS (VAR₁..LST₁) ... (VARₙ..LSTₙ) (File
Package Command) II: 17.36

APPLY FN ARGLIST — I: 10.11; II: 18.19

APPLY* FN ARG₁ ARG₂ ... ARGₙ I: 10.12; II: 18.19

APPLY-format input II: 13.4

Applying functions to arguments I: 10.11

Approval of DWIM corrections II: 20.4; 20.3

APPROVEFLG (Variable) II: 20.14; 20.22.24

APPROPOSgetString ALLFLG QUIETFLG OUTPUT I: 2.11

Arbitrary-size integers I: 7.1

ARCCOS X RADIUSFLG I: 7.14

ARCCOS: ARG NOT IN RANGE (Error Message) I: 7.14

*ARCHIVE* (history list property) II: 13.33

ARCHIVE EventSpec (Prog. Asst. Command) II: 13.16

ARCHIVEFLG (Variable) II: 13.23

ARCHIVEFN (Variable) II: 13.23; 13.16

ARCHIVELST (Variable) II: 13.31; 13.16

ARCSIN X RADIUSFLG I: 7.14

ARCSIN: ARG NOT IN RANGE (Error Message) I: 7.14

ARCTAN X RADIUSFLG I: 7.14

ARCTAN2 Y X RADIUSFLG I: 7.14

SET ARE SET (Masterscope Command) II: 19.5

(ARG VAR M) I: 10.5

*ARGN (Stack blip) I: 11.15

ARG NOT ARRAY (Error Message) I: 5.1-2; II: 14.30

ARG NOT HARRAY (Error Message) II: 14.31

ARG NOT LIST (Error Message) I: 3.2, 5, 15-16; II: 14.28

ARG NOT LITATOM (Error Message) I: 2.3, 5, 7, 9.8;
10.3, 11; II: 14.28; 17.54

(ARGLIST FN) I: 10.8; II: 14.10

ARGNAMES (Property Name) I: 10.8

ARGS (Break Command) II: 14.10

...ARGS (history list property) II: 13.33

ARGS NOT AVAILABLE (Error Message) I: 10.8

(ARGTYPE FN) I: 10.7

Argument lists of functions I: 10.2

*ARGVAL* (stack blip) I: 11.16

Arithmetic I: 7.1

AROUND (as argument to ADVISE) II: 15.10;
15.11-12

AROUND (as argument to BREAKIN) II: 15.6; 14.5

ARRAY SIZE TYPE INIT ORIG —) I: 5.1

ARRAYORIG ARRAY I: 5.2

ARRAYP X I: 5.1; 9.2

ARRAYRECORD (Record Type) I: 8.8

Arrays I: 5.1; 9.2

ARRAYS FULL (Error Message) II: 14.29; 22.5

ARRAYSIZE ARRAY I: 5.2

ARRAYTYP ARRAY I: 5.2

AS VAR (I.S. Operator) I: 9.15

ASCENT (Font property) III: 27.27

(ASKUSER WAIT DEFAULT MESS KEYLST
TYPEAHEAD LISPXPRTNLFLG OPTIONS/LST FILE)
III: 26.12

ASKUSERTTBL (Variable) III: 26.17

Assignments in CLISP II: 21.9

Assignments in pattern matching I: 12.28

(ASSOC KEY ALST) I: 3.15

Association lists I: 3.15

Association lists in EVALA I: 10.13

ASSOCRECORD (Record Type) I: 8.8

(ATOM X) I: 2.1; 9.1

ATOM HASH TABLE FULL (Error Message) II: 14.28

ATOM TOO LONG (Error Message) I: 2.2; II: 14.28

ATOMRECORD (Record Type) I: 8.9

Atoms I: 2.1; 9.1

(ATTACH X L) I: 3.5

Attached windows III: 28.45; 28.1

(ATTACHEDWINDOWS WINDOW COM) III: 28.47

ATTACHEDWINDOWS (Window Property) III: 28.54

(ATTACHMENUMENU MAINWINDOW EDGE
POSITIONONEDGE NOOPENFLG) III: 28.48

(ATTACHWINDOW WINDOW TO ATTACH
MAINWINDOW EDGE POSITIONONEDGE
WINDOW COM ACTION) III: 28.45

ATTEMPT TO BIND NIL OR T (Error Message) I: 9.8;
10.3; II: 14.30
INDEX

attempt to read DATATYPE with different field specification than currently defined (Error Message) III: 25.18
ATTEMPT TO RPLAC NIL (Error Message) I: 3.2; II: 14.28
ATTEMPT TO SET NIL (Error Message) I: 2.3; II: 14.28
ATTEMPT TO SET T (Error Message) I: 2.3
ATTEMPT TO USE ITEM OF INCORRECT TYPE (Error Message) II: 14.30
(AU-REVOIR VAL) I: 11.21
AUTHOR (File Attribute) III: 24.18
AUTOBACKTRACEFLG (Variable) II: 14.15
AUTOCOMPLETEFLG (ASKUSER option) III: 26.17
AUTOPROCESSFLG (Variable) II: 23.1
&AUX (DEFMACRO keyword) I: 10.26
AVOIDING SET (Masterscope Path Option) II: 19.16
(AWAIT.EVENT EVENT TIMEOUT TIMERP) II: 23.7

B
(B E1 ... E M) (Editor Command) II: 16.32
Background menu III: 28.6
Background shade III: 30.22
BACKGROUNDBUTTONEVENTFN (Variable) III: 28.29
BackgroundCopyMenu (Variable) III: 28.8
BackgroundCopyMenuCommands (Variable) III: 28.8
BACKGROUNDCURSORINFN (Variable) III: 28.29
BACKGROUNDCURSORMOVEDFN (Variable) III: 28.29
BACKGROUNDCURSOROUTFN (Variable) III: 28.29
BackgroundMenu (Variable) III: 28.8
BackgroundMenuCommands (Variable) III: 28.8
BACKGROUNDPAGEFREQ (Variable) I: 12.10
BACKGROUNDWHENSELECTEDFN (Function) III: 28.40
Backquote (') III: 25.42
Backslash functions I: 10.10
Backspace III: 30.5; 25.2; 26.23
(BACKTRACE IPOE EPOS FLAGS FILE PRINTFN) I: 11.11
Backtrace break commands II: 14.9
Backtrace frame window II: 14.3
Backtrace functions I: 11.11
BACKTRACEFONT (Variable) II: 14.15
BAD FILE NAME (Error Message) II: 14.31
BAD FILE PACKAGE COMMAND (Error Message) II: 17.34
BAD PROG BINDING (Error Message) II: 18.23
BAD SETQ (Error Message) II: 18.23
BAD SYSOUT FILE (Error Message) II: 14.29
(BAKTRACE IPOE EPOS SKIPFNS FLAGS FILE) I: 11.11
BAKTRACEST (Variable) I: 11.12
Bars on cursor III: 30.16
.BASE (PRINTOUT command) III: 25.27
Basic frames on stack I: 11.3; 11.1,6
(BCOMPL FILES CFILE — ) II: 18.21; 18.17-18
(BEEPOFF) III: 30.24
(BEEPON FREQ) III: 30.24
BEFORE (as argument to ADVISE) II: 15.10; 15.11
BEFORE (as argument to BREAKIN) II: 15.6; 14.5
Before (DEdit Command) II: 16.7
BEFORE (in INSERT editor command) II: 16.33
BEFORE (in MOVE editor command) II: 16.38
BEFORE LITATOM (Prog. Asst. Command) II: 13.15; 13.24,33
BEFOREEXIT (Process Property) II: 23.3
BEFRESYSOUTFORMS (Variable) I: 12.9
\BeginDST (Variable) I: 12.9
Bell (in history event) II: 13.19; 13.13,31,39
Bell in terminal III: 30.24
Bells printed by DWIM II: 20.3
(BELOW COM X) (Editor Command) II: 16.25
(BELOW COM) (Editor Command) II: 16.25
BF PATTERN NIL (Editor Command) II: 16.23
(BF PATTERN) (Editor Command) II: 16.23
BF PATTERN T (Editor Command) II: 16.23
BF PATTERN (Editor Command) II: 16.23
(BIN M) (Editor Command) II: 16.40
(BIN N) (Editor Command) II: 16.41
Bignums I: 7.1
(BIN STREAM) III: 25.5
(BIND COMS1 ... COMSN) (Editor Command) II: 16.63
BIND VARS (I.S. Operator) I: 9.12
BIND VAR (I.S. Operator) I: 9.12
BIND (in Masterscope template) II: 19.20
BIND (Masterscope Relation) II: 19.9
Bindings in stack frames I: 11.6
BINDS (Litatom) II: 21.21
BIR (Font face) III: 27.26
Bit tables I: 4.6
(BITBLT SOURCE SOURCELEFT SOURCEBOTTOM DESTINATION DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT)
INDEX

SOURCE TYPE OPERATION TEXTURE
CLIPPINGREGION) III: 27.14

(BitClear N MASK) (Macro) I: 7.9

Bitmap (Data Type) III: 27.3

(BitMapBit BITMAP X Y NEWVALUE) III: 27.3

(BitMapCopy BITMAP) III: 27.4

(BitMapCreate WIDTH HEIGHT BITS PERPIXEL) III: 27.3

(BitMapHeight BITMAP) III: 27.3

(BitMapImagesize BITMAP DIMENSION STREAM) III: 27.16

(BitMap X) III: 27.3

Bitmaps III: 27.3

(BitMapWidth BITMAP) III: 27.3

Bits (as a field specification) I: 8.21

Bits (record field type) I: 8.10

(BitSet N MASK) (Macro) I: 7.9

(BitsPerPixel BITMAP) III: 27.3

(BitTest N MASK) (Macro) I: 7.9

 BN (Editor Command) II: 16.16

BK (Editor Command) II: 16.16

(BkLinbuf Str) III: 30.12

(Bksysbuf X Flg RdTBl) III: 30.11; 30.12

BlackShade (Variable) III: 27.7

Blk (in stack frame) I: 11.3

Blips on the stack I: 11.14

(BlipScan BlipTyp Ipos) I: 11.16

(BlipVal BlipTyp Ipos Flg) I: 11.16

BkApply (Function) II: 18.19

BkApply* (Function) II: 18.19

BkApplyFns (in masterscope set specification) II: 19.12

BkApplyFns (Variable) II: 18.19; 18.18

Bkfns (in masterscope set specification) II: 19.12

BkLibrary (Variable) II: 18.20; 18.18

BkLibraryDef (Property Name) II: 18.20

BkName (Variable) II: 18.18

(Block MsEcsWait Timer) II: 23.5

Block compiling II: 18.17

Block compiling functions II: 18.20

Block declarations II: 18.17; 17.42

Block library II: 18.19

(BlockCompile BlkName Bkfns Entries Flg) II: 18.20; 18.18

Blocked (Printed by Editor) II: 16.65

BlockRecord (Record Type) I: 8.11

(Blocks Block ... block N (file package command) II: 17.42; 18.17

(Bit SHADE TEXTURE DESTINATION
DESTINATION LEFT DESTINATION BOTTOM
WIDTH HEIGHT OPERATION CLIPPINGREGION) III: 27.16

(BO N) (Editor Command) II: 16.41

&Body (DefMacro keyword) I: 10.25

BoldItalic (Font face) III: 27.26

Border (Window Property) III: 28.33

Both (File access) III: 24.2

(Both Template 1 Template 2) (in masterscope template) II: 19.20

Bottom (as argument to ADVISE) II: 15.11

Bottom margin III: 27.11

(BottomOfGridCoord Gridy GridSpec) III: 27.23

(Boundp Var) II: 2.3

(Bound Stream Byte) III: 25.9

(BoxCount Type N) II: 22.8

BoxCursor (Variable) III: 28.9; 30.15

Boxing numbers II: 7.1

Boyer-Moore fast string searching algorithm III: 25.21

BQuote (Function) III: 25.42

Break (DEdit Command) II: 16.9

Break (Error Message) II: 14.29

(Break X) II: 15.5; 14.5; 15.1,7

**Break** (in backtrace) II: 14.9

Break (Interrupt Channel) II: 23.15; III: 30.3

Break (Syntax Class) III: 25.37

Break characters III: 25.36; 25.4; 30.10

Break commands II: 14.5; 14.17

Break expression II: 14.5; 14.12

Break Inserted After (Printed by BREAKIN) II: 15.7

Break package II: 14.1

Break windows II: 14.3; 14.1

Break within a break on FN (Printed by system) II: 14.16

(Break.Nsfiling.Connection Host) III: 24.38

(Break0 Fn When Coms -----) II: 15.4; 15.5.8

(Break1 BrkExp BrkWhen Brkfn BrkComs
BrkType Errorn) II: 14.16; 14.20; 15.1,3-6; 20.24

BreakChar (Syntax Class) III: 25.35

(BreakCheck Errorpos Erxn) II: 14.13; 14.19,22,27

BreakChk (Variable) II: 14.23

BreakComsLst (Variable) II: 14.17

BreakConnection (Function) III: 24.37

INDEX 4
BREAKDELEMITER (Variable) II: 14.10
(BREAKDOWN FN1 ... FNn) II: 22.9
(BREAKIN FN WHERE WHEN COMS) II: 15.6; 14.5;
15.1,3,4-7-8
Breaking CLISP expressions II: 15.4
Breaking functions II: 15.1
BREAKMACROS (Variable) II: 14.17; 14.16
(BREAKREAD TYPE) II: 14.18
BREAKREGIONSPEC (Variable) II: 14.15
BREAKRESETFORMS (Variable) II: 14.18
(BRECOMPILE FILES CFN FNS —) II: 18.21; 17.12;
18.17-18
BRKCOMS (Variable) II: 14.17; 14.7-8,16; 15.4
BRKDNCOMPFLG (Variable) II: 22.11
(BRKDWNRESULTS RETURNVALUESFLG) II: 22.9
BRKDWNTYPE (Variable) II: 22.10; 22.11
BRKDWNTYPES (Variable) II: 22.10
BRKEXP (Variable) II: 14.5; 14.8,11-12,16; 15.4
BRKFILE (Variable) II: 14.17
BRKFN (Variable) II: 14.16; 14.6; 15.4
BRKINFO (Property Name) II: 15.4,7-8
BRKINFLST (Variable) II: 15.7-8
BRKTYPE (Variable) II: 14.16
BRKWHEN (Variable) II: 14.16; 15.4
BROADSCOPE (Property Name) II: 21.28
BROKEN (Property Name) II: 10.9; II: 15.4
BROKEN-IN (Property Name) I: 10.9; II: 15.7; 15.8
BROKENFNS (Variable) II: 15.4,7; 20.24
Brushes for drawing curves III: 27.18
BT (Break Command) II: 14.9
BT (Break Window Command) II: 14.3
BT! (Break Window Command) II: 14.3
BTV (Break Command) II: 14.9
BTV! (Break Command) II: 14.9
BTV* (Break Command) II: 14.9
BTV+ (Break Command) II: 14.9
BUF (Editor Command) III: 26.29
BUFFERS (File Attribute) III: 24.19
BUILDMAPFLG (Variable) II: 17.56; 17.5; 18.15
Bulk Data Transfer III: 31.24
Bury (Window Menu Command) III: 28.4
(BURYW WINDOW) III: 28.20
BUTTONEVENTFN (Window Property) III: 28.28;
28.38
(BUTTONEVENTINFN IMAGEOBJ WINDOWSTREAM
SELECTION RELX RELY WINDOW HOSTSTREAM
BUTTON) (IMAGEFNS Method) III: 27.38
Buttons on mouse III: 30.17
BY FORM (without IN/ON) (I.S. Operator) I: 9.14
BY FORM (with IN/ON) (I.S. Operator) I: 9.14; 9.18
BY (in REPLACE editor command) II: 16.33
BYTE (as a field specification) I: 8.21
BYTE SIZE POSITION (Macro) I: 7.10
BYTE (record field type) I: 8.10
(BYTEPOSITION BYTESPEC) (Macro) I: 7.10
BYTESIZE (File Attribute) III: 24.17
(BYTESIZE BYTESPEC) (Macro) I: 7.10
C
C (MAKEFILE option) II: 17.10
C...R functions I: 3.2
CAAR (Function) I: 3.2
CADR (Function) I: 3.2
CALCULATEREGION (Window Property) III: 28.20
CALL (in Masterscope template) II: 19.20
CALL (Masterscope Relation) II: 19.7
CALL DIRECTLY (Masterscope Relation) II: 19.8
CALL FOR EFFECT (Masterscope Relation) II: 19.9
CALL FOR VALUE (Masterscope Relation) II: 19.9
CALL INDIRECTLY (Masterscope Relation) II: 19.8
CALL SOMEHOW (Masterscope Relation) II: 19.8
(CALLS FN USEDATABASE —) II: 19.22
(CALLSCODE FN — —) II: 19.22
CAN'T - AT TOP (Printed by Editor) II: 16.15
CAN'T BE BOTH AN ENTRY AND THE BLOCK NAME
(Error Message) II: 18.22; 18.20
CAN'T FIND EITHER THE PREVIOUS VERSION ...
(Printed by System) II: 17.16
CANFILEDEF (File Package Type Property) II: 17.30
(CANONICAL.HOSTNAME HOSTNAME) III: 24.39
CAP (Editor Command) II: 16.52
(CAR X) I: 3.1
CAR/CDERRR (Variable) I: 3.1
#CAREFULCOLUMNs (Variable) III: 26.47
(CARET NEWCAREt) III: 28.31
(CARETRATE ONRATE OFFRATE) III: 28.31
Caret s III: 28.30
Carriage-return II: 13.37; III: 25.8; 25.4
Case arrays III: 25.21
(CASEARRAY OLDARRAY) III: 25.21
CAUTIOUS (DWIM mode) II: 20.4; 20.3,24; 21.4,6
CCODEP (data type) I: 10.6
(CCCODEP FN) I: 10.7
CDAR (Function) I: 3.2
CDDR (Function) I: 3.2
(CDR X) I: 3.1
Center (DEdit Command) II: 16.8
.CENTER POS EXPR (PRINTOUT command) III: 25.29
INDEX

CLISPCHARRAY (Variable) II: 21.25
CLISPCHARS (Variable) II: 21.25
(CLISPDEC DECLST) II: 21.12; 21.25
CLISPFLG (Variable) II: 21.25
CLISPFONT (Font class) III: 27.32
CLISPFORWORDSPLST (Variable) I: 9.10
CLISHPHELPFLG (Variable) II: 21.21; 21.6
CLISPI.S.GAG (Variable) I: 9.20
CLISPIFTRANFLG (Variable) II: 21.26
CLISPIFY (Variable) II: 17.11; 21.14
CLISPIFY (MAKEFILE option) II: 17.11; 21.26
(CLISPIFYFNs FNT .. FNNT) II: 21.23
CLISPIFYPACKFLG (Variable) II: 21.24
CLISPIFYPRETTYFLG (Variable) I: 12.3; II: 21.26;
17.11; III: 26.48
CLISPIFYUSERFN (Variable) II: 21.24
CLISPINFIX (Property Name) II: 21.29
CLISPINFIXSPLST (Variable) II: 21.25; 21.9
CLISPRECORDTYPES (Variable) I: 8.15
CLISPRETRANFLG (Variable) II: 21.22; 21.17
(CLISPRTRAN X TRAN) II: 21.25
CLISPTYPE (Property Name) II: 21.27; 21.28
CLISPW (Property Name) I: 8.19; II: 21.29
(CLOCK N —) I: 12.15
Close (Window Menu Command) III: 28.3
(CLOSEALL ALLFLG) III: 24.5; 24.20
CLOSEBREAKWINDOWFLG (Variable) II: 14.15
(CLOSE FILE) III: 24.4
(CLOSEFILE FILE) III: 24.4
CLOSEFN (Window Property) III: 28.15
(CLOSENSOCKET NSOC NOERRORFLG) III: 31.37
(CLOSEUPSOCKET PUSOC NOERRORFLG) III:
31.29
(CLOSEWINDOW) III: 28.15
Closing and reopening files III: 24.20
CLEMPARSFLG (Variable) II: 21.23
(CLRHASH HARRAY) I: 6.2
(CLRPROMPT) III: 28.3
(CNDIR HOST/DIR) III: 24.10
CNTRLV (syntax class) III: 30.6
CODE (Property Name) II: 17.27
CODERDDBL (Variable) III: 25.34
COLLECT FORM (I.S. Operator) I: 9.10
COMMAND (Variable) III: 26.38
**COMMENT** (printed by editor) II: 16.48
**COMMENT** (printed by system) III: 26.43
Comment pointers II: 16.55; III: 26.44
COMMENT USED FOR VALUE (Error Message) II:
18.23
**COMMENT**FLG (Variable) III: 26.43
(COMMENTL L —) III: 26.43
COMMENTFLG (Variable) III: 26.43; 26.45
COMMENTFONT (Font class) III: 27.32
COMMENTLINELENGTH (Variable) III: 27.34
Comments in functions III: 26.42
(COMPAR NAME1 NAME2 TYPE SOURCE1
SOURCE2) II: 17.29
(COMPARDEFS NAME TYPE SOURCES) II: 17.29
(COMPARELISTS X Y) I: 3.19
Comparing lists I: 3.19
(COMPIL X FLG) II: 18.14
COMPILE.EXT (Variable) II: 18.13
(COMPILE1 FN DEF —) II: 18.14
Compiled files II: 18.13
Compiled function objects I: 10.6
COMPILED ON (printed when file is loaded) II:
18.13
(COMPILEFILES FILE1 FILE2 ... FILEN) II: 17.14
COMPILEHEADER (Variable) II: 18.13
Compiler II: 18.1
Compiler error messages II: 18.22
Compiler functions II: 18.13; 18.20
Compiler printout II: 18.3
Compiler questions II: 18.1
COMPILERMACROPROPS (Variable) I: 10.22
COMPILETYPELIST (Variable) I: 10.14; II: 18.11; 18.9
COMPILERUSERFN (Function) II: 18.12
COMPILEUSERFN (Variable) II: 18.9; 18.11
Compiling CLISP II: 18.11; 18.9,14
Compiling data types II: 18.11
Compiling files II: 18.14; 18.21
Compiling FUNCTION II: 18.10
Compiling function calls II: 18.8
Compiling functional arguments II: 18.10
Compiling open functions II: 18.11
COMPLETEON (ASKUSER option) III: 26.16
COMPSET (Function) II: 18.1
Computed macros I: 10.23
(COMSP X1 ... XM) (Editor Command) II: 16.59
(COMSP COM1 ... COMn) (File Package Command)
II: 17.40
(CONSP COM1 ... COMn) (Editor Command) II:
16.59
(CONCAT X1 X2 ... Xn) I: 4.4
(CONCATLIST L) I: 4.4
(COND CLAUSE₁ CLAUSE₂ ... CLAUSEₖ) I: 9.4
COND clause I: 9.4
CONFIRMFLG (ASKUSER option) III: 26.15
Conjunctions in Masterscope II: 19.14
CONN HOST/DIR (Prog. Asst. Command) III: 24.11
Connected directory III: 24.9
Connection Lost (Error Message) III: 24.41
(CONS X Y) I: 3.1
(CONSCOUNT N) II: 22.8
(CONSTANT X) II: 18.7
(CONSTANTS VAR₁ ... VARₙ) (File Package Command) II: 17.37
(CONSTANTS VAR₁ VAR₂ ... VARₙ) II: 18.8
Constants in compiled code II: 18.7
Constructing lists in CLISP II: 21.10
CONTAIN (File Package Command Property) II: 17.46
CONTAIN (Masterscope Relation) II: 19.10
CONTENTS (File Package Command Property) II: 17.46
*CONTEXT* (history list property) II: 13.33
Context switching I: 11.4
CONTINUE SAVING? (Printed by System) II: 13.41
CONTINUE WITH T CLAUSE (printed by DWIM) II: 20.7
Continuing an edit session II: 16.50
(CONTROL MODE TTBL) III: 30.10; 25.3,5
Control chain (on stack) I: 11.3
Control-A III: 30.5; 25.41; 26.23
Control-B (Interrupt Character) II: 14.27,29; 23.15; III: 30.2
Control-character echoing III: 30.6
Control-D (Interrupt Character) II: 14.2,17,20; 16.49; 18.4; 23.14; III: 30.2; 30.11
CONTROL-E (Error Message) II: 14.31
Control-E (Interrupt Character) II: 13.18; 14.2,20,31; 15.7; 20.5,7; 23.14; III: 30.2; 24.40; 30.11
Control-F III: 26.23
Control-G (in history list) II: 13.19; 13.13
Control-G (Interrupt Character) II: 23.14; III: 30.2; 30.11
Control-L III: 25.26
Control-P (interrupt character) II: 14.10; III: 30.2; 30.11
Control-Q III: 30.5; 25.2,41; 26.23
Control-R III: 30.6; 26.23
Control-T (Interrupt Character) III: 30.2
Control-V III: 30.6; 25.3
Control-W III: 30.6; 25.2; 26.23
Control-X III: 26.24
Control-X (Editor Command) II: 16.18
Control-Y II: 16.75; III: 25.42; 26.23
Control-Z (Editor Command) II: 16.18
CONVERT.FILE.TO.TYPE.FOR.PRINTER (Function) III: 29.2
Coordinate Systems III: 28.23
COPY (DECLARE: Option) II: 17.41
Copy (DEdit Command) II: 16.9
(COPY X) I: 3.8
(COPYALL X) I: 3.8
(COPYARRAY ARRAY) I: 5.2
COPYBUTTONEVENTFN (Window Property) III: 27.41
(COPYBUTTON EVENTFN IMAGEOBJ WINDOWSTREAM) (IMAGEFNS Method) III: 27.38
(COPYBYTES SRCFIL DSTFIL START END) III: 25.20
(COPYCHARS SRCFIL DSTFIL START END) III: 25.20
(COPYDEF OLD NEW TYPE SOURCE OPTIONS) II: 17.27
(COPYFILE FROMFILE TOFILE) III: 24.31
(COPYFN IMAGEOBJ SOURCEHOSTSTREAM TARGETHOSTSTREAM) (IMAGEFNS Method) III: 27.38
COPYING (in CREATE form) I: 8.4
Copying files III: 24.31
Copying image objects between windows III: 27.41
Copying lists I: 3.8; 3.5,13-14,19
(COPYINSERT IMAGEOBJ) III: 27.42
COPYINSERTFN (Window Property) III: 27.42
(COPYREADTABLE RDTBL) III: 25.35
COPYRIGHTFLG (Variable) I: 12.3; II: 17.53
COPYRIGHTOWNERS (Variable) I: 12.3; II: 17.54
(COPYTERMTABLE TTBL) III: 30.5
COPYWHEN (DECLARE: Option) II: 17.42
CORE (file device) III: 24.29
(COREDEVICE NAME NODIRFLG) III: 24.30
(COROUTINE CALLPTR COROUTPTR COROUTFORM ENDFORM) I: 11.19
Coroutines I: 11.18
(COS X RADIANSFLG) I: 7.13
(COUNT X) I: 3.10
COUNT FORM (I.S. Operator) I: 9.11
(COUNTDOWN X N) I: 3.11
Courier III: 31.15
Courier programs III: 31.15
INDEX

INDEX

Destination bitmap III: 27.23
DESTINATION IS INSIDE EXPRESSION BEING MOVED (Printed by Editor) II: 16.38
Destructive functions I: 3.13,19; II: 22.14
Destructuring argument lists I: 10.27
(DETACHALLWINDOWS MAINWINDOW) III: 28.47
(DETACHWINDOW WINDOWTODETACH) III: 28.47
Determiners in Masterscope II: 19.13
DEVICE (File name field) III: 24.5
DEVICE (Font property) III: 27.27
Device-independent graphics III: 27.42
DEVICESPEC (Font property) III: 27.28
(DF FN NEW?) II: 16.2
DFNFLG (Variable) I: 10.10; II: 13.29; 16.69; 17.5,28
(DIFFERENCE ...) I: 7.3
different expression (Printed by Editor) II: 16.66
DIG (Device-Independent Graphics) III: 27.42
(DIR FILEGROUP COM ... COMn) III: 24.35
DIRCOMMANDS (Variable) III: 24.35
Directories III: 24.31
DIRECTORIES (Variable) I: 12.3; II: 17.16; III: 24.31; 24.32
DIRECTORY (File name field) III: 24.6
DIRECTORY FILES COMMANDS DEFAULTTEXT DEFAULTVERS) III: 24.33
(DIRECTORYNAME DIRNAME STRPTR ...) III: 24.11
(DIRECTORYNAMEP DIRNAME HOSTNAME) III: 24.11
Disabling CLISP operators II: 21.26
(DISCARDPUPS SOC) III: 31.30
(DISCARDXIPS NSOC) III: 31.38
(DISKFREEPAGES VOLUMENAME ...) III: 24.23; 24.21
(DISKPARTITION) III: 24.23; 24.21
(DISMISS MSECSTW W TIMER NOBLOCK) II: 23.5
DISPLAY (Image stream type) III: 27.23; 27.8
Display screens I: 12.4; III: 30.22
Display streams III: 27.23; 27.8
(DISPLAYDOWN FORM NSCANLINES) III: 30.24
(DISPLAYFN IMAGEOBJ IMAGESTREAM
IMAGESTREAMTYPE HOSTSTREAM)
(IMAGEFNS Method) III: 27.37
DISPLAYFONDIRECTORIES (Variable) I: 12.3; III: 27.31
DISPLAYFONTEXTENSIONS (Variable) I: 12.3; III: 27.31
DISPLAYHELP (Function) III: 26.30
DISPLAYTYPES (Variable) III: 26.39
Division by zero I: 7.2
DMACRO (Property Name) I: 10.21
(DMPHSH HARRAY1 HARRAY2 ... HARRAYn) I: 6.3
DO COM (Editor Command) II: 16.54; 13.43
DO FORM (I.S. Operator) I: 9.10
(DOBACKGROUNDCOM) III: 28.7
(DOCOLLECT ITEM LST) I: 3.7
DOCOPY (DECLARE: Option) II: 17.41
Document printing III: 29.1
DOEVALLCOMPIL (DECLARE: Option) II: 17.42
DOEVALLLOAD (DECLARE: Option) II: 17.41
DON'T.CHANGE.DATE (OPENSTREAM parameter) III: 24.3
DONTCOMPILFS (Variable) II: 18.14; 18.15,18
DONTCOPY (DECLARE: Option) II: 17.41
DOTEVALCOMPIL (DECLARE: Option) II: 17.42
DOTEVALLOAD (DECLARE: Option) II: 17.41
(DOSELECTEDITEM MENU ITEM BUTTON) III: 28.43
DOSHAPFN (Window Property) III: 28.18
DOVER (Printer type) III: 29.5
(DOWINDOWCOM WINDOW) III: 28.7
DOWINDOWCOMFN (Window Property) III: 28.7
(DP NAME PROP) II: 16.2
(DPB N BYTESPEC VAL) (Macro) I: 7.10
(DRAWBETWEEN POSITION1 POSITION2 STREAM COLOR DASHING) III:
27.17
(DRAWCIRCLE CENTERX CENTERY RADIUS BRUSH
DASHING STREAM) III: 27.19
(DRAWCURVE KNOTS CLOSED BRUSH DASHING STREAM) III: 27.19
(DRAWELLIPSE CENTERX CENTERY
SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION
BRUSH DASHING STREAM) III: 27.19
(DRAWLINE X1 Y1 X2 Y2 WIDTH OPERATION STREAM COLOR DASHING) III:
27.17
(DRAWPOINT X Y BRUSH STREAM OPERATION) III:
27.20
(DRAWTO X Y WIDTH OPERATION STREAM COLOR
DASHING) III: 27.17
(DREMOVE X L) I: 3.19
(DREVERSE L) I: 3.19
(DRIBBLE FILE APPENDFLG THAWEDFLG) III: 30.12
Dribble files III: 30.12
(DRIBBLEFILE) III: 30.13
DSK (file device) III: 24.21
(DSKDISPLAY NEWSSTATE) III: 24.23
DSKDISPLAY.POSITION (Variable) III: 24.23
INDEX

DSP (Window Property) III: 28.34
(DSPBACKCOLOR COLOR STREAM) III: 27.13
(DSPBACKUP WIDTH DISPLAYSTREAM) III: 27.25
(DSPBOTTOMMARGIN YPOSITION STREAM) III:
27.11
(DSPCLIPPINGREGION REGION STREAM) III: 27.11
(DSPCOLOR COLOR STREAM) III: 27.13
(DSPCREATE DESTINATION) III: 27.23
(DSPDESTINATION DESTINATION DISPLAYSTREAM)
III: 27.23
(DSPFILL REGION TEXTURE OPERATION STREAM) III:
27.20
(DSPFONT FONT STREAM) III: 27.11
(DSPLEFTMARGIN XPOSITION STREAM) III: 27.11
(DSPLINEFEED DELTAY STREAM) III: 27.12
(DSPNEWPAGE STREAM) III: 27.21
(DSPOPERATION OPERATION STREAM) III: 27.12
(DSPRESET STREAM) III: 27.21
(DSPRIGHTMARGIN XPOSITION STREAM) III: 27.11
(DSPSCALE SCALE STREAM) III: 27.12
(DSPSCROLL SWITCHSETTING DISPLAYSTREAM)
III: 27.24
(DPSOURCETYPE SOURCETYPE DISPLAYSTREAM)
III: 27.24
(DSPSPACEFACTOR FACTOR STREAM) III: 27.12
(DSTEXTURE TEXTURE DISPLAYSTREAM) III:
27.24
(DSPTOPMARGIN YPOSITION STREAM) III: 27.11
(DSPXOFFSET XOFFSET DISPLAYSTREAM) III: 27.23
(DSPXPOSITION XPOSITION STREAM) III: 27.13
(DSPYOFFSET YOFFSET DISPLAYSTREAM) III: 27.23
(DSPYPOSITION YPOSITION STREAM) III: 27.13
(DSUBLIS ALST EXPR FLG) I: 3.14
(DSUBST NEW OLD EXPR) I: 3.13
DT.EDITMACROS (Variable) II: 16.12
DUMMY-EDIT-FUNCTION-BODY (Variable) II:
16.70; 16.2
(DUMMYFRAMEP POS) I: 11.13
(DUMPDATABASE FNLS) II: 19.24
(DUNPACK X SCRATCHLIST FLG RDTBL) I: 2.9
Duration Functions I: 12.16
during INTERVAL (I.S. Operator) I: 12.18
(DV VAR) II: 16.2
DW (Editor Command) II: 16.55; 21.27
DWIM II: 20.1
(DWIM X) II: 20.4
DWIM interaction with user II: 20.4
DWIM variables II: 20.12
DWIMCHECK#ARGSFLG (Variable) II: 21.22
DWIMCHECKPROGLABELSFLG (Variable) II: 21.22;
21.19
DWIMESSGAG (Variable) II: 21.22; 18.12
DWIMFLG (Variable) II: 20.14; 16.66,68,71; 20.23
(DWIMIFY X QUIETF FlG L) II: 21.18; 21.20; 21.15
DWIMIFYCOMPFLG (Variable) II: 21.22;
18.12,15,21
DWIMIFYFLG (Variable) II: 20.13
(DWIMIFYFNS FN1 ... FNn) II: 21.20; 21.19
DWIMINMACROSFLG (Variable) II: 21.20
DWIMLOADFNS? (Function) II: 20.13
DWIMLOADFNSFLG (Variable) II: 20.14; 20.13
DWIMUSERFORMS (Variable) II: 20.11; 20.9-10
DWIMWAIT (Variable) II: 20.13; 20.5-6
E
(E X T) (Editor Command) II: 16.58
(E X) (Editor Command) II: 16.58
E (Editor Command) II: 16.57; 13.43; 16.55
(E FORM1 ... FORMn) (File Package Command) II:
17.40
E (in a floating point number) I: 7.11; III: 25.3
E (use in comments) III: 26.43
EACHTIME FORM (I.S. Operator) I: 9.16; 9.18
(ECHOCHAR CHARCODE MODE TTBL) III: 30.6
(ECHOCONTROL CHAR MODE TTBL) III: 30.7
Echoing characters III: 30.6
(ECHOMODE FLG TTBL) III: 30.7
ED (Editor Command) III: 26.29
RELATIONED BY SET (Masterscope Set Specification) II:
19.12
RELATIONED IN SET (Masterscope Set Specification)
II: 19.12
EDIT (Break Command) II: 14.11; 14.12-13
EDIT (Break Window Command) II: 14.3
Edit (EDIT) (Command) II: 16.9
(EDIT NAME —) II: 16.68
EDIT (Litatom) II: 16.50
EDIT SET [- EDITCOMS] (Masterscope Command) II:
19.6
edit (Printed by Editor) II: 16.72
Edit chain II: 16.13; 16.20
Edit macros II: 16.62
EDIT WHERE SET RELATION SET [- EDITCOMS]
(Masterscope Command) II: 19.6
EDIT-SAVE (Property Name) II: 16.49-50
(EDIT4E PAT X —) II: 16.72
(EDITBM BMSPEC) III: 27.4
(EDITCALLERS ATOMS FILES COMS) II: 16.74
ERRORTYPEST  (Variable)  II: 14.22; III: 24.3
(ERRORRX ERXM)  II: 14.19
ERRORX (Litatom)  II: 14.16
(ERSETQ FORM)  I: 9.9; II: 14.22
ESC (type of read macro)  III: 25.40
(ESCAPE FLG RDTBL)  III: 25.39
ESCAPE (Syntax Class)  III: 25.35
Escape ($) (in CLISP)  II: 21.10-11
Escape ($) (in Edit Pattern)  II: 16.18
Escape ($) (in Editor)  II: 16.45-46
Escape ($) (in spelling correction)  II: 20.15; 20.22
Escape ($) (in TTYIN)  III: 26.23
Escape ($) (Prog. Asst. Command)  II: 13.11
Escape ($) (use in ASKUSER)  III: 26.19
Escape-GO ($GO)  (TYPE-AHEAD command)  II: 13.18
Escape-Q ($Q)  (TYPE-AHEAD command)  II: 13.18
Escape-STOP ($STOP)  (TYPE-AHEAD command)  II: 13.18
ESQUOTE (type of read macro)  III: 25.40
(ESUBST NEW OLD EXPR ERRORFLG CHARFLG)  II: 16.73; 13.9
(ETHERHOSTNAME PORT USE.OCTAL.DEFAULT)  III: 31.6
(ETHERHOSTNUMBER NAME)  III: 31.6
Ethernet  III: 31.1
ETHERPACKET (data type)  III: 31.26
(ETHERPORT NAME ERRORFLG MULTFLG)  III: 31.6
ETHERTIMEOUT (Variable)  III: 31.38
EV (Editor Command)  II: 16.52
EV (Function)  II: 16.4
EVAL (Break Command)  II: 14.5; 14.6; 15.6
EVAL (Break Window Command)  II: 14.3
Eval (DEdit Command)  II: 16.9
Eval (Editor Command)  II: 16.58
(EVAL.X —)  I: 10.12
EVAL (in Masterscope template)  II: 19.19
EVAL (Litatom)  II: 21.21
EVAL-format input  II: 13.4
(EVALAS.PROCESS FORM)  II: 23.17
(EVALINTTYP.PROCESS FORM WAITFORRESULT)  II: 23.18
EVAL@COMPILe (DECLARE: Option)  II: 17.42
EVAL@COMPILeWHEN (DECLARE: Option)  II: 17.42
EVAL@LOAD (DECLARE: Option)  II: 17.41
EVAL@LOADWHEN (DECLARE: Option)  II: 17.41
(EVALA X A)  I: 10.13
(EVALHOOK FORM EVALHOOKFN)  I: 10.14
Evaluating arguments to functions  I: 10.2; 10.12
Evaluating data types  I: 10.2
Evaluating expressions  I: 10.13
Evaluating functions  I: 10.11
Evaluating nlambda arguments  I: 10.5
(EVALV VAR POS RELFLG)  I: 11.8
EVALV-format input  II: 13.4
(EVENP X Y)  I: 7.9
EVENT (Variable)  II: 13.22
Event addresses  II: 13.6
Event numbers  II: 13.31; 13.6,13,22,40
Event specifications  II: 13.5; 13.21
(EVERY EVERYX EVERYFN1 EVERYFN2)  II: 10.17
(EXAM X)  (Editor Command)  II: 16.61
(EXCHANGEUPS SOC OUTPUP DUMMY IDFILTER TIMEOUT)  III: 31.30
(EXCHANGEXIPS SOC OUTXIP IDFILTER TIMEOUT)  III: 31.38
Executive  II: 13.1
Executive window  III: 28.3
Exit (DEdit Command)  II: 16.10
EXP (Variable)  II: 15.4
Expand (Window Menu Command)  III: 28.5
(EXPANDBITMAP BITMAP WIDTHFACTOR HEIGHTFACTOR)  III: 27.4
EXPANDFN (Window Property)  III: 28.23
EXPANDINGBOX (Variable)  III: 30.15
(EXPANDMACRO EXP QUIETFLG —)  I: 10.24
(EXPANDOW ICONW)  III: 28.22
EXPANSION (Font property)  III: 27.27
EXPLAINDELIMITER (ASKUSER option)  III: 26.17
EXPLAINSTRING (ASKUSER option)  III: 26.16
(EXPORT COM1 ... COMN)  (File Package Command)  II: 17.43
EXPR (Litatom)  I: 10.7
EXPR (Property Name)  I: 10.10; II: 16.69-70; 17.5,18,27; 18.13; 20.9-10
EXPR (Variable)  II: 20.13; 19.21
Expr definitions  I: 10.2; 10.1
EXPR* (Litatom)  I: 10.7
EXPRESSIONS (File Package Type)  II: 17.23; 13.17
(EXPRP FN)  I: 10.7
(EXPT A N)  I: 7.13
(EXTENDREGION REGION INCLUDEREGION)  III: 27.2
EXTENSION (File name field)  III: 24.6
EXTENT (Window Property)  III: 28.26; 28.23-25,34
Extents  III: 28.23
(EXTRACT @1 FROM . @2) (Editor Command) II: 16.36
FILE (GETFN Property) III: 27.40
FILE (Property Name) II: 17.19
File access rights III: 24.2
File attributes III: 24.17
File devices III: 24.1
File directories III: 24.31
File enumeration III: 24.33
File maps II: 17.55
File names II: 22.13; III: 24.5; 24.1,9,12-13
FILE NOT FOUND (Error Message) II: 14.29; III: 24.3,31
FILE NOT OPEN (Error Message) II: 14.28; III: 24.4,14; 25.2,6,20
File package II: 17.1
File package commands II: 17.32
File package types II: 17.21
File pointers III: 25.18; 25.19,23
File servers III: 24.36
FILE SYSTEM RESOURCES EXCEEDED (Error Message) II: 14.29; III: 24.3,13
FILE WON'T OPEN (Error Message) II: 14.28; III: 24.3
FILE: (Compiler Question) III: 18.1
FILECHANGES FILE TYPE II: 17.52
FILECHANGES (Property Name) II: 17.20; 17.15
Filecoms II: 17.32; 17.4-5,48
FILECOMS FILE TYPE II: 17.49
FILECOMSLST FILE TYPE — II: 17.49
FILECREATED X II: 17.51; 18.13
FILEDATE (Property Name) II: 17.52
FILEDATE (Record Operator) II: 17.56
FILEDATES (Property Name) II: 17.20; 17.15,51
FILEDEF (Property Name) II: 20.10
FILEFN (Variable) II: 20.12
FILEGETDEF (File Package Type Property) II: 17.30
FILEGROUP (Property Name) II: 17.12
FILELINELENGTH (Variable) III: 25.11; 26.48
FILELST (Variable) II: 17.20; 17.6,12; 20.24
FILEMAP (Property Name) II: 17.20; 17.55
FILEMAP DOES NOT AGREE WITH CONTENTS OF (Error Message) II: 17.56
FILEMAPLN FILENAME FILENAME) III: 24.8
FILEOUTCHARN (Function) III: 27.48
FILEPKG.SCRATCH (file) II: 17.30
FILEPKGCHANGES TYPE LST II: 17.18
FILEPKGCOM COMMANDNAME PROP VAL ... PROP N VAL N II: 17.47
FILEPKGCOS LITATOM1 ... LITATOM N (File Package Command) II: 17.39
FILEPKGCOS (File Package Type) II: 17.23

INDEX
FILEPKGCOMPLST (Variable) II: 17.34
FILEPKGFLG (Variable) II: 17.5
(FILEPKGTYPE TYPE PROP1 VAL1 ... PROPNFLG II: 17.32
FILEPKGTYPES (Variable) II: 17.22
(FILEPOS PATTERN FILE START END SKIP TAIL CASEARRAY) III: 25.20; 25.21
FILERDTBL (Variable) II: 17.5-6, 50; III: 25.34; 25.7, 33; 26.44
Files III: 24.1
(FILES FILE1 ... FILEN) (File Package Command) II: 17.39
FILES (File Package Type) II: 17.23
.FILES? II: 17.12
(FILESLOAD FILE1 ... FILEN) II: 17.9
FILETYPE (Property Name) II: 18.12, 15; 21.26
Filevars II: 17.44; 17.5.49
FILEVARS (File Package Type) II: 17.23
FILINGENUMERATIONDEPHT (Variable) III: 24.38
FILINGTYPES (Variable) III: 24.18
(FILLCIRCLE CENTERX CENmERTY RADIUS TEXTURE STREAM) III: 27.21
(FILLPOLYGON POINTS TEXTURE STREAM) III: 27.20
FINALLY FORM (I.S. Operator) I: 9.16; 9.18
Find (DEdit Command) II: 16.8
FIND (I.S. Operator) I: 9.22
(FINDPROCESS PROCERRORFLG) II: 23.5
(FINDCALLERSATOMSFILES) II: 16.75
(FINDFILE FILENSFLGDIRLIST) III: 24.32
FIRST (as argument to ADVISE) II: 15.11
FIRST (DECLARE: Option) II: 17.42
FIRSTFORM (I.S. Operator) I: 9.16; 9.18
FIRST (type of read macro) III: 25.40
FIRSTCOL (Variable) I: 12.3; III: 26.47; 26.48
FIRSTNAME (Variable) I: 12.2
(FIX N) I: 7.7
FIX EventsSpec (Prog. Asst. Command) II: 13.12; 13.33
FIX format (in PRINTNUM) III: 25.15
FIXEDITDATE (Function) II: 16.76
FIXP (as a field specification) I: 8.21
(FIXP X) I: 7.2; 9.1
FIXP (record field type) I: 8.10
(FIXR N) I: 7.7
(FIXSPELLXWORDREL SPLITFLG TAIL FN TIEFLG DONTMOVETOPFLG — — ) II: 20.22; 20.24
FIXSPELLUPPERCASEQU memorandum II: 20.22
FIXSPELLDEFAULT (Variable) II: 20.13; 20.5; 21.19
FIXSPELLREL (Variable) II: 20.22
FLAG (record field type) I: 8.10
Flash bars on cursor III: 30.16
(FLASHWINDOW WIN? N FLASHINTERVAL SHADE) III: 28.32
(FLAST X) I: 3.9; II: 21.13
(FLCRELENGTH X) I: 3.10
(FLESSP X Y) I: 7.12
(FLIPCURSOR) III: 30.14
(FLOAT X) I: 7.13
FLOAT format (in PRINTNUM) III: 25.15
FLOATING (record field type) I: 8.10
FLOATINGOVERFLOW (Error Message) II: 14.31
Floating point arithmetic I: 7.11
Floating point numbers I: 7.11; 7.1-2; 9.1; III: 25.3
Floating point overflow I: 7.2
FLOATINGUNDERFLOW (Error Message) II: 14.31
FLOATP (as a field specification) I: 8.21
(FLOATP X) I: 7.2, 9.1
FLOATP (record field type) I: 8.10
FLOPPY (file device) III: 24.24
Floppy disk drive III: 24.24
Floppy disk modes III: 24.24
Floppy image file III: 24.27
(FLOPPYARCHIVE FILESNAMN) III: 24.28
(FLOPPYCANREADP) III: 24.27
(FLOPPYCANWRITEP) III: 24.27
(FLOPPYFORMAT NAME AUTOCONFIRMFLG SLOWFLG) III: 24.26
(FLOPPYFREEPAGES) III: 24.27
(FLOPPYFROMFILEFROMFILE) III: 24.28
(FLOPPYMODEMODE) III: 24.24
(FLOPPYNAME NAME) III: 24.27
(FLOPPYSCAVENGE) III: 24.27
(FLOPPYTOFILETOFILE) III: 24.27
(FLOPPYUNARCHIVEHOSTDIRECTORY) III: 24.28
(FLOPPYWAITFORFLOPPYNEWFLG) III: 24.27
(FLTFMTFORMAT) III: 25.13
(FLUSHRIGHTPOS X MIN P2FLAG CENTERFLAGFILE) III: 25.32
(FMAXX1X2 ... XN) I: 7.13
(FMEMBXY) I: 3.13; II: 21.13
(FMINX1X2 ... XN) I: 7.12
(FMINUS X) I: 7.12
*FN* (stackblip) I: 11.16
FN (Variable) II: 19.7
(FNCHECK FNOERRORFLG SPELLFLGPROPFLG TAIL) I: 10.8; II: 20.23
(GATHEREXPORTS FROMFILES TOFILE FLG) II: 17.43
(GCD N1 N2) I: 7.7
(GCGAG MESSAGE) II: 22.3
(GCTRIP) II: 22.3
(GDATE DATE FORMAT —) II: 12.14
GE (CLISP Operator) II: 21.8
(GENERATE HANDLE VAL) I: 11.17
(GENERATOR FORM COMVAR) I: 11.17
Generator handles I: 11.17
Generators I: 11.16
Generators for spelling correction II: 20.19
Generic arithmetic I: 7.3
GENNUM (Variable) I: 2.11
(GENSYM PREFIX — — —) I: 2.10; II: 15.10-11
(GEQ X Y) I: 7.4
GET (old name for LISTGET1) I: 3.16
GET* (Editor Command) II: 16.55; III: 26.44
(GETATOMVAL VAR) I: 2.4
(GETBOXPOSITION BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG) III: 28.9
(GETBOXREGION WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG) III: 28.11
(GETBRK RDTBL) III: 25.38
(GETCASEARRAY CASEARRAY FROMCODE) III: 25.22
(GETCHARBITMAP CHRACTR CODE FONT) III: 27.30
(GETCOMMENT X DESTFL —) III: 26.44
(GETCONTROL TTBL) III: 30.10
GETD (Editor Command) II: 16.56
(GETD FN) I: 10.10
GETDEFINE (File Package Type Property) II: 17.30
(GETDEFINE NAME TYPE SOURCE OPTIONS) II: 17.25
(GETDELETECONTROL TYPE TTBL) III: 30.9
(GETDESCRIPTORS TYPENAME) I: 8.22
GETDUMMYVTR (Function) I: 9.20
(GETECHOMODE TTBL) III: 30.7
(GETEOF PTR FILE) III: 25.20
(GETFIELDSPECS TYPENAME) I: 8.22
(GETFILEINFO FILE ATTRIB) III: 24.17
(GETFILEPTR FILE) III: 25.19
(GETFN FILESTREAM) (IMAGEFNS Method) III: 27.37
(GETHASH KEY HARRAY) I: 6.2; II: 21.17
(GETLIS X PROPS) I: 2.7
(GETMENUPROP MENU PROPERTY) III: 28.43
(GETMOUSESTATE) III: 30.19
GETP (old name of GETPROP) I: 2.5
(GETPOSITION WINDOW CURSOR) III: 28.9
(GETPROMPTWINDOW MAINWINDOW #LINES FONT DONTCREATE) III: 28.50
(GETPROP ATM PROP) I: 2.5
(GETPROP LIST ATM) I: 2.7
(GETPUP PUPSOC WAIT) III: 31.30
(GETPUPBYTE PUP BYTE #) III: 31.31
(GETPUPSTRING PUP OFFSET) III: 31.32
(GETPUPWORD PUP WORD #) III: 31.31
(GETRAISE TTBL) III: 30.8
(GETREADTABLE RDTBL) III: 25.34
(GETREGION MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS) III: 28.10
(GETRELATION ITEM RELATION INVERTED) II: 19.23
(GETRESOURCE RESOURCENAME . ARGS) (Macro) I: 12.23
(GETSEPR RDTBL) III: 25.38
(GETSTREAM FILE ACCESS) III: 25.2
(GETSYNTAX CH TABLE) III: 25.36
(GETTEMPLATE FN) II: 19.21
(GETTERMTABLE TTBL) III: 30.5
(GETTPOPVAR VAR) I: 2.4
GETVAL (Editor Command) II: 16.58
(GETXIP NSOC WAIT) III: 31.37
(GIVE.TTY.PROCESS WINDOW) II: 23.13
(GLC X) I: 4.3
Global variables II: 18.4; 21.19; 22.5
GLOBALVAR (Property Name) II: 18.4; 21.19
Globalvars II: 18.4
(GLOBALVARS VAR1 ... VARn) (File Package Command) II: 17.37; 18.4
GLOBALVARS (in Masterscope Set Specification) II: 19.12
GLOBALVARS (Variable) II: 18.4; 18.18; 21.19
(GNC X) I: 4.3
GO (Break Command) II: 14.5; 14.6
(GO LABEL) (Editor Command) II: 16.23
(GO U) I: 9.8
GO (in iterative statement) I: 9.18
$GO (escape-GO) (TYPE-AHEAD command) II: 13.18
GRAYSHADE (Variable) III: 27.7
(GREATERNUMBER —) I: 12.2
GREETDATES (Variable) I: 12.2
(GREETFILENAME USER) I: 12.2
Greeting I: 12.1
GRID Grid specification  III: 27.22
Grds  III: 27.22
(GridXCOORD XCOORD Gridspec)  III: 27.22
(GridYCOORD YCOORD Gridspec)  III: 27.22
*GROUP* (history list property)  II: 13.33
GT (CLISP Operator)  II: 21.8

H
Hard disk device  III: 24.21
HARD DISK ERROR (Error Message)  II: 14.28;  III: 24.24
Hardcopy (Background Menu Command)  III: 28.6
Hardcopy (Window Menu Command)  III: 28.4
Hardcopy facilities  III: 29.1
HARDCOPYFN (Window Property)  III: 28.34
(HARDCOPYW WINDOW/BITMAP/REGION FILE
   HOST SCALEFACTOR ROTATION PRINTERTYPE)
   III: 29.3
(HARDFRESET)  II: 23.1; 14.26
(HARRAY MINKEYS)  l: 6.2
(HARRAYP X)  l: 6.2;  9.2
(HARRAYPROP HARRAY PROP NEWVALUE)  l: 6.2
(HARRAYSIZE HARRAY)  l: 6.2
HASDEF (File Package Type Property)  II: 17.30
(HASDEF NAME TYPE SOURCE SPELLFLG)  II: 17.26
HASH ARRAY FULL (Error Message)  l: 6.3
Hash arrays  l: 6.1
Hash keys  l: 6.1
Hash overflow  l: 6.3
HASH TABLE FULL (Error Message)  l: 6.3;  II: 14.29
Hash values  l: 6.1
(HASHHARRAY MINKEYS OVERFLOW HASHBITSFN
   EQUIVFN)  l: 6.1
Hashing functions  l: 6.4
HASHLINK (Record Type)  l: 8.9
HASHOVERFLOW (Function)  l: 6.3
(HASTTYWINDOWP PROCESS)  II: 23.11
(HCOPYALL X)  l: 3.8;  III: 25.18
HEIGHT (Font property)  III: 27.28
HEIGHT (Window Property)  III: 28.34
(HEIGHTWINDOW INTERIORHEIGHT TITLEFLG
   BORDER)  III: 28.32
HELP MESS1 MESS2 BRKTYPE  II: 14.20
HELP (Interrupt Channel)  II: 23.14;  III: 30.3
Help! (Error Message)  II: 14.20
HELPCLK (Variable)  II: 14.14; 13.9,35
HELPDEPTH (Variable)  II: 14.13
HELPFLAG (Variable)  II: 14.14; 14.27
HELPTIME (Variable)  II: 14.14
HERALDSTRING (Variable)  l: 12.9
HERE (in edit command)  II: 16.34
*HISTORY* (history list property)  II: 13.33
HISTORY (Property Name)  II: 13.14
HISTORY (Variable)  II: 13.22
History list format  II: 13.31
History lists  II: 13.1; 13.31; 16.54
HISTORYCOMS (Variable)  II: 13.43
(HISTORYFIND LST INDEX MOD EVENTADDRESS —)
   II: 13.40; 13.39
(HISTORYMATCH INPUT PAT EVENT)  II: 13.40
(HISTORYSAVE HISTORY ID INPUT1 INPUT2 INPUT3
   PROPS)  II: 13.38; 13.31,33-34,43
HISTORYSAVEFORMS (Variable)  II: 13.22
HISTSTR0 (Variable)  II: 13.32
HISTSTR1 (Variable)  III: 26.32
HorizScrollCursor (Variable)  III: 30.16
HorizThumbCursor (Variable)  III: 30.16
(HORRIBLEVARS VAR1 ... VARN) (File Package
   Command)  II: 17.36;  III: 25.18
HOST (File name field)  III: 24.5
(HOSTNAMEP NAME)  III: 24.11
Hot spot of cursor  III: 30.14
Hotspot  III: 30.14
(HPRINT EXPR FILE UNCIRCULAR DATATYPESEEN)
   III: 25.17
HPRINT.SCRATCH (File name)  III: 25.17
(HREAD FILE)  III: 25.18
I
(l C X1 ... XN) (Editor Command)  II: 16.58
.IFORMAT NUMBER (PRINTOUT command)  III: 25.30
(I.S.OPR NAME FORM OTHERS EVALFLG)  l: 9.20
I.S.OPR (Property Name)  II: 17.18
I.s.oprs  l: 9.9
(I.S.OPRS OPR1 ... OPRN) (File Package Command)
   l: 9.22;  II: 17.39
I.S.OPRS (File Package Type)  II: 17.23
I.s.types  l: 9.10; 9.20
ICON (Window Property)  III: 28.22
ICONFN (Window Property)  III: 28.22
Icons  III: 28.21; 28.5
ICONWINDOW (Window Property)  III: 28.23
IconWindowMenu (Variable)  III: 28.8
IconWindowMenuCommands (Variable)  III: 28.8
ICREATIONDATÈ (File Attribute)  III: 24.18
Id (Variable) II: 13.22
(IDATE STR) I: 12.13
(IDDIFFERENCE X Y) I: 7.6
Idle (Background Menu Command) III: 28.6
IDLE (Function) I: 12.4
idle mode I: 12.4
(IDLE.BOUNCING.BOX WINDOW BOX WAIT) I: 12.6
IDLE.BOUNCING.BOX (Variable) I: 12.6
IDLE.FUNCTIONS (Variable) I: 12.6
IDLE.PROFILE (Variable) I: 12.4
Idling I: 12.4
(IF X COMS_1 COMS_2) (Editor Command) II: 16.60
(IF X COMS_1) (Editor Command) II: 16.60
(IF X) (Editor Command) II: 16.60
(IF EXPRESSION TEMPLATE_1 TEMPLATE_2) (in Masterscope template) II: 19.21
IF (Statement) I: 9.5
IF-THEN-ELSE statements I: 9.5
(IFPROP PROPNAMELITATOM ... LITATOMN) (File Package Command) II: 17.38; 17.45
IFY (Editor Command) II: 16.55
(IMAGEFNSCREATE DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN BUTTONEVENTINFN COPYBUTTONEVENTINFN WHENMOVEDFN WHENINSERTEDFN WHENDELETEDFN WHENCOPIEDFN WHENOPERATEDONFN PREPRINTFN — ) III: 27.36
IMAGEFNS (Data Type) III: 27.35
(IMAGEFNSCREATE OBJECTDATUM IMAGEFNS) III: 27.36
IMAGEOBJGETFNS (Variable) III: 27.40
(IMAGEOBJP X) III: 27.36
(IMAGEOBJPIMAGEOBJECTPROPERTY NEWVALUE) III: 27.36
IMAGEOPS (Data type) III: 27.43
IMAGEOPS (Stream Field) III: 27.43
(IMAGESTAMP X IMAGETYPE) III: 27.10
(IMAGESTAMPSTREAM) III: 27.10
(IMAGESTAMPSTAMP STREAM TYPE) III: 27.10
(IMAGESTAMPTYPES (Variable) III: 27.42
IMAGETYPE (IMAGEOPS Field) III: 27.44
IMAGEWIDTH (Menu Field) III: 28.42
(IMAX X_1 X_2 ... X_7) I: 7.7
(INBACKCOLOR STREAM COLOR) (Image Stream Method) III: 27.48
(IMBITBLT SOURCEBITMAP SOURCELEFT
SOURCEBOTTOM STREAM DESTINATIONLEFT
DESTINATIONBOTTOM WIDTH HEIGHT
SOURCETYPE OPERATION TEXTURE
CLIPPINGREGION CLIPPEDSOURCELEFT
CLIPPEDSOURCEBOTTOM SCALE) (Image Stream Method) III: 27.45
(IMBITMAPSIZE STREAM BITMAP DIMENSION)
(Image Stream Method) III: 27.46
(IMBLTSHADE TEXTURE STREAM DESTINATIONLEFT
DESTINATIONBOTTOM WIDTH HEIGHT
OPERATION CLIPPINGREGION) (Image Stream Method) III: 27.45
(IMBOTTOMMARGIN STREAM YPOSITION) (Image Stream Method) III: 27.47
(IMCHARWIDTH STREAM CHARCODE) (Image Stream Method) III: 27.46
INDEX

(IMCHARWIDTH STREAM CHARCODE) (Image Stream Method) III: 27.46
(IMCLIPPINGREGION STREAM REGION) (Image Stream Method) III: 27.47
(IMCLOSEFN STREAM) (Image Stream Method) III: 27.44
(IMCOLOR STREAM COLOR) (Image Stream Method) III: 27.48
(IMDRAWCIRCLE STREAM CENTERX CENTERY RADIUS BRUSH DASHING) (Image Stream Method) III: 27.44
(IMDRAWCURVE STREAM KNOTS CLOSED BRUSH DASHING) (Image Stream Method) III: 27.44
(IMDRAWELLIPSE STREAM CENTERX CENTERY SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING) (Image Stream Method) III: 27.45
(IMDRAWLINE STREAM X1 Y1 X2 Y2 WIDTH OPERATION COLOR DASHING) (Image Stream Method) III: 27.44
(IMFILLCIRCLE STREAM CENTERX CENTERY RADIUS TEXTURE) (Image Stream Method) III: 27.45
(IMFILLPOLYGON STREAM POINTS TEXTURE) (Image Stream Method) III: 27.45
(IMFONT STREAM FONT) (Image Stream Method) III: 27.47
(IMFONTCREATE (IMAGEOPS Field) III: 27.44
(IMIN X1 X2 ... XN) I: 7.7
(IMINUS X) I: 7.6
(IMLEFTMARGIN STREAM LEFTMARGIN) (Image Stream Method) III: 27.47
(INLINEFEED STREAM DELTA) (Image Stream Method) III: 27.47
(IMMED (type of read macro) III: 25.41
(IMMEDIATE (type of read macro) III: 25.41
(IMMOVETO STREAM X Y) (Image Stream Method) III: 27.45
(IMNEWPAGE STREAM) (Image Stream Method) III: 27.46
(IMOD X N) I: 7.6
(IMOPERATION STREAM OPERATION) (Image Stream Method) III: 27.48
(IMPORTFILE FILE RETURNFLG) II: 17.43
(IMRESET STREAM) (Image Stream Method) III: 27.46
(IMRIGHTMARGIN STREAM RIGHTMARGIN) (Image Stream Method) III: 27.47
(IMSCALE STREAM SCALE) (Image Stream Method) III: 27.48; 27.44
(IMSCALEDBITBL SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE) (Image Stream Method) III: 27.45
(IMSPACEFACTOR STREAM FACTOR) (Image Stream Method) III: 27.48
(IMSTRINGWIDTH STREAM STR RDTBL) (Image Stream Method) III: 27.46
(IMTERPRI STREAM) (Image Stream Method) III: 27.46
(IMTOPMARGIN STREAM YPOSITION) (Image Stream Method) III: 27.47
(IMXPOSITION STREAM XPOSITION) (Image Stream Method) III: 27.47
(IMYPOSITION STREAM YPOSITION) (Image Stream Method) III: 27.47
(FN1 IN FN2) (arg to BREAKO) II: 15.4
IN FORM (I. S. Operator) I: 9.13; 9.14, 18
IN (in EMBED editor command) II: 16.37
IN (in USE command) II: 13.9
IN EXPRESSION (Masterscope Set Specification) II: 19.11
ON OLD (VAR→FORM) (I. S. Operator) I: 9.13
IN OLD (VAR→FORM) (I. S. Operator) I: 9.13
IN OLD VAR (I. S. Operator) I: 9.13
IN? (Break Command) II: 14.13
Incomplete file names II: 22.13; III: 24.9; 24.14
INCORRECT DEFINING FORM (Error Message) I: 10.9
(INFILE FILE) III: 24.15
(INFILECOMS? NAME TYPE COMS —) II: 17.48
(INFILEP FILE) III: 24.13
INFIX (type of read macro) III: 25.39
Infix operators in CLISP II: 21.7
INFO (Property Name) I: 10.4; II: 21.21; 13.41; 21.18, 23
INFOHOOK (Process Property) II: 23.16; 23.3
RELATIONING SET (Masterscope Set Specification) II: 19.11
INIT (in record declarations) I: 8.14
Init files I: 12.1
INIT.LISP (File name) I: 12.1
INITCORNERSFN (Window Property) III: 28.18
Initialization files I: 12.1
INITIALS (Variable) II: 16.76
INITIALSLST (Variable) I: 12.4; II: 16.76
(INITRECORDS REC1 ... REC_N) (File Package Command) I: 8.11; II: 17.38
(INITRESOURCE RESOURCE_NAME . ARGS) (Macro) I: 12.23
(INITRESOURCES RESOURCE1 ... RESOURCE_N) (File Package Command) I: 12.20,24; II: 17.39
(INITVARS VAR1 ... VAR_N) (File Package Command) II: 17.36
INPUT (File access) III: 24.2
(INPUT FILE) III: 25.3
Input buffer II: 14.16; III: 30.11; 25.6
Input functions III: 25.2
Input/Output functions III: 25.1
(INREADMACROP) III: 25.42
(INSERT E1 ... EM BEFORE . @) (Editor Command) II: 16.33
(INSERT E1 ... EM AFTER . @) (Editor Command) II: 16.33
(INSERT E1 ... EM FOR . @) (Editor Command) II: 16.33
INSIDE FORM (I.S. Operator) I: 9.13
(INSIDE_REGION POSORX Y) III: 27.3
(INPECT OBJECT ASTYPE WHERE) III: 26.2
INSPECT/ARRAY (Function) III: 26.5
INSPECTALLFIELDSFLG (Variable) III: 26.6
(INPECTCODE FN WHERE — — — — ) III: 26.2
INSPECTMACROS (Variable) III: 26.6
Inspector III: 26.1
INSPECTPRINTLEVEL (Variable) III: 26.5
(INPECTW.CREATE DATUM PROPERTIES FETCHFN STOREFN PROPCOMMANDFN VALUECOMMANDFN TITLECOMMANDFN TITLE SELECTIONFN WHERE PROPPRINTFN) III: 26.7
(INPECTW.REDISPLAY INSPECTW PROPS — — ) III: 26.9
(INPECTW.REPLACE INSPECTW PROPERTY NEWVALUE) III: 26.9
(INPECTW.SELECTITEM INSPECTW PROPERTY VALUEFLG) III: 26.9
INSPECTWTITLE (Window Property) III: 26.8
(INSTALLBRUSH BRUSHNAME BRUSHFN BRUSHARRAY) III: 27.19
INSTRUCTIONS (Litatom) I: 10.23
INTEGER (record field type) I: 8.10
Integer arithmetic I: 7.5
Integer input syntax I: 7.4; III: 25.3,9
(INTEGERLENGTH X) I: 7.9
Integers I: 7.4; 9.1
Interlisp-D executive II: 13.1
Interlisp-D executive window III: 28.3
INTERPRESS (Image stream type) III: 27.8
Interpress format I: 12.3; III: 27.8-10,12,31,33; 29.1,5
INTERPRESSFONTDIRECTORIES (Variable) I: 12.3; III: 27.31
Interpreter and the stack I: 11.14
Interpreting expressions I: 10.11
Interpreter blips on the stack I: 11.14
INTERRUPT (Litatom) II: 14.16
Interrupt characters III: 30.1
(INTERRUPTABLE FLAG) III: 30.4
(INTERRUPTCHAR CHAR TYPFORM HARDFLG —) III: 30.3
INTERSECTION X Y I: 3.11
(INTERSECTREGIONS REGION1 REGION2 ... REGION N) III: 27.2
Inverted cursor III: 30.16
(INVERTW WINDOW SHADE) III: 28.31
(IOFILE FILE) III: 24.15
(IPLUS X1 X2 ... X N) I: 7.6
(IQUOTIENT X Y) I: 7.6
IREADDATE (File Attribute) III: 24.18
(I REMAINDER X Y) I: 7.6
SETIS SET (Masterscope Command) II: 19.5
ISTHERE (I.S. Operator) I: 9.22
IT (Variable) II: 13.20
ITALIC (Font face) III: 27.26
ITEMHEIGHT (Menu Field) III: 28.41
ITEMS (Menu Field) III: 28.39
ITEMWIDTH (Menu Field) III: 28.41
Iterative statements I: 9.9
(ITIMES X1 X2 ... X N) I: 7.6
IT—datum (Inspect Window Command) III: 26.4
IT—selection (Inspect Window Command) III: 26.5
IWRIEDATE (File Attribute) III: 24.18
J
JMACRO (Property Name) I: 10.21
JOIN FORM (I.S. Operator) I: 9.11
JOINC (Editor Command) II: 16.53
K
&KEY (DEFMACRO keyword) I: 10.25
Key names III: 30.19
(KEYACTION KEYNAME ACTIONS — ) III: 30.20
Keyboard III: 30.19
(KEYDOWNP KEYNAME) III: 30.19
KEYLST (ASKUSER argument) III: 26.13
KEYLST (ASKUSER option) III: 26.15
Keys on mouse III: 30.17
KEYSTRING (ASKUSER option) III: 26.16
Keyword macro arguments I: 10.24
KNOWN (Masterscope Set Specification) II: 19.12
(KWOTE X) I: 10.13
L (L-CASE X FLG) II: 2.10; II: 16.52
LABELS (Litatom) II: 21.21,23
LAMBDA (Litatom) I: 10.2
LAMBDA (Macro Type) I: 10.22
Lambda functions I: 10.2
Lambda-nospread functions I: 10.5
Lambda-spread functions I: 10.3
LAMBDACONT (Font class) III: 30.11
LAMBDASPLST (Variable) I: 10.8; II: 20.14; 20.9-11
LAMS (Variable) II: 18.9; 18.14
Landscape fonts III: 27.27
LAPFLG (Variable) I: 18.1
Large integers I: 7.1; 7.2; 9.1
LARGEST FORM (I.S. Operator) I: 9.12
LAST (as argument to ADVISE) II: 15.11
(LAST X) I: 3.9
LASTAIL (Variable) II: 16.14; 16.15,21,72
LASTC FILE III: 25.5
LASTKEYBOARD (Variable) III: 30.19
LASTMOUSEBUTTONS (Variable) III: 30.18
(LASTMOUSESTATE BUTTONFORM) (Macro) III: 30.18
(LASTMOUSEX DISPLAYSTREAM) III: 30.18
LASTMOUSEX (Variable) III: 30.18
(LASTMOUSEY DISPLAYSTREAM) III: 30.18
LASTMOUSEY (Variable) III: 30.18
(LASTN L N) I: 3.10
LASTPOS (Variable) II: 14.6; 14.4.7-10,12
LASTVALUE (Property Name) II: 16.50
\LASTVMEMFILEPAGE (Variable) I: 12.11
LASTWORD (Variable) II: 20.18; 20.21-23; 21.10
(LC _ @) (Editor Command) II: 16.24
\LCASELST (Variable) III: 26.46
LCFIL (Variable) II: 18.1-2
(LCL _ @) (Editor Command) II: 16.24
\LCONC PTR X I: 3.6; 3.7
(LDB BYTESPEC VAL) (Macro) I: 7.10
LDFLG (Argument to LOAD) II: 17.5
(LDIFF LST TAIL ADD) I: 3.12
LDIFF: NOT A TAIL (Error Message) I: 3.12
(LDIFFERENCE X Y) I: 3.11
LE (CLISP Operator) II: 21.8
LEFT (key indicator) III: 30.17
Left margin III: 27.11
LEFTBRACKET (Syntax Class) III: 25.35
(LEFTOFGRIDCOORD GRIDX GRIDSPEC) III: 27.23
LEFTPAREN (Syntax Class) III: 25.35
LENGTH (File Attribute) III: 24.17
(LENGTH X) I: 3.10
(LEQ X Y) I: 7.4
(LESSP X Y) I: 7.4
(LET VARLST E1 E2 _ E_N) (Macro) I: 9.9
(LET* VARLST E1 E2 _ E_N) (Macro) I: 9.9
(LI N) (Editor Command) II: 16.41
LIKE ATOM (Masterscope Set Specification) II: 19.11
(LINBUF FLG) III: 30.11; 30.12
LINE (Variable) III: 26.38
Line buffer III: 30.9; 30.11
Line length III: 27.12
Line-buffering III: 30.9; 25.3-6
line-feed (Editor Command) II: 16.18
LINEDELETE (syntax class) III: 30.5,8
(LINELENGTH N FILE) III: 25.11; 27.12
LINELENGTH N (Masterscope Path Option) II: 19.17
(LISP-IMPLEMENTATION-TYPE) I: 12.12
(LISP-IMPLEMENTATION-VERSION) I: 12.12
(LISPDIRECTORYP VOLUMENAME) III: 24.23
LISPFILE (Property Name) II: 21.28
(LISPINTERRUPTS) III: 30.4
(LISPSOURCEFILEP FILE) II: 17.52
LISPUSERSDIRECTORIES (Variable) I: 12.3; II: 17.9; III: 24.32
(LISPX LISPXX LISPXID LISPXXMACROS
LISPXXUSERFN LISPXXFLG) II: 13.35;
13.12,19,32-34,36,43; 16.51,57; 20.4,17,24
LISP Printing Functions II: 13.25
(LISPX X FN VARS) II: 13.41; 13.27
LISPXCOMS (Variable) II: 13.35; 17.39
(LISPXVALID LISPXFORM LISPXID) II: 13.36
(LISPXFIND HISTORY LINE TYPE BACKUP —) II: 13.39; 13.44
LISPXFINDPLST (Variable) II: 13.8
LISPXHIST (Variable) II: 13.33; 13.30,34,42
LISPXHISTORY (Variable) II: 13.31; 13.35,43
LISPXHISTORYMACROS (Variable) II: 13.23
LISPELINE (Variable) II: 13.23
(LISPXMACROS LITATOM_1 ... LITATOM_N) (File Package Command) II: 17.39
<table>
<thead>
<tr>
<th>INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISPXMACROS (File Package Type) II: 17.23</td>
</tr>
<tr>
<td>LISPXMACROS (Variable) II: 13.23; 13.35</td>
</tr>
<tr>
<td>(LISPXPRIN1 X Y Z NODOFGLG) II: 13.25</td>
</tr>
<tr>
<td>(LISPXPRIN2 X Y Z NODOFGLG) II: 13.25</td>
</tr>
<tr>
<td>(LISPXPRINT X Y Z NODOFGLG) II: 13.25; 13.33</td>
</tr>
<tr>
<td><em>LISPXPRINT</em> (history list property) II: 13.33</td>
</tr>
<tr>
<td>(LISPXPRINTDEF EXPR FILE LEFT DEF TAIL NODOFGLG) II: 13.25</td>
</tr>
<tr>
<td>LISPXPRINTFLG (Variable) II: 13.25</td>
</tr>
<tr>
<td>(LISPXREAD FILE RDTBL) II: 13.38; 13.39; 13.43</td>
</tr>
<tr>
<td>LISPXREADFN (Variable) II: 13.36; 13.5; 38; III: 26.28</td>
</tr>
<tr>
<td>(LISPXREADP FLG) II: 13.38; 13.43</td>
</tr>
<tr>
<td>(LISPXSPACES X Y Z NODOFGLG) II: 13.25</td>
</tr>
<tr>
<td>(LISPXSTOREVALUE EVENT VALUE) II: 13.39</td>
</tr>
<tr>
<td>(LISPXTAB X Y Z NODOFGLG) II: 13.25</td>
</tr>
<tr>
<td>(LISPXTERPRI X Y Z NODOFGLG) II: 13.25</td>
</tr>
<tr>
<td>(LISPXUNREAD LST — ) II: 13.38</td>
</tr>
<tr>
<td>LISPXUSERFN (Variable) II: 13.24; 13.35</td>
</tr>
<tr>
<td>LISPXVALUE (Variable) II: 13.24</td>
</tr>
<tr>
<td>(LIST X1 X2 ... XN) I: 3.4</td>
</tr>
<tr>
<td>LIST (MAKEFILE option) II: 17.11</td>
</tr>
<tr>
<td>LIST (Property Name) II: 17.27</td>
</tr>
<tr>
<td>List cells I: 3.1; 9.2</td>
</tr>
<tr>
<td>List structure editor II: 16.1</td>
</tr>
<tr>
<td>(LIST* X1 X2 ... XN) I: 3.4</td>
</tr>
<tr>
<td>(LISTFILES FILE1 FILE2 ... FILEN) II: 17.14; 17.11</td>
</tr>
<tr>
<td>LISTFILES1 (Function) II: 17.14</td>
</tr>
<tr>
<td>LISTFILESTR (Variable) III: 27.34</td>
</tr>
<tr>
<td>(LISTGET LST PROP) II: 3.16</td>
</tr>
<tr>
<td>(LISTGET1 LST PROP) II: 3.16</td>
</tr>
<tr>
<td>Listing file directories III: 24.33</td>
</tr>
<tr>
<td>LISTING? (Compiler Question) II: 18.1</td>
</tr>
<tr>
<td>(LISP X) I: 3.1; 9.2</td>
</tr>
<tr>
<td>LISPST checks in pattern matching I: 12.25</td>
</tr>
<tr>
<td>(LISPSTPUT LST PROP VAL) I: 3.16</td>
</tr>
<tr>
<td>(LISPSTPUT1 LST PROP VAL) I: 3.16</td>
</tr>
<tr>
<td>Lists I: 3.1; 3.3</td>
</tr>
<tr>
<td>(LITATOM X) I: 2.1; 9.1</td>
</tr>
<tr>
<td>Litatoms I: 2.1; 9.1</td>
</tr>
<tr>
<td>Literal atoms I: 2.1</td>
</tr>
<tr>
<td>(LLSH X N) I: 7.8</td>
</tr>
<tr>
<td>(LO M) (Editor Command) II: 16.41</td>
</tr>
<tr>
<td>(LOAD FILE LDFLG PRINTFLG) II: 17.6; 13.40; 18.13</td>
</tr>
<tr>
<td>(LOAD? FILE LDFLG PRINTFLG) II: 17.6</td>
</tr>
<tr>
<td>(LOADBLOCK FN FILE LDFLG) II: 17.8</td>
</tr>
<tr>
<td>(LOADBYTE N POS SIZE) I: 7.10</td>
</tr>
<tr>
<td>(LOADCOMP FILE LDFLG) II: 17.8</td>
</tr>
<tr>
<td>(LOADDEF NAME TYPE SOURCE) II: 17.28</td>
</tr>
<tr>
<td>LOADEDFILEST (Variable) I: 12.11; II: 17.20</td>
</tr>
<tr>
<td>(LOADFNS FNS FILE LDFLG VARS) II: 17.6</td>
</tr>
<tr>
<td>(LOADFROM FILE FNS LDFLG) II: 17.8; 18.16</td>
</tr>
<tr>
<td>Loading files II: 17.5</td>
</tr>
<tr>
<td>LOADOPTIONS (Variable) II: 17.6</td>
</tr>
<tr>
<td>(LOADTIMECONSTANT X) II: 18.8</td>
</tr>
<tr>
<td>(LOADVARS VARS FILE LDFLG) II: 17.8</td>
</tr>
<tr>
<td>Local CLISP declarations II: 21.13</td>
</tr>
<tr>
<td>Local hard disk device III: 24.21</td>
</tr>
<tr>
<td>Local record declarations I: 8.7; 11; II: 21.13</td>
</tr>
<tr>
<td>Local variables I: 9.8; II: 18.5; 22.5</td>
</tr>
<tr>
<td>LOCALLY (use in Masterscope) II: 19.8</td>
</tr>
<tr>
<td>\LOCALNDBS (Variable) III: 31.39</td>
</tr>
<tr>
<td>Localvars II: 18.5</td>
</tr>
<tr>
<td>(LOCALVARS VAR1 ... VARN) (File Package Command) II: 17.37</td>
</tr>
<tr>
<td>LOCALVARS (in Masterscope Set Specification) II: 19.12</td>
</tr>
<tr>
<td>LOCALVARS (Variable) II: 18.5</td>
</tr>
<tr>
<td>Location specification in the editor II: 16.23; 16.24; 60</td>
</tr>
<tr>
<td>LOCATION UNCERTAIN (Printed by Editor) II: 16.14</td>
</tr>
<tr>
<td>LOCF (Macro) I: 8.11</td>
</tr>
<tr>
<td>(LOG X) I: 7.13</td>
</tr>
<tr>
<td>(LOGAND X1 X2 ... XN) I: 7.8</td>
</tr>
<tr>
<td>Logging into file servers III: 24.39</td>
</tr>
<tr>
<td>Logical arithmetic functions I: 7.8</td>
</tr>
<tr>
<td>Logical volumes III: 24.21</td>
</tr>
<tr>
<td>(LOGIN HOSTNAME LDFLG DIRECTORY MSG) III: 24.40</td>
</tr>
<tr>
<td>LOGINHOST/DIR (Variable) I: 12.3; III: 24.11</td>
</tr>
<tr>
<td>(LOGNOT N) (Macro) I: 7.9</td>
</tr>
<tr>
<td>Logo window III: 28.2</td>
</tr>
<tr>
<td>(LOGOR X1 X2 ... XN) I: 7.8</td>
</tr>
<tr>
<td>(LOGOUT FAST) I: 12.7</td>
</tr>
<tr>
<td>(LOGOW STRING WHERE TITLE ANGLEDELTA) III: 28.2</td>
</tr>
<tr>
<td>LOGOW (Variable) III: 28.2</td>
</tr>
<tr>
<td>(LOGXOR X1 X2 ... XN) I: 7.8</td>
</tr>
<tr>
<td>(LONG-SITE-NAME) I: 12.12</td>
</tr>
<tr>
<td>(LOOKUP.NS.SERVER NAME TYPE FULLFLG) III: 31.10</td>
</tr>
<tr>
<td>(LOWER X) (Editor Command) II: 16.53</td>
</tr>
<tr>
<td>LOWER (Editor Command) II: 16.52</td>
</tr>
<tr>
<td>Lower case characters I: 2.10</td>
</tr>
<tr>
<td>Lower case comments III: 26.46</td>
</tr>
</tbody>
</table>
INDEX

Lower case in CLISP II: 21.27
Lower case input III: 30.8
(LOWERCASE FLG) II: 21.27
LowerLeftCursor (Variable) III: 30.15
LowerRightCursor (Variable) III: 30.15
(LP COMS₁ ... COMSₙ) (Editor Command) II: 16.60; 16.61
LPARKEY (Variable) II: 20.14; 20.6
(LPQ COMS₁ ... COMSₙ) (Editor Command) II: 16.61
LPT (printer device) III: 29.4
(LRSH X N) I: 7.8
(LSH X N) I: 7.8
LSTFIL (Variable) II: 18.1
(LSUBST NEW OLD EXPN) I: 3.13
LT (CLISP Operator) II: 21.8
(LVLPRIN1 X FILE CARLVL CDRLVL TAIL) III: 25.13
(LVLPRIN2 X FILE CARLVL CDRLVL TAIL) III: 25.13
(LVLPRINT X FILE CARLVL CDRLVL TAIL) III: 25.13

M
(M C (ARG₁ ... ARGₙ) COMS₁ ... COMSₙ) (Editor Command) II: 16.62
(M C ARG COMS₁ ... COMSₙ) (Editor Command) II: 16.62
(M C COMS₁ ... COMSₙ) (Editor Command) II: 16.62
(MACHINE-INSTANCE) I: 12.12
(MACHINE-TYPE) I: 12.12
(MACHINE-VERSION) I: 12.12
(MACHINETYPE) I: 12.13
MACRO (File Package Command Property) II: 17.45
(MACRO . MACRO) (in Masterscope template) II: 19.21
MACRO (Property Name) I: 10.21; II: 17.18; 18.11
MACRO (type of read macro) III: 25.39
Macro expansion in Masterscope II: 19.17
MACROCHARS (ASKUSER option) III: 26.17
MACROPROPS (Variable) I: 10.21
Macros I: 10.21
(MACROS LITATOM₁ ... LITATOMₙ) (File Package Command) II: 17.35
MACROS (File Package Type) II: 17.24
Macros in the editor II: 16.62
Maintanance panel III: 30.24
(MAINWINDOW WINDOW RECURSEFLG) III: 28.47
MAINWINDOW (Window Property) III: 28.54

MAINWINDOW MAXSIZE (Window Property) III: 28.54
MAINWINDOW MINSIZE (Window Property) III: 28.54
(MAKE ARGNAME EXP) (Editor Command) II: 16.57
(MAKEBITTABLE L NEG A) I: 4.6
(MAKEFILE FILE OPTIONS REPRINTFNS SOURCEFILE) II: 17.10; 17.14; 18.16; 20.24
MAKEFILE and CLISP II: 21.26
MAKEFILEFORMS (Variable) II: 17.12
MAKEFILEOPTIONS (Variable) II: 17.10
MAKEFILEREMAKEFLG (Variable) II: 17.15; 17.11
(MAKEFILES OPTIONS FILES) II: 17.12
(MAKEFN (FN . ACTUALARGS) ARGLIST N₁ N₂) (Editor Command) II: 16.56
(MAKEKEYLST LST DEFAULTKEY LCASEFLG AUTOCOMPLETEFLG) III: 26.13
(MAKENEWCOM NAME TYPE — —) II: 17.49
(MAKEYSYS FILE NAME) I: 12.9
MAKEYSYSDATE (Variable) I: 12.13; 12.10
MAKEYSYNAME (Variable) I: 12.13
(MAKEWITHINREGION REGION LIMITREGION) III: 27.2
Manipulating file names III: 24.5
(MAP MAPX MAPFN₁ MAPFN₂) I: 10.15
(MAP.PROCESSES MAPFN) II: 23.5
(MAP2C MAPX MAPY MAPFN₁ MAPFN₂) I: 10.16
(MAP2CAR MAPX MAPY MAPFN₁ MAPFN₂) I: 10.16
(MAPATOMS FN) I: 2.11
(MAPC MAPX MAPFN₁ MAPFN₂) I: 10.15
(MAPCAR MAPX MAPFN₁ MAPFN₂) I: 10.15
(MAPCON MAPX MAPFN₁ MAPFN₂) I: 10.15; II: 21.13
(MAPCONC MAPX MAPFN₁ MAPFN₂) I: 10.16; II: 21.13
(MAPDML MAPDLFN MAPDLPOS) I: 11.13
(MAPHASH HARRAY MAPFN) I: 6.3
(MAPLIST MAPX MAPFN₁ MAPFN₂) I: 10.15
(MAPRELATION RELATION MAPFN) II: 19.24
(MAPPRINT LST FILE LEFT RIGHT SEP PFN LISPXPRINTFLG) I: 10.17
(MARK LITATOM) (Editor Command) II: 16.28
MARK (Editor Command) II: 16.27; 16.28
Mark-and-sweep garbage collection II: 22.1
(MARKASCHANGED NAME TYPE REASON) II: 17.17
MARKASCHANGEDFNS (Variable) II: 17.18
Marking changes II: 17.17
INDEX

MRR (Font face) IlI: 27.26
MSMACROPROPS (Variable) IlI: 19.17
(MSMARKCHANGED NAME TYPE REASON) IlI: 19.24
(MSNEEDUNSAVE FNS MSG MARKCHANGEFLG) IlI: 19.24
MSNEEDUNSAVE (Variable) IlI: 19.25
MSPRINTFLG (Variable) IlI: 19.2
Multiple streams to a file IllI: 24.15
MULTIPLY DEFINED TAG (Error Message) IlI: 18.23
MULTIPLY DEFINED TAG, ASSEMBLE (Error Message) IlI: 18.23
MULTIPLY DEFINED TAG, LAP (Error Message) IlI: 18.23

N
(-N E_1 . . . E_M) (N > 1) (Editor Command) IlI: 16.29
(N E_1 . . . E_M) (N > 1) (Editor Command) IlI: 16.29
(N E_1 . . . E_M) (Editor Command) IlI: 16.29
(N) (N > 1) (Editor Command) IlI: 16.29
-N (N > 1) (Editor Command) IlI: 16.15
-N (N > 1) (Editor Command) IlI: 16.15; 16.29;
16.55
-N (N a number) (PRINTOUT command) IllI: 25.26
N (N a number) (PRINTOUT command) IllI: 25.25;
25.30
NAME (File name field) IllI: 24.6
NAME (Process Property) IlI: 23.2
NAME LITATOM (ARG_1 . . . ARG_N) : EventSpec (Pro.
Asst. Command) IlI: 13.14
NAME LITATOM ARG_1 . . . ARG_N : EventSpec (Pro.
Asst. Command) IlI: 13.14
NAME LITATOM EventSpec (Pro. Asst. Command)
IlI: 13.14; 13.16,33
NAMES RESTORED (Printed by System) IlI: 15.9
NAMESCHANGED (Property Name) IlI: 15.5
(NARGS FN) IlI: 10.8
(NCHARS X FLG RDTBL) IlI: 2.9; 4.2
(NCONC X_1 X_2 . . . X_N) IlI: 3.5; 3.6; IlI: 21.13
(NCONC LST X) IlI: 3.5; 3.6; IlI: 21.13
(NCREATE TYPE OLDJOB) IlI: 22.22
(NDIR FILEGROUP COM_1 . . . COM_N) IlI: 24.35
NEGATE (Editor Command) IlI: 16.54
(NEGATE X) IlI: 3.20; IlI: 16.54
(NEQ X Y) IlI: 9.3
NETWORKOSTYPES (Variable) IlI: 24.38
NEVER FORM (I.S. Operator) IlI: 9.11
NEW (MAKEFILE option) IlI: 17.11
(NEW/FN FN) IlI: 13.41
NEWCOM (File Package Type Property) IlI: 17.31
NEWREGIONFN (Window Property) IlI: 28.18
(NEWRESOURCE RESOURCENAME . ARGs) (Macro)
IlI: 12.23
NEWVALUE (Variable) IlI: 8.12
(NEX COM) (Editor Command) IlI: 16.26
NEX (Editor Command) IlI: 16.26
NIL (Editor Command) IlI: 16.55; 16.59
NIL (in block declarations) IlI: 18.18
NIL (in Masterscope template) IlI: 19.18
NIL (Litatom) IlI: 2.3; 9.2
NIL (Primary stream) IllI: 25.1
NILCOMS (Variable) IlI: 17.13
(NILL X_1 . . . X_N) IlI: 10.18
NILNUMPRINTFLG (Variable) IllI: 25.16
NLAMA (Variable) IlI: 18.9; 18.14
NLAMBDAs (Litatom) IlI: 10.2
NLAMBDAs (Macro Type) IlI: 10.22
Nlambda functions IlI: 10.2
Nlambda-nospread functions IlI: 10.6
Nlambda-spread functions IlI: 10.4
(NLAMBDAS-ARGS X) IlI: 10.13
NLAML (Variable) IlI: 18.9; 18.14
(NLEFT L N TAIL) IlI: 3.9
(NLSTP X) IlI: 3.1; 9.2
(NLSETQ FORM) IlI: 9.9; IlI: 14.22; 13.30
NLSETQGAG (Variable) IlI: 14.22
NO BINARY CODE GENERATED OR LOADED (Error
Message) IlI: 18.23
(FN - NO BREAK INFORMATION SAVED) (value of
REBREAK) IlI: 15.8
NO DO, COLLECT, OR JOIN (Error Message) IlI: 9.19
NO FILE PACKAGE COMMAND FOR (Error Message)
IlI: 17.40
NO LONGER INTERPRETED AS FUNCTIONAL
ARGUMENT (Error Message) IlI: 18.23
NO PROPERTY FOR (Error Message) IlI: 17.38
NO USERMACRO FOR (Error Message) IlI: 17.34
NO VALUE SAVED: (Error Message) IlI: 13.29
NOBIND (Litatom) IlI: 2.2; 11.8; IlI: 13.28-29; 17.5
NOBREAKS (Variable) IlI: 15.7
NOCASEFLG (ASKUSER option) IllI: 26.15
NOCLEARSTKLST (Variable) IlI: 11.10
NODIRCORE (file device) IllI: 24.30
NOECHOFLG (ASKUSER option) IllI: 26.16
NOESC (type of read macro) IllI: 25.40
NOESCQUOTE (type of read macro) IllI: 25.40
NOEVAL (Litatom) IlI: 21.21
INDEX

NOFILESPELFLG (Variable) III: 24.32
NOFIXFNSLST (Variable) II: 21.21; 17.8; 18.12; 21.19
NOFIXVARSLST (Variable) II: 21.21; 17.8; 18.12; 21.15,19
NON-ATOMIC CAR OF FORM (Error Message) II: 18.23
Non-existent directory (Error Message) III: 24.10
NON-NUMERIC ARG (Error Message) I: 5.2; 7.3,6,11; II: 14.28
NONE (syntax class) III: 30.6
NONIMMED (type of read macro) III: 25.41
NONIMMEDIATE (type of read macro) III: 25.41
NOPRINT (Litatom) II: 13.29
(NORMALCOMMENTS FLG) III: 26.44; 26.45
NOSAVE (Function) II: 13.41
NOSAVE (Litatom) II: 13.29,40
NOSCROLLBARS (Window Property) III: 28.26; 28.25
NOSPELLFLG (Variable) II: 20.13; 21.21; III: 24.32
Nospread functions I: 10.3
NOSTACKUNDO (Litatom) II: 13.29
NOT X I: 9.3
NOT A BINDABLE VARIABLE (Error Message) II: 18.23
NOT A FUNCTION (Error Message) I: 10.8; II: 15.11
NOT BLOCKED (Printed by Editor) II: 16.65
(NOT BROKEN) (value of UNBREAK0) II: 15.8
not changed, so not unsaved (Printed by Editor) II: 16.69
NOT COMPILEABLE (Error Message) II: 18.22; 18.14,18
(FILE NOT DUMPED) (returned by MAKEFILE) II: 17.12
not editable (Error Message) II: 16.70-71
NOT FOUND (Error Message) II: 18.22
(FN NOT FOUND) (printed by break) II: 14.7
(NOT FOUND) (printed by BREAKIN) II: 15.6-7
FILENAME NOT FOUND (printed by LISTFILES) II: 17.14
(FN1 NOT FOUND IN FN2) (value of BREAK0) II: 15.4
NOT FOUND, SO IT WILL BE WRITTEN ANEW (Error Message) II: 17.51
NOT IN FILE - USING DEFINITION IN CORE (Error Message) II: 18.22
NOT ON BLKFNS (Error Message) II: 18.22; 18.19-20
NOT ON FILE, COMPILING IN CORE DEFINITION
(Error Message) II: 18.18
(FN NOT PRINTABLE) (returned by PRETTYPRINT) III: 26.40
NOT-FOUND: (Litatom) II: 17.7
(NOT ANY SOMEFN1 SOMEFN2) I: 10.17
NOTCOMPILEDFILES (Variable) II: 17.14; 17.10-11
(NOTE VAL LSTFLG) I: 11.20
NOTE: BRKEXP NOT CHANGED. (Printed by Break) II: 14.12
(NOTE EVERY EVERYX EVERYFN1 EVERYFN2) I: 10.17
NOTFIRST (DECLARE: Option) II: 17.42
nothing saved (Printed by Editor) II: 16.64-65
nothing saved (Printed by System) II: 13.26; 13.13
Noticing files II: 17.19
(NOTIFY.EVENT EVENT ONCEONLY) II: 23.7
NOTLISTEDFILES (Variable) II: 17.14; 17.10
NOTRACE SET (Masterscope Path Option) II: 19.16
NS character I/O III: 25.22; 25.6,9,19
NS characters I: 2.12; 4.2; III: 25.19-20,36, 27.27; 30.3,6-7,20
NS.ECHOUSER (Function) III: 31.38
NSADDRESS (Data type) III: 31.7; 31.17
NSNAME (Data type) III: 31.8; 31.17-18
(NSNAME.TO.STRING NSNAME FULLNAMEFLG) III: 31.9
(NSOCKETEVENT NSOC) III: 31.37
(NSOCKETNUMBER NSOC) III: 31.37
(NSPRINT PRINTER FILE OPTIONS) III: 31.12
NSPRINT.DEFAULT.MEDIUM (Variable) III: 29.2
(NSPRINTER.PROPERTIES PRINTER) III: 31.12
(NSPRINTER.STATUS PRINTER) III: 31.12
(NTH COM) (Editor Command) II: 16.26
(NTH N) (Editor Command) II: 16.17; 16.26
(NTH X N) I: 3.9
(NTHCHAR X N FLG RDTBL) I: 2.10
(NTHCHARCODE X N FLG RDTBL) I: 2.13
NULL (file device) III: 24.30
(NULL X) I: 9.3
Null strings I: 4.1
NULLDEF (File Package Type Property) II: 17.30
(NUMBERP X) I: 7.2; 9.1
Numbers I: 7.1; 9.1; III: 25.4
(NX N) (Editor Command) II: 16.16
NX (Editor Command) II: 16.16

INDEX 28
INDEX

(RECORDACCESS FIELD DATUM DEC TYPE NEWVALUE)  I: 8.16
(RECORDACCESSFORM FIELD DATUM TYPE NEWVALUE)  I: 8.17
(RECORDFIELDNAMES RECORDNAME — )  I: 8.16
(RECORDS REC_1 ... REC_N) (File Package Command)  I: 8.2,11; II: 17.38
RECORDS (File Package Type) II: 17.24
REDEFINE? (Compiler Question) II: 18.1
(FN redefined) (printed by system) I: 10.10
Redisplay (Window Menu Command) III: 28.4
(REDISPLAYW WINDOW REGION ALWAYSFLG) III: 28.16
REDO EventSpec UNTIL FORM (Prog. Asst. Command) II: 13.8
REDO EventSpec WHILE FORM (Prog. Asst. Command) II: 13.8
REDO EventSpec N TIMES (Prog. Asst. Command) II: 13.8
REDO EventSpec (Prog. Asst. Command) II: 13.8; 13.33
REDOCNT (Variable) II: 13.9
REFERENCE (Masterscope Relation) II: 19.8
Reference-counting garbage collection II: 22.2
ReFetch (Inspect Window Command) III: 26.4
REGION (Record) III: 27.1
REGION (Window Property) III: 28.34; 28.24
(REGIONP X) III: 27.2
Regions III: 27.1
(REGIONSINTERSECTP REGION1 REGION2) III: 27.2
Registering image objects III: 27.39
(REDISPLAVWWINDOW) II: 6.3
REJECTMAINCOMS (Window Property) III: 28.51
SET RELATION SET (Masterscope Command) II: 19.5
Relations in Masterscope II: 19.7
(RELDrawTO DX DY WIDTH OPERATION STREAM COLOR DASHING) III: 27.18
(RELEASE.ETHERPACKET EPKT) (Function) III: 31.39
(RELEASE.MONITORLOCK LOCK EVENIFNOTMINE) II: 23.9
(RELEASE.PUP PUP) III: 31.28
(RELEASE.XIP XIP) III: 31.36
Releasing stack pointers I: 11.9
(RELMOVETO DX DY STREAM) III: 27.14
(RELMOVEW WINDOW POSITION) III: 28.19
(RELPROCESSP PROchandle) II: 23.4
(RELSTK POS) I: 11.9; 11.10

INDEX

(RANDOM X) I: 7.14
(RATELOG FLG) III: 25.4
(RATOM FILE RDTBL) III: 25.4; 25.36; 32.10
(RATOMS A FILE RDTBL) III: 25.4
RAVEN (Printer type) III: 29.5
(RC X Y) (Editor Command) II: 16.46
RC (MAKEFILE option) II: 17.10
(RC1 X Y) (Editor Command) II: 16.46
(READ FILE RDTBL FLG) III: 25.3; 32.10
Read macros III: 25.39
Read tables III: 25.33; 25.3,8
READ-MACRO CONTEXT ERROR (Error Message) II: 14.30
(READBYTE FILE) III: 27.4
READBUF (Variable) II: 13.36; 13.38
(READC FILE RDTBL) III: 25.5; 30.10
(READCODE FILE RDTBL) III: 25.5
(READCOMMENT FILE RDTBL LAST) III: 26.45
READDATE (File Attribute) III: 24.18
(READFILE FILE RDTBL ENDTOKEN) III: 25.33
(READIMAGEOBJ STREAM GETFN NOERROR) III: 27.41
(READLINE RDTBL — — ) II: 13.36;
13.24,32,35,37,43; 16.67
(READMACROS FLG RDTBL) III: 25.42
(READPERFILE FLG) III: 25.6
(READTABLEP RDTBL) III: 25.34
READVISE (Property Name) II: 15.12-13
(REALFRAMEP POS INTERPFLG) II: 11.13
_REALMEMORYSIZE) I: 12.10
(REALSTKTH N POS INTERPFLG OLDPOS) I: 11.13
REANALYZE SET (Masterscope Command) II: 19.4
(REBREAK X) II: 15.8; 15.4
(RECLAIM) II: 22.3
(RECLAIMMIN N) II: 22.3
RECLAIMWAIT (Variable) II: 22.3
(RECLOCK RECNAME — — — — ) I: 8.16
Recognition of file versions III: 24.11
(RECOMPILE PFFILE CFILE FNS) II: 18.15; 17.12;
18.14,18
RECOMPILEDDEFAULT (Variable) II: 18.16; 18.22
Reconstruction in pattern matching I: 12.30
RECORD (in Masterscope template) II: 19.20
RECORD (Record Type) I: 8.7
Record declarations I: 8.6
Record declarations in CLISP II: 21.14
Record package I: 8.1
Record types I: 8.7; 8.6
(RECORDACCESS FIELD DATUM DEC TYPE NEWVALUE) I: 8.16
(RECORDACCESSFORM FIELD DATUM TYPE NEWVALUE) I: 8.17
(RECORDFIELDNAMES RECORDNAME — ) I: 8.16
(RECORDS REC_1 ... REC_N) (File Package Command) I: 8.2,11; II: 17.38
RECORDS (File Package Type) II: 17.24
REDEFINE? (Compiler Question) II: 18.1
(FN redefined) (printed by system) I: 10.10
Redisplay (Window Menu Command) III: 28.4
(REDISPLAYW WINDOW REGION ALWAYSFLG) III: 28.16
REDO EventSpec UNTIL FORM (Prog. Asst. Command) II: 13.8
REDO EventSpec WHILE FORM (Prog. Asst. Command) II: 13.8
REDO EventSpec N TIMES (Prog. Asst. Command) II: 13.8
REDO EventSpec (Prog. Asst. Command) II: 13.8;
13.33
REDOCNT (Variable) II: 13.9
REFERENCE (Masterscope Relation) II: 19.8
Reference-counting garbage collection II: 22.2
ReFetch (Inspect Window Command) III: 26.4
REGION (Record) III: 27.1
REGION (Window Property) III: 28.34; 28.24
(REGIONP X) III: 27.2
Regions III: 27.1
(REGIONSINTERSECTP REGION1 REGION2) III: 27.2
Registering image objects III: 27.39
(REDISPLAVWWINDOW) II: 6.3
REJECTMAINCOMS (Window Property) III: 28.51
SET RELATION SET (Masterscope Command) II: 19.5
Relations in Masterscope II: 19.7
(RELDrawTO DX DY WIDTH OPERATION STREAM COLOR DASHING) III: 27.18
(RELEASE.ETHERPACKET EPKT) (Function) III: 31.39
(RELEASE.MONITORLOCK LOCK EVENIFNOTMINE) II: 23.9
(RELEASE.PUP PUP) III: 31.28
(RELEASE.XIP XIP) III: 31.36
Releasing stack pointers I: 11.9
(RELMOVETO DX DY STREAM) III: 27.14
(RELMOVEW WINDOW POSITION) III: 28.19
(RELPROCESSP PROchandle) II: 23.4
(RELSTK POS) I: 11.9; 11.10

INDEX
REMAKE (MAKEFILE option) II: 17.11
Remaking a symbolic file II: 17.15
REMEMBER EventSpec (Prog. Asst. Command) II: 13.17
(REMOVE X Y) I: 7.3
(REMOVE X L) I: 3.19
(REMOVEPROMPTWINDOW MAINWINDOW) III: 28.50
(REMOVEWINDOW WINDOW) III: 28.47
(REMPROP ATM PROP) I: 2.6
(REMPROPIMPIER ATM PROPS) I: 2.6
(RENAME OLD NEW TYPES FILES METHOD) II: 17.29
(RENAMEFILE OLDFILE NEWFILE) III: 24.31
Renaming files III: 24.31
Reopening files III: 24.20
(REPACK @) (EditorCommand) II: 16.53
(REPACK (EditorCommand) II: 16.53
(REPAINTFN (Window Property) III: 28.16; 28.38
(REPEAT EventSpec UNTIL FORM (Prog. Asst. Command) II: 13.8
(REPEAT EventSpec WHILE FORM (Prog. Asst. Command) II: 13.8
(REPEAT EventSpec (Prog. Asst. Command) II: 13.8
(REPEATUNTIL N (N a number) (I.S. Operator) I: 9.16
(REPEATUNTIL FORM (I.S. Operator) I: 9.16
(REPEATWHILE FORM (I.S. Operator) I: 9.16
(Replace (DEdit Command) II: 16.7
(REPLACE @ WITH E1 ... EM (EditorCommand) II: 16.33
(REPLACE @ BY E1 ... EM (EditorCommand) II: 16.33
(REPLACE (in Masterscope template) II: 19.19
(REPLACE (Masterscope Relation) II: 19.9
(REPLACE (Record Operator) I: 8.2; 8.3; II: 21.10
(REPLACE UNDEFINED FOR FIELD (Error Message) I: 8.12
(REPLACEFIELD DESCRIPTOR DATUM NEWVALUE) I: 8.22
Replacements in pattern matching I: 12.29
(REPOSITIONATTACHEDWINDOWS WINDOW) III: 28.47
(REPRINTINGATTACHEDWINDOWS WINDOW) III: 28.47
(REPRINTFILE DESCRIPTOR DATUM NEWVALUE) I: 8.22
(REPRINT WHOLE FILE cousins) I: 8.22
(REPRINT WHOLE FILE) II: 13.33
(REPRINT) II: 13.33
(REPRINT) II: 13.33
(RETYPE (syntax class) III: 30.6
(REUSING (in CREATE form) I: 8.4
(RETRY EventSpec (Prog. Asst. Command) II: 13.9; 13.33
(RETO POS VAL FLG) I: 11.9
(RETURN (ASKUSER option) III: 26.15
(RETURN FORM (Break Command) II: 14.6
(RETURN X) I: 9.8
(RETURN (in iterative statement) I: 9.18
(RETURN (in Masterscope template) II: 19.19
(RETURN) II: 13.33
Reusing stack pointers  I: 11.10
(REVERSE L)  I: 3.19
REVERT (Break Command)  II: 14.10
revert (Break Window Command)  II: 14.3
(RI N M) (Editor Command)  II: 16.41
RIGHT (key indicator)  III: 30.17
Right margin  III: 27.11
Right-button background menu  III: 28.6
Right-button window menu  III: 28.3
RIGHTBRACKET (Syntax Class)  III: 25.35
RIGHTBUTTONFN (Window Property)  III: 28.28
RIGHTPAREN (Syntax Class)  III: 25.35
(RINGBELLS N)  III: 30.24
(RO N) (Editor Command)  II: 16.41
Root name of a file  II: 17.4
ROOTFILENAME (Function)  II: 17.4, 20
(ROT X N SIZE)  I: 7.10
ROTATION (Font property)  III: 27.27
(RPAQ VAR VALUE)  II: 17.54, 13.28, 17.5
(RPAQ VAR VALUE)  II: 17.54, 17.5
(RPAQQ VAR VALUE)  II: 17.54, 13.28, 17.5, 50
RPARKEY (Variable)  II: 20.14, 20.6
#RPARS (Variable)  III: 26.47
(RPLACA X Y)  I: 3.2; II: 21.13
(RPLACD X Y)  I: 3.2; II: 21.13
(RPLCHARCODE X N CHAR)  I: 4.5
(RPLNODE X A D)  I: 3.2; II: 13.40
(RPLNODE2 X Y)  I: 3.3; II: 13.40
(RPLSTRING X N Y)  I: 4.4
(RPT N FORM)  I: 10.15
(RPTQ N FORM1 FORM2 ... FORMN)  I: 10.15
(RSH X N)  I: 7.8
(RSTRING FILE RDtbl)  III: 25.4
RUBOUT (Interrupt Channel)  II: 23.15; III: 30.3
Run-encoding of NS characters  III: 25.22
Run-on spelling corrections  II: 20.22; 20.4
RUNONF LG (Variable)  II: 20.14; 20.22

S
S LITATOM @ (Editor Command)  II: 16.29
S (Response to Compiler Question)  II: 18.2
(SASSOC KEY AS LT)  I: 3.15
SAVING cursor  I: 12.7
SAVE (Editor Command)  II: 16.49; 16.51, 72
SAVE EXPR S? (Compiler Question)  II: 18.2
(SAVEDEF NAME TYPE DEFINITION)  II: 17.27
(SAVEPUT ATM PROP VAL)  II: 17.55
(SAVESET NAME VALUE TOPFLG FLG)  II: 13.29
SAVESETQ (Function)  II: 13.28
SAVESETQQ (Function)  II: 13.28
SaveVM (Function)  III: 28.6
(SAVEVM —)  I: 12.7
SAVEVMMAX (Variable)  I: 12.7
SAVEVMWAIT (Variable)  I: 12.7
Saving bitmaps on files  III: 27.3
SAVINGCURSOR (Variable)  I: 12.7; III: 30.15
SCALE (Font property)  III: 27.28
(SCAVENGEDSKDIRECTORY VOLUMENAME SILEN T)  III: 24.23
(SCRATCHLIST L ST X1 X2 ... XN)  I: 3.8
(SCREENBITMAP)  III: 30.22
SCREENHEIGHT (Variable)  III: 30.22
Screens  I: 12.4; III: 30.22
SCREENWIDTH (Variable)  III: 30.22
(SCROLL.HANDLER WINDOW)  III: 28.24
SCROLLBARWIDTH (Variable)  III: 28.24
(SCROLLBYREPAINTFN WINDOW DELTAX DELTAY CONTINUOUSFLG)  III: 28.25
ScrollDownCursor (Variable)  III: 30.15
SCROLLEXTENTUSE (Window Property)  III: 28.26
SCROLLEXTENTUSE (Window Property)  III: 28.26; 28.25
SCROLLFN (Window Property)  III: 28.26; 28.25, 38
Scrolling  III: 28.23; 27.24
ScrollLeftCursor (Variable)  III: 30.16
ScrollRightCursor (Variable)  III: 30.16
ScrollUpCursor (Variable)  III: 30.15
(SCROLLW WINDOW DELTAX DELTAY CONTINUOUSFLG)  III: 28.24
SCROLLWAITTIME (Variable)  III: 28.24
Searching file directories  III: 24.31
Searching files  III: 25.20
Searching in the editor  II: 16.18; 16.20
Searching strings  I: 4.5
SEARCHING... (Printed by BREAKIN)  II: 15.7
(SEARCHPDL SRCHFN SRCHPOS)  I: 11.14
SECONDS (Timer Unit)  I: 12.16
(SEE FROM FILE TO FILE)  III: 26.41
(SEE FROM FILE TO FILE)  III: 26.41
Segment patterns in pattern matching  I: 12.27
(SELCHARQ E CLAUSE1 ... CLAUSEN DEFAULT) (Macro)  I: 2.15
SELECTABLEITEMS (Window Property)  III: 26.8
(SELECTC X CLAUSE1 CLAUSE2 ... CLAUSEK DEFAULT)  I: 9.7
SELECTIONFN (Window Property)  III: 26.8
STACK PTR HAS BEEN RELEASED (Error Message) II: 14.30
(STACKP X) I: 11.9
STANDARD (Font face) III: 27.26
(START.CLEARINGHOUSE RESTARTFLG) III: 31.9
STF (Response to Compiler Question) II: 18.2
(STKAPPLY POS FN ARGS FLG) I: 11.8
(STKARG N POS ->) I: 11.7; II: 14.8
(STKARGNAME N POS) I: 11.7
(STKARGS POS ->) I: 11.7
(STKEVAL POS FORM FLG ->) I: 11.8; II: 14.8
(STKNAME POS) I: 11.6
(STKNARGS POS ->) I: 11.7
(STKTH N POS OLPPOS) I: 11.6
(STKTHNAME N POS) I: 11.6
(STKPOS FRAMENAME N POS OLPPOS) I: 11.5
(STKSCAN VAR IPSOS OPOS) I: 11.6
STOP (at the end of a file) II: 17.6; III: 25.33
Stop (DEdit Command) II: 16.10
STOP (Editor Command) II: 16.49; 15.6; 16.53,72
$STOP (escape-STOP) (TYPE-AHEAD command) II: 13.18
(STORAGE TYPES PAGETHRESHOLD) II: 22.3
Storage allocation II: 22.1
STORAGE FULL (Error Message) II: 14.30; 23.15
STORAGE.ARRAYSIZES (Variable) II: 22.4
(STORAGE.LEFT) II: 22.5
STOREFN (Window Property) III: 26.8
Storing files II: 17.10
(STREAM P X) III: 25.2
Streams III: 24.1
(STREQUAL X Y) I: 4.1
STRF (Variable) II: 18.1; 18.2,14
String pointers I: 4.1
(STRING-EQUAL X Y) I: 4.2
STRINGDELIM (Syntax Class) III: 25.35
(STRINGHASHBITS STRING) I: 6.5
(STRINGP X) I: 4.1; 9.2
(STRINGREGION STR STREAM PRIN2FLG RDTBL) III:
27.30
Strings I: 4.1; 9.2; III: 25.3
(STRINGWIDTH STR FONT FLG RDTBL) III: 27.30
(STRMBOUTFN STREAM CHARCODE) (Stream Method) III: 27.48
(STRPOS PAT STRING START SKIP ANCHOR TAIL CASEARRAY BACKWARDSFLG) I: 4.5; III:
25.20
(STRPOSL A STRING START NEG BACKWARDSFLG) I: 4.6

Structure modification commands in the editor II:
16.29
.SUB (PRINTOUT command) III: 25.27
(SUB1 X) I: 7.6
(SUBATOM X N M) I: 2.8
Subdeclarations I: 8.14
SUBITEMFN (Menu Field) III: 28.39
SUBITEMS (Litatom) III: 28.39
(SUBLIS ALST EXPR FLG) I: 3.14
(SUBPAIR OLD NEW EXPR FLG) I: 3.14
SUBRECORD (in record declarations) I: 8.14
(SUBREGIONP LARGEREGION SMALLREGION) III:
27.2
(SUBSET MAPX MAPFN1 MAPFN2) I: 10.17
(SUBST NEW OLD EXPR) I: 3.13
Substitution macros I: 10.22
SUBSTRING X N M OLPTR) I: 4.3
SUCHTHAT (I.S. Operator) I: 9.22
SUCHTHAT (in event address) II: 13.6
SUM FORM (I.S. Operator) I: 9.11
.SUP (PRINTOUT command) III: 25.27
SURROUND (Editor Command) II: 16.37
SUSPEND (Process Property) II: 23.2
(SUSPEND.PROCESS PROC) II: 23.6
SUSPICIOUS PROG LABEL (Error Message) II: 21.19
SVFLG (Variable) II: 18.1-2
(SW N M) (Editor Command) II: 16.47
(SWAP DATUM1 DATUM2) (Change Word) II: 8.19
Swap (DEdit Command) II: 16.8
(SWAP @1 @2) (Editor Command) II: 16.47
SWAPBLOCK TOO BIG FOR BUFFER (Error Message)
II: 14.31
SWAPC (Editor Command) II: 16.54
(SWAPPUPPORTS PUP) III: 31.31
Switch (DEdit Command) II: 16.7
Symbols I: 2.1
SYNONYM (in record declarations) I: 8.15
Synonyms for file package commands II: 17.47
Synonyms for file package types II: 17.32
Synonyms in spelling correction II: 20.16
Syntax classes III: 25.35
(SYNTAXP CODE CLASS TABLE) III: 25.37
SYS/OUT cursor I: 12.8
(SYSBUF FLG) III: 30.11; 30.12
SYSFILES (Variable) II: 17.6
SYSHASHARRAY (Variable) I: 6.1
SYSLOAD (LOAD option) II: 17.5; 17.6; 20.10
(SYSOUT FILE) I: 12.8
INDEX

Sysout files I: 12.8; III: 24.25
SYSOUT.EXT (Variable) I: 12.8
SYSOUTCURSOR (Variable) I: 12.8; III: 30.15
SYSOUTDATE (Variable) I: 12.13; 12.8
SYSOUTFILE (Variable) I: 12.8
SYSOUTGAG (Variable) I: 12.9
SYSPRETTYFLG (Variable) I: 11.12; II: 13.13,42; 14.8-9; III: 25.10
SYSPROPS (Variable) I: 2.5; II: 17.38
SYSTEM (in record declarations) I: 8.15
System buffer III: 30.9; 30.11
SYSTEM ERROR (Error Message) II: 14.27
System version information I: 12.11
SYSTEMFONT (Font class) III: 27.32
(SYSTEMTYPE) I: 12.13

T
T (Litatom) I: 2.3
T (Macro Type) I: 10.23
T (PRINTOUT command) III: 25.26
T (Terminal stream) III: 25.1; 25.2
T FIXED (printed by DWIM) II: 20.6
(TAB POS MINSVACES FILE) III: 25.10
.TAB POS (PRINTOUT command) III: 25.25
.TAB0 POS (PRINTOUT command) III: 25.26
*TAIL* (stack blip) I: 11.16
TAIL (Variable) II: 20.12
Tail of a list I: 3.9
(TAILP X Y) I: 3.9
(TAN X RADIANSFLG) I: 7.13
(TCOMPL FILES) II: 18.14; 18.15,18.21
(TCONC PTR X) I: 3.6; 3.7
TCP/IP III: 24.36
Teletype list structure editor II: 16.1
(TEMPLATES LITATOM1 ... LITATOMN) (File Package Command) II: 17.39
TEMPLATES (File Package Type) II: 17.24
Templates in Masterscope II: 19.18
Terminal input/output III: 30.1; 25.3
Terminal streams III: 25.1; 25.2
Terminal syntax classes III: 30.5
Terminal tables III: 30.4
(TERMTABLEP TTBL) III: 30.5
(TERPRI FILE) III: 25.9
TEST (Editor Command) II: 16.65
TEST (in Masterscope template) II: 19.19
TEST (Masterscope Relation) II: 19.8
(TESTRELATION ITEM RELATION ITEM2 INVERTED) II: 19.23
TESTRETURN (in Masterscope template) II: 19.19
(TEXTUREP OBJECT) III: 27.7
Textures III: 27.6
THEREIS FORM (I.S. Operator) I: 9.11
(THIS.PROCESS) II: 23.4
THOSE (Masterscope Set Specification) II: 19.12
(@1 THRU @2) (Editor Command) II: 16.42
(@1 THRU) (Editor Command) II: 16.42; 16.44
THRU (I.S. Operator) I: 9.22
THRU (in event specification) II: 13.7
TICKS (Timer Unit) I: 12.16
(TIME TIMEX TIMEN TIMETYP) II: 22.8
Time stamps I: 10.9; II: 16.76
Time-slice of history list II: 13.31; 13.21
TIME.ZONES (Variable) I: 12.15
(TIMEALL TIMEFORM NUMEROFTIMES TIMEWHAT
INTERPFLG —) II: 22.7
(TIMEREXPIRED? TIMER ClockValue.or.timerUnits)
I: 12.17
Timers I: 12.16
timerUnits UNITS (I.S. Operator) I: 12.18
(TIMES X1 X2 ... XN) I: 7.3
TINES (use with REDO) II: 13.8
\TimeZoneComp (Variable) I: 12.16
TITLE (Menu Field) III: 28.41
TITLE (Window Property) III: 28.33
(@1 TO @2) (Editor Command) II: 16.42
(@1 TO) (Editor Command) II: 16.42; 16.44
TO FORM (I.S. Operator) I: 9.14; 9.15
TO (in event specification) II: 13.7
TO SET (Masterscope Path Option) II: 19.16
TOO MANY ARGUMENTS (Error Message) I: 10.3;
II: 14.31
TOO MANY FILES OPEN (Error Message) II: 14.28
TOO MANY USER INTERRUPT CHARACTERS (Error
Message) II: 14.30
TOP (as argument to ADVISE) II: 15.11
**TOP** (in backtrace) II: 14.9
Top margin III: 27.11
TOTOPFN (Window Property) III: 28.20
(TOTOPW WINDOW NOCALLTOTOPFNFLG) III:
28.20
(TRACE X) II: 15.5; 14.5,17; 15.1,7
TRACEREGION (Variable) II: 14.16
TRACEWINDOW (Variable) II: 14.16
Tracing functions II: 15.1
Transcript files III: 30.12
Translations in CLISP II: 21.17

INDEX
Types in Masterscope  II: 19.13
(TYPESOF NAME POSSIBLETYPES IMPOSSIBLETYPES SOURCE)  II: 17.27
U
(U-CASE X)  I: 2.10; II: 16.52
(U-CASEP X)  I: 2.10
(UALPHORDER A B)  I: 3.18
UB (Break Command)  II: 14.6
UCASELST (Variable)  III: 26.46
(UGLYVARS VAR1 ... VARn) (File Package Command)  II: 17.36; III: 25.18
UNABLE TO DWIMIFY (Error Message)  II: 18.12
(UNADVISE X)  II: 15.12; 15.11,13
UNADVISED (Printed by System)  II: 15.9
UNARYOP (Property Name)  II: 21.28
UNBLOCK (Editor Command)  II: 16.65
UNBOUND ATOM (Error Message)  I: 2.2-3; II: 14.31
Unboxing numbers  I: 7.1
(UNBREAK X)  II: 15.7; 15.5,8; 22.9
(UNBREAK0 FN —)  II: 15.7; 15.8
(FN UNBREAKABLE) (value of BREAKIN)  II: 15.6
(UNBREAKIN FN)  II: 15.8; 15.7
UNBROKEN (Printed by ADVISE)  II: 15.11
UNBROKEN (printed by compiler)  II: 18.13
UNBROKEN (Printed by System)  II: 15.9
UNDEFINED CAR OF FORM (Error Message)  II: 14.31
UNDEFINED FUNCTION (Error Message)  II: 14.31;
20.2
UNDEFINED OR ILLEGAL GO (Error Message)  I: 9.8;
II: 14.28
UNDEFINED TAG (Error Message)  I: 10.28; II: 18.23
UNDEFINED TAG, ASSEMBLE (Error Message)  II:
18.23
UNDEFINED TAG, LAP (Error Message)  II: 18.23
Undo (DEdit Command)  II: 16.8
(UNDO EventSpec) (Editor Command)  II: 16.66
UNDO (Editor Command)  II: 16.64; 13.43
UNDO EventSpec: X1 ... XN (Prog. Asst. Command)
II: 13.14
UNDO EventSpec (Prog. Asst. Command)  II: 13.13;
13.7,28,33,42-43; 20.3
Undoing  II: 13.26; 13.44
Undoing DWIM corrections  II: 13.14; 21.20
Undoing in the editor  II: 16.64; 13.44; 16.29
Undoing out of order  II: 13.27; 13.13
(UNDOLISPX LINE)  II: 13.42
(UNDOLISPX1 EVENT FLG —)  II: 13.42
VALUE OUT OF RANGE EXPT (Error Message)  I: 7.13
VALUECOMMANDFN (Window Property) III: 26.8
(VALUEOF LINE) II: 13.19; 13.34
Variable bindings I: 11.1; 10.19; II: 17.54
Variable bindings in stack frames I: 11.6
(VARIABLES POS) I: 11.7; II: 14.10
(VARS VAR1 ... VARN) (File Package Command) II: 17.35
VARS (File Package Type) II: 17.24
VARTYPE (Property Name) II: 17.22; 17.18
VAXMACRO (Property Name) I: 10.21
VERSION (File name field) III: 24.6
Version information I: 12.11
Version recognition of files III: 24.11
VertScrollCursor (Variable) III: 30.15
VertThumbCursor (Variable) III: 30.15
Video display screens I: 12.4; III: 30.22
Video taping from the screen III: 30.23
(VIDEOCOLOR BLACKFLG) III: 30.23
(VIDEBRATE TYPE) III: 30.23
(VIRGINFN FN FLG) II: 15.8
Virtual memory I: 12.6
Virtual memory file I: 12.6; III: 24.21,23
(VMEM.PURE.STATE X) I: 12.10
(VMEMSIZE) I: 12.11
(VOLUMES) III: 24.23
(VOLUMESIZE VOLUMENAME —) III: 24.23
W
(WAITFOR.TTY MSEC NEEDWINDOW) II: 23.12
WAIITBEFORESCROLLTIME (Variable) III: 28.24
WAITBETWEENSCROLLTIME (Variable) III: 28.24
(WAITFORINPUT FILE) III: 25.6
WAITINGCURSOR (Variable) III: 30.15
(WAKE.PROCESS PROG STATUS) II: 23.5
WBorder (Variable) III: 28.14,32-33
(WBREAK ONFLG) II: 14.15
WEIGHT (Font property) III: 27.27
(WFROMDSS DISPLAYSTREAM DONTCREATE) III: 27.25
(WFROMMENUS MENU) III: 28.42
WHEN FORM (I.S. Operator) I: 9.15
WHENCLOSED (File Package Type Property) II: 17.31
WHENCLOSED FILE PROP1 VAL1 ... PROP N VAL N) III: 24.20
(WHENCOPIEDFN IMAGEOBJ TARGETWINDOWSTREAM
SOURCEHOSTSTREAM TARGETHOSTSTREAM) (IMAGEFS Method) III: 27.39
(WHENDELETEDFN IMAGEOBJ
TARGETWINDOWSTREAM) (IMAGEFS Method) III: 27.39
WHENFILED (File Package Type Property) II: 17.32
WHENHELDFN (Menu Field) III: 28.40
(WHENINSERTEDFN IMAGEOBJ
TARGETWINDOWSTREAM SOURCEHOSTSTREAM TARGETHOSTSTREAM) (IMAGEFS Method) III: 27.39
(WHENMOVEDFN IMAGEOBJ
TARGETWINDOWSTREAM SOURCEHOSTSTREAM TARGETHOSTSTREAM) (IMAGEFS Method) III: 27.39
(WHENOPERATEDONFN IMAGEOBJ
WINDOWSTREAM HOWOPERATEDON SELECTION HOSTSTREAM) (IMAGEFS Method) III: 27.39
WHENSELectedFN (Menu Field) III: 28.40
WHENUNFILED (File Package Type Property) II: 17.32
WHENUNHELDFN (Menu Field) III: 28.40
WHERE (I.S. Operator) I: 9.22
WHEREATTACHED (Window Property) III: 28.54
(WHEREIS NAME TYPE FILES FN) II: 17.14
(WHICH X Y) III: 28.32
WHILE FORM (I.S. Operator) I: 9.16
WHILE (use with REDO) II: 13.8
WHITISHADE (Variable) III: 27.7
WHOLEDISPLAY (Variable) III: 27.27
WHITESHARE (Variable) III: 27.7
&W (DEFMACRO keyword) I: 10.27
WHOLEDISPLAY (Variable) III: 30.22; 27.2
(WIDEPAPER FLG) III: 26.48
WIDTH (Window Property) III: 28.34
(WIDTHIFWINDOW INTERIORWIDTH BORDER) III: 28.32
WINDOW (Process Property) II: 23.3
Window command menu III: 28.3
Window has no REPINTFN. Can’t redisplay.
(printed in prompt window) III: 28.16
Window menu III: 28.3
Window properties III: 28.13
Window system III: 28.2; 28.1
(WINDOWADDPROP WINDOW PROP ITEMTOADD FIRSTFLG) III: 28.13
WINDOWBACKGROUNDSHADE (Variable) III: 28.23
(WINDOWDELETEPROP WINDOW PROP ITEMDELETE) III: 28.13
INDEX

(← PATTERN) (Editor Command) II: 16.25
← (Editor Command) II: 16.25; 16.27
← (in event address) II: 13.6
← (in pattern matching) I: 12.28
← (in record declarations) I: 8.14
← (Printed by System) II: 14.2
↔ (Editor Command) II: 16.28

( ) (backquote) (Read Macro) III: 25.42

| (change character) II: 16.30; III: 26.49
| (Read Macro) I: 7.4; III: 25.43

- (CLISP Operator) II: 21.11
- (in pattern matching) I: 12.27

! (in Masterscope template) II: 19.20
! (in PA commands) II: 13.9
! (in pattern matching) I: 12.27-28
! (use with <,> in CLISP) II: 21.10
!! (use with <,> in CLISP) II: 21.10
!0 (Editor Command) II: 16.15
!E (Editor Command) II: 16.55; 13.43
!EVAL (Break Command) II: 14.6
!EVAL (Break Window Command) II: 14.3
!F (Editor Command) II: 16.55; 13.43
!GO (Break Command) II: 14.6
!H (Editor Command) II: 16.55; 13.43
!N (Editor Command) II: 16.16; 16.17
!OK (Break Command) II: 14.6
!Undo (DEdit Command) II: 16.8
!UNDO (Editor Command) II: 16.64
!VALUE (Variable) II: 14.5; 14.16; 15.9-10

" " (string delimiter) I: 4.1; III: 25.3-4
"" (use in ASKUSER) III: 26.20
"<c.r.>" (in history commands) II: 13.32

#N (N a number) (in pattern matching) I: 12.29
# FORM (PRINTOUT command) III: 25.30
(# # COM1 COM2... COMN) II: 16.59; 16.24
### (in INSERT, REPLACE, and CHANGE commands) II: 16.34

## (Printed by System) III: 30.10
#CAREFULCOLUMNS (Variable) III: 26.47
#RPARS (Variable) III: 26.47
#SPELLINGS1 (Variable) II: 20.18
#SPELLINGS2 (Variable) II: 20.18
#SPELLINGS3 (Variable) II: 20.18
#UNDOSAVES (Variable) II: 13.41; 13.30
#USERWORDS (Variable) II: 20.18

\$
$ X FOR Y IN EventSpec (Prog. Asst. Command) II: 13.11
$ Y -> X IN EventSpec (Prog. Asst. Command) II: 13.11
$ Y TO X IN EventSpec (Prog. Asst. Command) II: 13.11
$ Y = X IN EventSpec (Prog. Asst. Command) II: 13.11
$ Y IN EventSpec (Prog. Asst. Command) II: 13.11
$(dollar) (in pattern matching) I: 12.27
$(escape) (in CLISP) II: 21.10-11
$(escape) (in Edit Pattern) II: 16.18
$(escape) (in Editor) II: 16.45-46
$(escape) (in spelling correction) II: 20.15; 20.22
$(escape) (Prog. Asst. Command) II: 13.11
$(escape) (use in ASKUSER) III: 26.19
$$ (escape, escape) (in Edit Pattern) II: 16.18
$$ (escape, escape) (use in ASKUSER) III: 26.20
$$EXTREME (Variable) I: 9.12
$$VAL (Variable) I: 9.12; 9.19
$1 (in pattern matching) I: 12.19
$GO (escape-GO) (TYPE-AHEAD command) II: 13.18
$Q (escape-Q) (TYPE-AHEAD command) II: 13.18
$STOP (escape-STOP) (TYPE-AHEAD command) II: 13.18

% % I: 2.1; 4.1; III: 25.3; 25.4; 38; 30.11
% (use in comments) III: 26.46
% (use in comments) III: 26.46

& & (in Edit Pattern) II: 16.18
& (in MBD command) II: 16.36-37
& (in pattern matching) I: 12.26
& (Printed by System) III: 25.12
& (use in ASKUSER) III: 26.19
INDEX

&ALLOW-OTHER-KEYS (DEFMACRO keyword) I: 10.26

&AUX (DEFMACRO keyword) I: 10.26

&BODY (DEFMACRO keyword) I: 10.25

&KEY (DEFMACRO keyword) I: 10.25

&OPTIONAL (DEFMACRO keyword) I: 10.25

&REST (DEFMACRO keyword) I: 10.25

&Undo (DEdit Command) II: 16.8

&WHOLE (DEFMACRO keyword) I: 10.27

' (CLISP Operator) II: 21.11

'(in DWIM) II: 20.8

'(in pattern matching) I: 12.26

'LST (Masterscope Set Specification) II: 19.11

'ATOM (Masterscope Set Specification) II: 19.10

'(Read macro) I: 10.12; III: 25.42

( (in DEdit Command) II: 16.7

(out (DEdit Command) II: 16.8

()) I: 3.3

() (DEdit Command) II: 16.7

() out (DEdit Command) II: 16.7

)

) in (DEdit Command) II: 16.7

) out (DEdit Command) II: 16.8

*

* (as a prettyprint macro) III: 26.44

* (as a read macro) III: 26.44

* (CLISP Operator) II: 21.7

(*.* X) (Editor Command) II: 16.56

(*. TEXT) (File Package Command) II: 17.40

* (Function) III: 26.42

* (in File Group) III: 24.33

* (in file package command) II: 17.44

* (in pattern matching) I: 12.26

* (use in comments) III: 26.42; 26.43

*** note: FILENAME dated DATE isn't current version; FILENAME dated DATE is. (printed by EDITLOADFNS?) II: 16.74

***** (in compiler error messages) II: 18.22

**BREAK** (in backtrace) II: 14.9

**COMMENT** (printed by editor) II: 16.48

**COMMENT** (printed by system) III: 26.43

**COMMENT** (Variable) I: 12.3; II: 16.48; III: 26.43

**DEALLOC** (data type name) I: 8.21; II: 22.4

**EDITOR** (in backtrace) II: 14.9

**TOP** (in backtrace) II: 14.9

*ANY* (in edit pattern) II: 16.18

*ARCHIVE* (History list property) II: 13.33; 13.16

*ARGN* (Stack blip) I: 11.15

*ARGVAL* (Stack blip) II: 11.16

*CONTEXT* (History list property) II: 13.33

*ERROR* (History list property) II: 13.33

*FN* (Stack blip) I: 11.16

*FORM* (Stack blip) II: 11.16

*GROUP* (History list property) II: 13.33

*HISTORY* (History list property) II: 13.33

*LISPXPRINT* (History list property) II: 13.33

*PRINT* (History list property) II: 13.33

*TAIL* (Stack blip) I: 11.16

+(

+ (CLISP Operator) II: 21.7

.

. (PRINTOUT command) III: 25.26

" (PRINTOUT command) III: 25.26

... (PRINTOUT command) III: 25.26

-

- (CLISP Operator) II: 21.7

-- (in Edit Pattern) II: 16.19

-- (in pattern matching) I: 12.27

-> EXPR (Break Command) II: 14.11

-> (in pattern matching) I: 12.30

-> (printed by DWIM) II: 20.4; 20.2-3,6

-> (printed by editor) II: 16.46

.

. (CLISP Operator) II: 21.9

. (in a floating point number) I: 7.11

. (in a list) I: 3.3

. (in Masterscope) II: 19.2

. (in pattern matching) I: 12.28

. (printed by Masterscope) II: 19.2

PATTERN .. @ (Editor Command) II: 16.27

.. (in Edit Pattern) II: 16.19

.. TEMPLATE (in Masterscope template) II: 19.20

.. (in Edit Pattern) II: 16.19-20

.. (printed by DWIM) II: 20.3,5

.. (Printed by Editor) II: 16.14

.. (printed during input) II: 13.37; 13.5
INDEX

... VARS (Prog. Asst. Command) II: 13.10; 13.33
...ARGS (history list property) II: 13.33
.BASE (PRINTOUT command) III: 25.27
.CENTER POS EXPR (PRINTOUT command) III: 25.29
.CENTER2 POS EXPR (PRINTOUT command) III:
25.29
.FFORMAT NUMBER (PRINTOUT command) III:
25.30
.FONT FONTSPEC (PRINTOUT command) III: 25.27
.FR POS EXPR (PRINTOUT command) III: 25.29
.FR2 POS EXPR (PRINTOUT command) III: 25.29
.IFORMAT NUMBER (PRINTOUT command) III:
25.30
.N FORMAT NUMBER (PRINTOUT command) III:
25.30
.P2 THING (PRINTOUT command) III: 25.28
.PAGE (PRINTOUT command) III: 25.26
.PARA L MARG R MARG LIST (PRINTOUT command)
III: 25.28
.PARA2 L MARG R MARG LIST (PRINTOUT command)
III: 25.28
.PPF THING (PRINTOUT command) III: 25.28
.PPF TL THING (PRINTOUT command) III: 25.28
.PPV THING (PRINTOUT command) III: 25.28
.PPV TL THING (PRINTOUT command) III: 25.28
.SKIP LINES (PRINTOUT command) III: 25.26
.SP DISTANCE (PRINTOUT command) III: 25.26
.SUB (PRINTOUT command) III: 25.27
.SUP (PRINTOUT command) III: 25.27
.TAB POS (PRINTOUT command) III: 25.25
.TAB0 POS (PRINTOUT command) III: 25.26

/ (CLISP Operator) II: 21.7
/ (use with @ break command) II: 14.7
/ functions II: 13.26; 13.27,41
/FNS (Variable) II: 13.26
/MAPCON (Function) II: 21.13
/MAPCONC (Function) II: 21.13
/NCONC (Function) II: 21.13
/NCONC1 (Function) II: 21.13
/REPLACE (Record Operator) I: 8.3
/RPLACA (Function) II: 21.13
/RPLACD (Function) II: 21.13
/RPLNODE (Function) II: 13.40
/RPLNODE2 (Function) II: 13.40
0 (Editor Command) II: 16.15

0 (instead of right parenthesis) II: 20.5; 20.1,8,10
1
10MACRO (Property Name) I: 10.21
2
(2ND . @) (Editor Command) II: 16.24
3
32MBADDRESSABLE (Function) II: 22.5
(3ND . @) (Editor Command) II: 16.25
7
7 (instead of ') II: 20.9
8
8 (instead of left parenthesis) II: 16.67
8044 (Printer type) III: 29.5
9
9 (instead of left parenthesis) II: 20.5; 20.1,8,10-11
:
: (CLISP Operator) II: 21.9
(: E1 ... EM) (Editor Command) II: 16.32
(:) (Editor Command) II: 16.32
:: (Printed by System) II: 14.1
::: (CLISP Operator) II: 21.9
;
; FORM (Prog. Asst. Command) II: 13.18
<
< (CLISP Operator) II: 21.10
<,> (use in CLISP) II: 21.10
=
= FORM (Break Command) II: 14.10
= (CLISP Operator) II: 21.8
= (in event address) II: 13.6
= (in pattern matching) I: 12.26
= (printed by DWIM) II: 20.5
= (use with @ break command) II: 14.7
= = (in Edit Pattern) II: 16.19
= = (in pattern matching) I: 12.26
> (in pattern matching) I: 12.30
E (Printed by Editor) II: 16.67
>
> (CLISP Operator) II: 21.10
\?
\? (Editor Command) II: 16.48
\? (Litatom) I: 3.11
\? (printed by DWIM) II: 20.4-5
\? (printed by Masterscope) II: 19.18
\? (Read Macro) II: 14.8; III: 25.43
\? = (Break Command) II: 14.7
\? = (Break Window Command) II: 14.3
\? = (Editor Command) II: 16.48
\? = (in TTYIN) III: 26.33
?? EventSpec (Prog. Asst. Command) II: 13.13;
13.33
?ACTIVATEFLG (Variable) III: 26.36; 26.23
?Undo (DEdit Command) II: 16.8
@?
@ (Break Command) II: 14.6; 14.12
@ (in event specification) II: 13.39
(@ EXPRFORM TEMPLATEFORM) (in Masterscope template) II: 19.21
@ (in pattern matching) I: 12.26, 28
@ (location specification in editor) II: 16.24
@ PREDICATE (Masterscope Set Specification) II:
19.11
@ (use with @ break command) II: 14.7
@@ (in event specification) II: 13.8; 13.16, 39