

EXPLORER WINDOW SYSTEM REFERENCE

MANUAL REVISION HISTORY

Explorer™ Window System Reference (2243200-0001 *B)

Original Issue June 1985

Revision A June 1987

Revision B December 1987

© 1987, Texas Instruments Incorporated. All Rights Reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Texas Instruments Incorporated.

The system-defined windows shown in this manual are examples of the software as this manual goes into production. Later changes in the software may cause the windows on your system to be different from those in the manual.

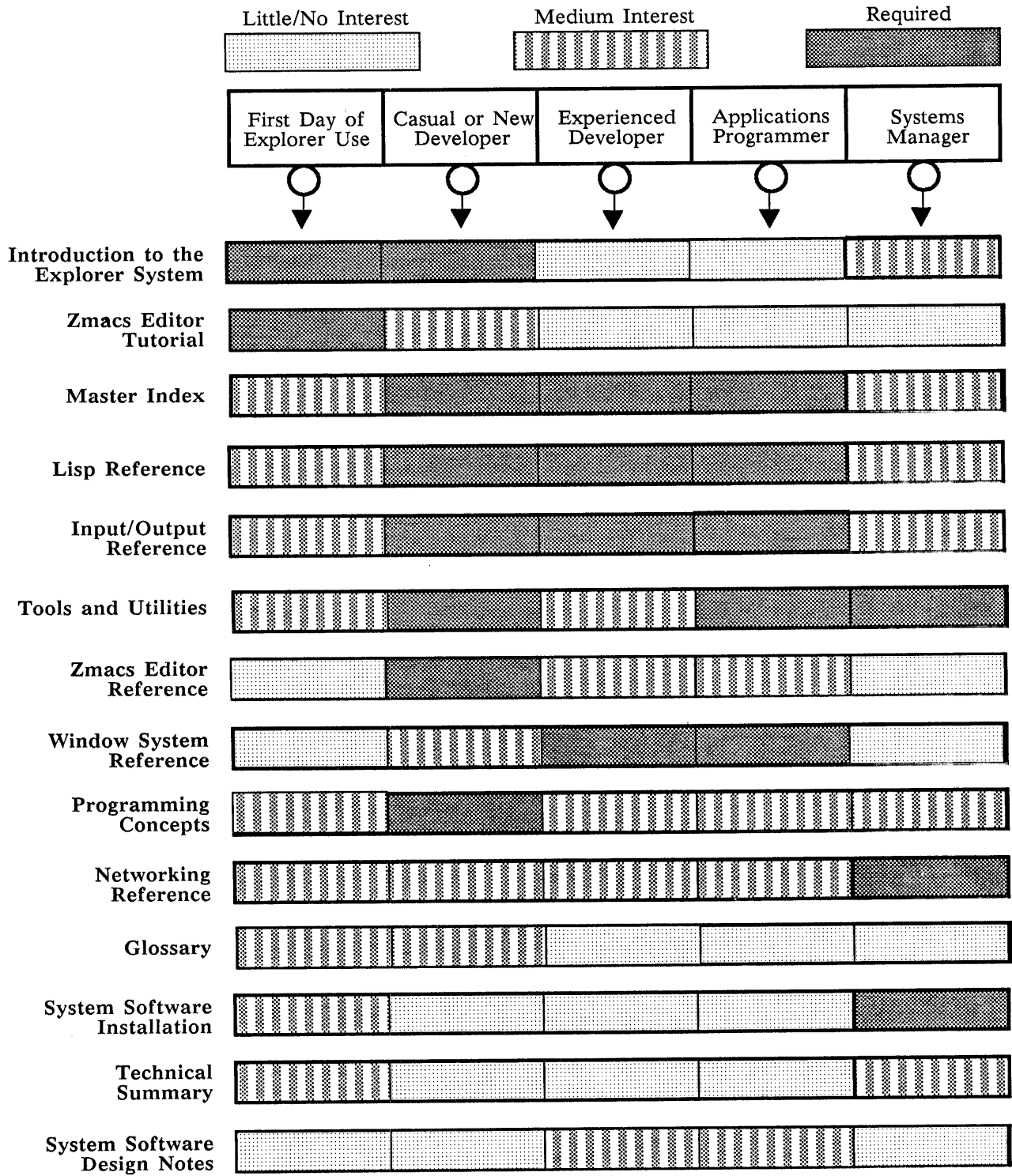
RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

Texas Instruments Incorporated
ATTN: Data Systems Group, M/S 2151
P.O. Box 2909
Austin, Texas 78769-2909

Produced by the Publishing Center
Texas Instruments Incorporated
Data Systems Group
Austin, Texas

THE EXPLORER™ SYSTEM SOFTWARE MANUALS



THE EXPLORER™ SYSTEM SOFTWARE MANUALS

Mastering the Explorer Environment	Explorer Technical Summary	2243189-0001
	Introduction to the Explorer System	2243190-0001
	Explorer Zmacs Editor Tutorial	2243191-0001
	Explorer Glossary	2243134-0001
	Explorer Networking Reference	2243206-0001
	Explorer Diagnostics	2533554-0001
	Explorer Master Index to Software Manuals	2243198-0001
Explorer System Software Installation Guide	2243205-0001	

Programming With the Explorer	Explorer Programming Concepts	2549830-0001
	Explorer Lisp Reference	2243201-0001
	Explorer Input/Output Reference	2549281-0001
	Explorer Zmacs Editor Reference	2243192-0001
	Explorer Tools and Utilities	2549831-0001
	Explorer Window System Reference	2243200-0001

Explorer Options	Explorer Natural Language Menu System User's Guide	2243202-0001
	Explorer Relational Table Management System User's Guide	2243203-0001
	Explorer Grasper User's Guide	2243135-0001
	Explorer TI Prolog User's Guide	2537248-0001
	Programming in Prolog, by Clocksin and Mellish	2249985-0001
	Explorer Color Graphics User's Guide	2537157-0001
	Explorer TCP/IP User's Guide	2537150-0001
	Explorer LX™ User's Guide	2537225-0001
	Explorer LX System Installation	2537227-0001
	Explorer NFS™ User's Guide	2546890-0001
	Explorer DECnet™ User's Guide	2537223-0001
Personal Consultant™ Plus Explorer	2537259-0001	

System Software Internals	Explorer System Software Design Notes	2243208-0001
	Release Information, Explorer System Software	2549844-0001

Explorer and NuBus are trademarks of Texas Instruments Incorporated.
Explorer LX is a trademark of Texas Instruments Incorporated.
NFS is a trademark of Sun Microsystems, Inc.
DECnet is a trademark of Digital Equipment Corporation.
Personal Consultant is a trademark of Texas Instruments Incorporated.

THE EXPLORER™ SYSTEM HARDWARE MANUALS

System Level Publications	Explorer 7-Slot System Installation 2243140-0001 Explorer System Field Maintenance 2243141-0001 Explorer System Field Maintenance Documentation Kit 2243222-0001 Explorer System Field Maintenance Supplement 2537183-0001 Explorer System Field Maintenance Supplement Documentation Kit 2549278-0001 Explorer NuBus™ System Architecture General Description 2537171-0001
----------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

System Enclosure Equipment Publications	Explorer 7-Slot System Enclosure General Description 2243143-0001 Explorer Memory General Description (8-megabytes) 2533592-0001 Explorer 32-Megabyte Memory General Description 2537185-0001 Explorer Processor General Description 2243144-0001 68020-Based Processor General Description 2537240-0001 Explorer II Processor and Auxiliary Processor Options General Description 2537187-0001 Explorer System Interface General Description 2243145-0001 Explorer Color System Interface Board General Description 2537189-0001 Explorer NuBus Peripheral Interface General Description (NUPI board) 2243146-0001
------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Display Terminal Publications	Explorer Display Unit General Description 2243151-0001 CRT Data Display Service Manual, Panasonic code number FTD85055057C 2537139-0001 Explorer Color Console General Description 2537195-0001 TRINITRON® Graphic Display Monitor GDM-1603 Service Manual, Sony® part number 0-558-986-01 2551107-0001 Model 924 Video Display Terminal User's Guide 2544365-0001
--------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

143-Megabyte Disk/Tape Enclosure Publications	Explorer Mass Storage Enclosure General Description 2243148-0001 Explorer Winchester Disk Formatter (ADAPTEC) Supplement to Explorer Mass Storage Enclosure General Description 2243149-0001 Explorer Winchester Disk Drive (Maxtor) Supplement to Explorer Mass Storage Enclosure General Description 2243150-0001 Explorer Cartridge Tape Drive (Cipher) Supplement to Explorer Mass Storage Enclosure General Description 2243166-0001 Explorer Cable Interconnect Board (2236120-0001) Supplement to Explorer Mass Storage Enclosure General Description 2243177-0001
------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

143-Megabyte Disk Drive Vendor Publications	XT-1000 Service Manual, 5 1/4-inch Fixed Disk Drive, Maxtor Corporation, part number 20005 (5 1/4-inch Winchester disk drive, 112 megabytes) 2249999-0001 ACB-5500 Winchester Disk Controller User's Manual, Adaptec, Inc., (formatter for the 5 1/4-inch Winchester disk drive) 2249933-0001
------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1/4-Inch Tape Drive Vendor Publications	Series 540 Cartridge Tape Drive Product Description, Cipher Data Products, Inc., Bulletin Number 01-311-0284-1K (1/4-inch tape drive) 2249997-0001 MT01 Tape Controller Technical Manual, Emulex Corporation, part number MT0151001 (formatter for the 1/4-inch tape drive) 2243182-0001 Viper™ Half-High Intelligent 4 1/4-Inch Streaming Cartridge Tape Drive SCSI Models 2060S and 2125S, Archive Corporation, part number 21136-001 2551106-0001
----------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

182-Megabyte Disk/Tape Enclosure MSU II Publications	Mass Storage Unit (MSU II) General Description 2537197-0001
---------------------------------------------------------------------	----------------------------------------------------------------------

182-Megabyte Disk Drive Vendor Publications	Control Data® WREN™ III Disk Drive OEM Manual, part number 77738216, Magnetic Peripherals, Inc., a Control Data Company 2546867-0001
------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------

515-Megabyte Mass Storage Subsystem Publications	SMD/515-Megabyte Mass Storage Subsystem General Description (includes SMD/SCSI controller and 515-megabyte disk drive enclosure) 2537244-0001
-----------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------

515-Megabyte Disk Drive Vendor Publications	515-Megabyte Disk Drive Documentation Master Kit (Volumes 1, 2, and 3), Control Data Corporation 2246129-0002 Volume 1, General Description, Operation, Installation and Checkout, and Part Data 2246125-0004 Volume 2, Theory, General Maintenance, Trouble Analysis, Electrical Checks, and Repair Information 2246125-0005 Volume 3, Diagrams 2246125-0006
------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1/2-Inch Tape Drive Publications	MT3201 1/2-Inch Tape Drive General Description 2537246-0001
---------------------------------------------	----------------------------------------------------------------------

Viper is a trademark of Archive Corporation.
 Control Data is a registered trademark of Control Data Corporation.
 WREN is a trademark of Control Data Corporation.

1/2-Inch Tape Drive Vendor Publications	Cipher CacheTape® Documentation Manual Kit (Volumes 1 and 2 With SCSI Addendum and, Logic Diagram), Cipher Data products	2246130-0001
	1/2-Inch Tape Drive Operation and Maintenance (Volume 1), Cipher Data Products	2246126-0001
	1/2-Inch Tape Drive Theory of Operation (Volume 2), Cipher Data Products	2246126-0002
	SCSI Addendum With Logic Diagram, Cipher Data Products	2246126-0003

Printer Publications	Model 810 Printer Installation and Operation Manual	2311356-9701
	Omni 800™ Electronic Data Terminals Maintenance Manual for Model 810 Printers	0994386-9701
	Model 850 RO Printer User's Manual	2219890-0001
	Model 850 RO Printer Maintenance Manual	2219896-0001
	Model 850 XL Printer User's Manual	2243250-0001
	Model 850 XL Printer Quick Reference Guide	2243249-0001
	Model 855 Printer Operator's Manual	2225911-0001
	Model 855 Printer Technical Reference Manual	2232822-0001
	Model 855 Printer Maintenance Manual	2225914-0001
	Model 860 XL Printer User's Manual	2239401-0001
	Model 860 XL Printer Maintenance Manual	2239427-0001
	Model 860 XI Printer Quick Reference Guide	2239402-0001
	Model 860/859 Printer Technical Reference Manual	2239407-0001
	Model 865 Printer Operator's Manual	2239405-0001
	Model 865 Printer Maintenance Manual	2239428-0001
	Model 880 Printer User's Manual	2222627-0001
	Model 880 Printer Maintenance Manual	2222628-0001
	OmniLaser™ 2015 Page Printer Operator's Manual	2539178-0001
	OmniLaser 2015 Page Printer Technical Reference	2539179-0001
	OmniLaser 2015 Page Printer Maintenance Manual	2539180-0001
	OmniLaser 2108 Page Printer Operator's Manual	2546348-0001
	OmniLaser 2108 Page Printer Technical Reference	2546349-0001
	OmniLaser 2108 Page Printer Maintenance Manual	2546350-0001
OmniLaser 2115 Page Printer Operator's Manual	2546344-0001	
OmniLaser 2115 Page Printer Technical Reference	2546345-0001	
OmniLaser 2115 Page Printer Maintenance Manual	2546346-0001	

Communications Publications	990 Family Communications Systems Field Reference	2276579-9701
	EI990 Ethernet® Interface Installation and Operation	2234392-9701
	Explorer NuBus Ethernet Controller General Description	2243161-0001
	Communications Carrier Board and Options General Description	2537242-0001

CacheTape is a registered trademark of Cipher Data Products, Inc.
Omni 800 is a trademark of Texas Instruments Incorporated.
OmniLaser is a trademark of Texas Instruments Incorporated.
Ethernet is a registered trademark of Xerox Corporation.

CONTENTS

Section	Title
	About This Manual
1	Window System Concepts
2	Basic Windows
3	Outside Edges of Windows
4	Sizes and Positions
5	Visibility and Exposure
6	Selection
7	Output of Text
8	Input
9	Fonts
10	Blinkers
11	The Mouse
12	Graphics
13	Typeout Windows
14	Choice Facilities
15	Frames
16	Text Scroll Windows
17	General Scroll Windows
18	Miscellaneous Features
19	Using Color
Appendix A	Obsolete Symbols
Appendix B	Converting Applications to Color

Section	Paragraph	Title	Page
---------	-----------	-------	------

About This Manual

Introduction	xxiii
Assumptions	xxiii
Contents of This Manual	xxiii
Suggestions for Using This Manual	xxiv
User Comments	xxiv
Notational Conventions	xxv
Keystroke Sequences	xxv
Mouse Clicks	xxv
Lisp Language Notation	xxvi
Flavor Notation	xxvi
Flavor Naming Conventions	xxviii
Initialization Options, Methods, and Instance Variables	xviii

1

Window System Concepts

1.1	General Concepts	1-1
1.1.1	Screens, Windows, and Panes	1-1
1.1.1.1	The Who-Line Screen	1-3
1.1.1.2	The Main Screen	1-4
1.1.2	The Window System as a User Interface	1-4
1.1.3	States of a Window	1-4
1.2	Windows as Instances of Flavors	1-5
1.2.1	Characteristics of Window Flavors	1-5
1.2.2	Mixing Flavors	1-6
1.2.3	Methods and Their Flavors	1-6
1.3	Features Common to All Windows	1-7
1.3.1	Outside Edges of Windows	1-7
1.3.2	Sizes and Positions	1-7
1.3.3	Visibility and Exposure	1-7
1.3.4	Selection	1-7
1.3.5	Output of Text	1-8
1.3.6	Input	1-8
1.3.7	Fonts	1-8
1.3.8	Blinkers	1-8
1.3.9	The Mouse	1-9
1.3.10	Graphics	1-9
1.3.11	Typeout Windows	1-9
1.4	Types of Windows	1-9
1.4.1	Choice Facilities	1-9
1.4.2	Frames	1-10
1.4.3	Text Scroll Windows	1-10
1.4.4	General Scroll Windows	1-11
1.4.5	Miscellaneous Features	1-11
1.4.5	Optional Color	1-11
1.5	Obsolete Symbols	1-11

Section	Paragraph	Title	Page
	1.6	Designing a Window	1-12
	1.7	General Choices Among Windows	1-13
	1.7.1	For Any Window	1-15
	1.7.1.1	Before You Begin	1-15
	1.7.1.2	After You Finish	1-15
	1.7.2	Using a Frame	1-16
	1.7.2.1	Questions About the Frame as a Whole	1-16
	1.7.2.2	Questions About a Non-Constraint Frame as a Whole	1-16
	1.7.3	Types of Output Windows	1-17
	1.7.4	Types of Input Windows	1-18
	1.7.4.1	Obtaining Confirmation	1-18
	1.7.4.2	Gathering Information	1-19
	1.7.5	Types of Standalone Windows	1-21
	1.7.6	Types of Mixins	1-22
<hr/>			
2		Basic Windows	
	2.1	Introduction	2-1
	2.2	Window System Packages	2-1
	2.3	Creation of Windows	2-2
	2.4	Basic Window Flavors	2-3
<hr/>			
3		Outside Edges of Windows	
	3.1	Introduction	3-1
	3.2	Margins	3-2
	3.3	Borders	3-2
	3.3.1	Border Functions	3-4
	3.3.2	Deleting Borders on Full-Screen Windows	3-4
	3.4	Labels	3-5
	3.4.1	Names of Windows	3-5
	3.4.2	The <code>w:label-mixin</code> Flavor	3-5
	3.4.3	Positioning the Label	3-7
	3.4.4	Boxing the Label	3-7
	3.4.5	Delaying Redisplay of a Label	3-9
	3.5	Margin Regions	3-9
	3.5.1	About <code>w:margin-region-function</code>	3-11
	3.5.2	Defining Margin Item Flavors	3-12
<hr/>			
4		Sizes and Positions	
	4.1	Introduction	4-1
	4.2	Initialization Options for Sizes and Positions	4-1
	4.3	Methods for Sizes and Positions	4-4
	4.3.1	The <i>option</i> Argument	4-4
	4.3.2	The Methods	4-4
	4.4	Low-Level Edges Functions	4-6

Section	Paragraph	Title	Page
---------	-----------	-------	------

5		Visibility and Exposure	
	5.1	Introduction	5-1
	5.2	Screens	5-2
	5.3	Hierarchy of Windows	5-3
	5.4	Lists of Windows	5-5
	5.5	Pixels	5-7
	5.6	Bit-Save Arrays	5-9
	5.7	Screen Arrays and Exposure	5-11
	5.7.1	Concepts of Screen Arrays	5-11
	5.7.2	Concepts of Exposure	5-12
	5.7.3	Symbols That Manipulate Screen Arrays and Exposure	5-13
	5.8	Temporary Windows	5-16
	5.8.1	Flavors and Methods	5-17
	5.8.2	Temp Locking	5-18
	5.9	The Screen Manager	5-18
	5.9.1	Autoexposure	5-19
	5.9.2	Autoselection	5-19
	5.9.3	Control of Partial Visibility	5-20
	5.9.4	Priority Among Windows for Exposure	5-21
	5.9.5	Negative Priorities	5-22
	5.9.6	Delaying Screen Management	5-22

6		Selection	
	6.1	Introduction	6-1
	6.2	How Programs Select Windows	6-2
	6.3	Teams of Windows	6-4
	6.3.1	The System Menu Select Command	6-5
	6.3.2	Selection With TERM and SYSTEM Keys	6-6
	6.4	Selection Substitutes	6-8
	6.4.1	Typeout Windows and Selection Substitutes	6-9
	6.4.2	Nonhierarchical Selection Substitutes	6-10
	6.5	The Status of a Window	6-10
	6.6	Windows and Processes	6-11
	6.6.1	The Inspector Example	6-11
	6.6.2	Process-Related Methods and Flavors	6-12
	6.6.3	Associating a Process With a Window	6-13
	6.6.4	Handling a Long-Running Process	6-14

7		Output of Text	
	7.1	Introduction	7-1
	7.2	How a Character Is Displayed	7-3
	7.3	Stream Output	7-5
	7.4	Output Exceptions	7-9
	7.4.1	Deexposed Typeout Actions	7-9
	7.4.2	Output-Hold and End-of-Page Exceptions	7-11
	7.4.3	**MORE** Exceptions	7-12
	7.4.4	End-of-Line Exceptions	7-14

Section	Paragraph	Title	Page
	7.5	Cursor Motion	7-15
	7.5.1	Cursor Position for Stream Operations	7-15
	7.5.2	Cursor Position Relative to Outside Coordinates	7-17
	7.6	Erasing	7-17
	7.7	Inserting and Deleting Characters and Lines	7-19
	7.8	Anticipating the Effect of Output	7-20
	7.9	Explicit (Noncursor) Output	7-23
	7.10	Window Parameters Affecting Text Output	7-25

8

Input

	8.1	Introduction	8-1
	8.2	Input Buffers	8-2
	8.3	Blips	8-3
	8.4	Input Editor	8-4
	8.4.1	How the Input Editor Works	8-4
	8.4.2	Common Input Editors	8-5
	8.4.3	The <code>w:rubout-handler</code> Variable	8-5
	8.4.4	Functional Interface to an Input Editor	8-6
	8.4.5	A Sample Input Editor Function	8-7
	8.5	Stream Input Operations	8-8
	8.5.1	Common Lisp-Compatible Read Functions and Methods	8-8
	8.5.2	Methods for Stream Operations	8-10
	8.6	I/O Buffers	8-13
	8.6.1	I/O Buffers and Type-Ahead	8-16
	8.6.2	I/O Buffers as Input Buffers	8-16
	8.7	Intercepted Characters	8-18
	8.7.1	Synchronously Intercepted Characters	8-18
	8.7.2	Asynchronously Intercepted Characters	8-20
	8.7.3	Global Asynchronous Characters	8-22
	8.8	Querying the Keyboard Explicitly	8-25
	8.9	Keyboard Parameters	8-26

9

Fonts

	9.1	Introduction	9-1
	9.2	Specifying Fonts	9-1
	9.3	Font Purposes	9-2
	9.4	Flavors and Methods	9-3
	9.5	Font Specifiers	9-4
	9.6	Attributes of Fonts	9-6
	9.7	Displaying Fonts	9-8
	9.8	Format of Fonts	9-9

10

Blinkers

	10.1	Types of Blinkers	10-1
	10.2	Visibility and Deselected Visibility of Blinkers	10-2
	10.3	Blinker Position	10-4

Section	Paragraph	Title	Page
	10.4	List of Blinkers	10-6
	10.5	Blinker Size	10-7
	10.6	Rectangular and Character Blinkers	10-7

11 The Mouse

11.1	Using the Mouse	11-1
11.2	Mouse Variables and Functions	11-2
11.3	Mouse Parameters	11-3
11.4	Mouse Clicks	11-4
11.4.1	Button Masks	11-4
11.4.2	Encoding Mouse Clicks as Characters	11-6
11.5	Ownership of the Mouse	11-7
11.5.1	Grabbing the Mouse	11-8
11.5.2	Usurping the Mouse	11-11
11.6	How Windows Handle the Mouse	11-12
11.7	Mouse Blinkers	11-16
11.7.1	Variables and Functions	11-17
11.7.2	Flavors for Mouse Blinkers	11-17
11.7.3	Reusable Mouse Blinker Types	11-18
11.7.4	Mapping Mouse Characters	11-19
11.7.5	Standard Values for Mouse Characters	11-21
11.7.6	Creating a New Glyph	11-23
11.8	Mouse Scrolling	11-24
11.8.1	Scroll Bars	11-25
11.8.2	Scrolling Protocol	11-29
11.8.3	Methods Retained for Compatibility	11-29

12 Graphics

12.1	Introduction	12-1
12.2	ALU Arguments	12-2
12.2.1	General ALU Arguments	12-3
12.2.2	Available Monochrome ALU Arguments	12-4
12.3	Windows and Worlds	12-5
12.4	Methods and Flavors to Draw Graphics Images	12-8
12.4.1	Methods That Use <code>w:graphics-mixin</code>	12-9
12.4.1.1	Smallest Size	12-9
12.4.1.2	Picture Lists	12-10
12.4.1.3	Methods That Draw Graphics	12-10
12.4.2	Bit Block Transferring	12-25
12.5	Drawing Graphic Images Using Subprimitives	12-30
12.5.1	Preparing the Sheet	12-32
12.5.2	Defining the Clipping Rectangle for Drawing	12-32
12.5.3	Subprimitives for Drawing	12-33
12.6	Using Graphic Objects	12-34
12.6.1	Windows With Graphic Object Capabilities	12-34
12.6.2	Graphic Objects	12-35
12.6.3	Graphic Characters	12-35
12.6.4	Graphic Cursors	12-35
12.6.5	Example	12-36

Section	Paragraph	Title	Page
	12.7	Functions Used With Graphic Objects	12-36
	12.7.1	Functions That Return Distances	12-36
	12.7.2	Functions That Verify Position	12-37
	12.7.3	Functions That Perform Transformations	12-38
	12.7.4	Miscellaneous Functions	12-39
	12.8	General Flavors Used With Graphic Objects	12-40
	12.8.1	World Flavor	12-40
	12.8.1.1	Parameters	12-40
	12.8.1.2	Display Lists	12-41
	12.8.1.3	Entities	12-42
	12.8.1.4	Extents	12-44
	12.8.2	Graphics Window Flavors	12-44
	12.8.3	Cache Window Flavor	12-46
	12.8.4	Transform Mixin	12-47
	12.8.5	Mouse Handler Mixin	12-48
	12.8.6	Basic Cursor Mixin	12-50
	12.8.7	Cursor Flavor	12-51
	12.8.8	Bitblt Blinker Flavor	12-51
	12.8.9	Block Cursor Flavor	12-52
	12.8.10	Sprite Cursor Flavor	12-52
	12.9	Standard Operations on Graphic Objects	12-54
	12.10	Basic Graphics Mixin	12-55
	12.11	Simple Graphic Objects	12-57
	12.11.1	Arc Flavor	12-57
	12.11.1.1	Creating the Object	12-58
	12.11.1.2	Manipulating the Object	12-59
	12.11.2	Circle Flavor	12-59
	12.11.2.1	Creating the Object	12-59
	12.11.2.2	Manipulating the Object	12-60
	12.11.3	Line Flavor	12-60
	12.11.3.1	Creating the Object	12-60
	12.11.3.2	Manipulating the Object	12-61
	12.11.4	Polyline Flavor	12-61
	12.11.4.1	Creating the Object	12-61
	12.11.4.2	Manipulating the Object	12-62
	12.11.5	Rectangle Flavor	12-62
	12.11.5.1	Creating the Object	12-62
	12.11.5.2	Manipulating the Object	12-63
	12.11.6	Spline Flavor	12-64
	12.11.6.1	Creating the Object	12-64
	12.11.6.2	Manipulating the Object	12-64
	12.11.7	Triangle Flavor	12-65
	12.11.7.1	Creating the Object	12-65
	12.11.7.2	Manipulating the Object	12-66
	12.12	Fonts and Text Objects	12-66
	12.12.1	Font Flavor	12-67
	12.12.2	Text Flavor	12-68
	12.12.2.1	Creating the Object	12-68
	12.12.2.2	Manipulating the Object	12-69
	12.12.3	Basic Character Mixin	12-70
	12.12.4	Vector Character Flavor	12-70
	12.12.5	Raster Character Flavor	12-71

Section	Paragraph	Title	Page
	12.13	Ruler Flavor	12-72
	12.13.1	Creating the Object	12-72
	12.13.2	Manipulating the Object	12-74
	12.14	Raster Object Flavor	12-74
	12.14.1	Creating the Object	12-75
	12.14.2	Manipulating the Object	12-76
	12.15	Picture Object	12-76
	12.15.1	Subpicture Flavor	12-76
	12.15.1.1	Creating the Object	12-76
	12.15.1.2	Manipulating the Object	12-77
	12.15.2	Background Picture Flavor	12-78

13 Typeout Windows

13.1	Using Typeout Windows	13-1
13.2	Activation and Deactivation	13-2
13.3	Windows With Inferior Typeout Windows	13-3
13.4	Delaying Redisplay After Typeout	13-4

14 Choice Facilities

14.1	Introduction	14-1
14.2	Menus Facility	14-2
14.2.1	Menu Items	14-2
14.2.2	Column Specification List	14-6
14.2.3	Icons	14-7
14.2.4	Functions That Create Menus	14-11
14.2.4.1	The Most General Function	14-11
14.2.4.2	Special Functions for Compatibility	14-14
14.2.4.3	Other Special Functions	14-16
14.2.5	Keyboard Interface for Menus	14-18
14.2.6	Flavor and Initialization Options That Define Menus	14-19
14.2.6.1	Command Menus	14-19
14.2.6.2	Multiple Menus	14-20
14.2.6.3	Margin Choices	14-21
14.2.6.4	Permanent Versus Pop-Up Menus	14-22
14.2.6.5	Dynamic Item List Menus	14-22
14.2.6.6	Other Kinds of Menus	14-23
14.2.6.7	General Options	14-23
14.2.7	Geometry	14-26
14.2.7.1	Filled Versus Columnar Format	14-26
14.2.7.2	Elements of a Geometry List	14-27
14.2.7.3	Initialization Options and Methods	14-28
14.2.8	Menu Format	14-30
14.2.9	Methods That Implement Type Value Item Types	14-31
14.2.10	Methods That Operate on Menus	14-32
14.2.11	Methods That Reposition the Menu Current Item	14-33
14.3	Multiple-Choice Facility	14-34
14.3.1	Multiple-Choice Functional Interface	14-34
14.3.2	Making Your Own Multiple-Choice Windows	14-36

Section	Paragraph	Title	Page
	14.4	Choose-Variable-Values Facility	14-38
	14.4.1	Specifying the Variables	14-39
	14.4.1.1	Variables in Separate Items	14-39
	14.4.1.2	Variables in Tables	14-41
	14.4.2	Choose-Variable-Values Functional Interface	14-42
	14.4.3	Defining Your Own Variable Type	14-52
	14.4.4	Making Your Own Window	14-53
	14.5	Mouse-Sensitive Items	14-58
	14.5.1	How Mouse-Sensitive Items Work	14-58
	14.5.2	The <code>-M</code> format Directive	14-60
	14.5.3	Using <code>w:basic-mouse-sensitive-items</code>	14-61
	14.6	Margin Choices	14-65

15

Frames

	15.1	Using Frames	15-1
	15.2	Constraint Frames	15-2
	15.3	Constraint Frame Editor	15-4
	15.3.1	Invoking WINIFRED	15-5
	15.3.2	Using WINIFRED	15-5
	15.3.2.1	Initializing the Size and Position of the Constraint Frame	15-6
	15.3.2.2	Changing the Default Values	15-6
	15.3.2.3	Specifying Pane Size and Position	15-6
	15.3.2.4	Specifying Pane Flavor Type and Name	15-9
	15.3.2.5	Writing the Code to a Buffer or File	15-10
	15.3.2.6	Creating a Sample Frame	15-10
	15.3.2.7	Exiting WINIFRED	15-11
	15.3.3	Editing the Generated Code	15-11
	15.3.4	Using the Generated Code in Your Application	15-13
	15.4	Constraint Frame Flavors	15-13
	15.5	Examples of Specifications of Panes and Constraints	15-15
	15.5.1	Simple Constraint Frame	15-15
	15.5.2	Graphics Constraint Frame	15-16
	15.5.3	Multiple-Configuration Constraint Frame	15-17
	15.5.4	Horizontal Constraint Frame	15-21
	15.6	Specifying Panes and Constraints	15-22
	15.6.1	Configuration	15-23
	15.6.1.1	Configuration Description	15-24
	15.6.1.2	Pane-Description-Group Association List	15-25
	15.6.1.3	Description-Group List	15-26
	15.6.1.4	Description-Group	15-26
	15.6.1.5	Constraint	15-27
	15.6.1.6	Pane-Description-Group Association List	15-28
	15.6.1.7	Description-Group List	15-28
	15.6.1.8	Description-Group	15-28
	15.7	Constraint Frame Keys	15-29
	15.7.1	Arguments Used With <code>w:basic-frame</code>	15-29
	15.7.1.1	<code>fixnum</code>	15-29
	15.7.1.2	<code>flonum</code>	15-29
	15.7.1.3	<code>:even</code>	15-29
	15.7.1.4	<code>:ask</code>	15-30
	15.7.2	The <code>:limit</code> Clause	15-30
	15.7.3	<code>:pane-size</code> and Similar Methods	15-31

Section	Paragraph	Title	Page
	15.8	Embedded Configurations	15-32
	15.9	Constraint Frame Methods	15-34
	15.10	Pane-Frame Interaction	15-35
	15.10.1	The Selected Pane	15-36

16 Text Scroll Windows

	16.1	Using Text Scroll Windows	16-1
	16.2	Specifying the Item List	16-2
	16.3	Function Text Scroll Windows	16-5
	16.4	Item Generators	16-6
	16.5	Mouse-Sensitive Text Scroll Windows	16-9
	16.5.1	Mouse Blips	16-11
	16.6	Inspector Flavors	16-12

17 General Scroll Windows

	17.1	Using General Scroll Windows	17-1
	17.2	Specifying Items and Entries	17-3
	17.3	Using a Scroll Window	17-5
	17.4	Inserting and Deleting Items	17-7
	17.5	Automatically Updating Items	17-8
	17.6	Representation of Items	17-10
	17.7	Mouse-Sensitive Scroll Windows	17-11
	17.8	Peek Flavor	17-12

18 Miscellaneous Features

	18.1	Introduction	18-1
	18.2	Notifications	18-1
	18.3	Creating and Recording Sounds	18-4
	18.3.1	Beeps	18-4
	18.3.2	Making Sounds	18-6
	18.3.3	Recording and Playing Sounds	18-8
	18.4	Specific Types of Windows	18-10
	18.4.1	Lisp Listeners	18-10
	18.4.2	Editor Windows	18-11
	18.4.3	Window Flavors for Other Programs	18-14
	18.5	The Who-Line	18-15
	18.5.1	Mouse Documentation Window	18-15
	18.5.2	Status Line	18-16
	18.5.2.1	File Streams	18-16
	18.5.2.2	Servers	18-17
	18.6	The System Menu	18-18
	18.7	Window Resources	18-20
	18.8	Finding Windows	18-21

Section	Paragraph	Title	Page
19		Using Color	
	19.1	Introduction	19-1
	19.2	Requirements for Coding in Color	19-1
	19.3	How Color Works on a Monitor	19-2
	19.3.1	Foreground and Background Colors	19-2
	19.3.2	LUTs and the Color Map	19-3
	19.3.3	Contents of the Color Map	19-4
	19.3.4	Reserved Colors	19-5
	19.3.5	Naming Colors	19-5
	19.4	Initialization Options and Methods Used With Color Windows	19-7
	19.5	Functions That Manipulate the Color Map	19-8
	19.5.1	General Functions	19-10
	19.5.2	Hardware-Specific Functions	19-11
	19.6	Color ALU Functions	19-12
	19.6.1	Add and Subtract	19-14
	19.6.2	Add With Saturate and Subtract With Clamping	19-14
	19.6.3	Minimum, Maximum, and Average	19-14
	19.6.4	Background and Transparency	19-15
	19.6.5	Color Versus Monochrome ALU Functions	19-16
	19.7	Profile Variables for Color	19-18
	19.8	Color and Texture in Graphic Output	19-18
	19.9	Printing a Color Screen on a Monochrome Printer	19-21
	19.10	Plane Masks	19-23
	19.11	Methods to Control Monitors Directly	19-23

Section	Paragraph	Title	Page
A		Obsolete Symbols	
	A.1	Introduction	A-1
	A.2	Scrolling	A-1
	A.3	Graphics	A-5
	A.3.1	tv:graphics-mixin	A-5
	A.3.2	tv:stream-mixin	A-9
	A.3.3	gwin:draw-mixin	A-9
	A.3.4	Primitives	A-11
	A.4	Menus	A-12
	A.4.1	Flavors and Methods	A-12
	A.4.2	Functional Interface	A-13
	A.4.3	Geometry	A-13
	A.4.4	Ordinary Menus	A-14
	A.4.5	Command Menus	A-17
	A.4.6	Dynamic Item List Menus	A-18
	A.4.7	Multiple Menus	A-21
	A.4.8	Making Your Own Multiple Menus	A-21
	A.5	Input	A-23

Section	Paragraph	Title	Page
B		Converting Applications to Color	
	B.1	Introduction	B-1
	B.2	Converting to Color	B-1
	B.2.1	Requirements	B-1
	B.2.2	Converting the Load Band	B-1
	B.3	Compatibility Issues	B-2
	B.3.1	Using the Correct ALU Arguments	B-3
	B.3.2	Using Bit-Save Arrays	B-3
	B.3.3	Using Drawing Methods With Added Arguments	B-4
	B.3.4	Using Previously Unused Instance Variables	B-5

	Figure	Title	Page
Figures	5-1	Overlapping Windows	5-1
	7-1	Pseudo-Code for Character Displaying	7-4
	19-1	Menu Produced by <code>w:select-texture-with-mouse</code>	19-20

	Table	Title	Page
Tables	9-1	Some Commonly Used Fonts	9-2
	9-2	Purpose Keywords for Fonts	9-3
	12-1	Color Values for Graphic Methods for Monochrome Environments	12-12
	12-2	ALU Values for Graphic Methods	12-13
	14-1	Type Value Keywords for Menus	14-4
	14-2	Menu Item Modifier Keywords	14-5
	14-3	Predefined Variable Types for <code>w:choose-variable-values</code>	14-43
	14-4	Item Modifiers for <code>w:choose-variable-values</code>	14-46
	14-5	The <code>w:choose-variable-value</code> Options Keywords	14-48
	16-1	Keywords for the Item Generator Function	16-7
	19-1	Elements in the Color Map <code>defstruct</code>	19-4
	19-2	Named Colors in the Default Color Map Table	19-6
	19-3	Color ALU Operations	19-16
	19-4	Monochrome ALU Functions Used by the Color ALU Functions on a Monochrome System	19-16
	19-5	Truth Tables for Color ALU Functions on Monochrome Displays	19-17
	19-6	Profile Variables for Color	19-18
	19-7	Gray Patterns Used for Printing the Named Colors	19-22

ABOUT THIS MANUAL

Introduction

The *Explorer Window System Reference* manual is intended to explain how you, as a programmer, can use the set of Explorer facilities known collectively as the window system. Specifically, this document explains how to create windows and what operations can be performed on them. It also explains how you can customize the windows you produce by mixing together existing window definitions to produce a window that can perform as your program requires.

Assumptions

This manual assumes that you have:

- A working familiarity with Lisp as documented in the *Explorer Lisp Reference* manual.
- Some experience with the user interface of the Explorer system, including how to manipulate windows, such as the Edit Screen, Split Screen, and Create commands from the System menu.
- An understanding of the following:
 - What message passing is
 - How message passing is used on the Explorer
 - What a flavor is
 - What a mixin flavor is
 - How to define a new flavor by mixing existing flavors

These concepts are explained in the *Explorer Programming Concepts* manual and in the *Explorer Lisp Reference* manual.

To use the predefined flavors and methods, you need not be familiar with how methods are defined and combined. However, to use the information provided here on where to add `:before` and `:after` methods, you must be thoroughly familiar with programming with flavors.

Contents of This Manual

The first section of this manual, Basic Concepts:

- Discusses the concepts upon which the window system is based. You must understand these concepts before you can understand the discussions in the other sections.
- Briefly describes the contents of each of the other sections.
- Gives a simple example of creating a window.

- Lists a simple decision tree to help you identify the features you want for a particular window. This tree lists where to find more information about a feature.

Section 2, Basic Windows, discusses the underlying flavors of the window system, such as `w:minimum-window`.

The remaining sections of this manual each describe a particular behavior or type of window and discuss the flavors, methods, initialization options, variables, and functions that provide that behavior or type.

Appendix A, Obsolete Symbols, describes the symbols that are no longer supported and that will be not be available in the next release.

Suggestions for Using This Manual

If you are unfamiliar with creating windows, you should read Section 1 carefully, examining the various windows already existing on the system. After you have an idea of how you want your window to behave, refer to the section that includes the specific behavior you want to include in your window.

If you are already familiar with using the window system, you should refer to the section that includes the specific behavior you want to include in your window.

User Comments

We are trying to make this manual an easily understood tool to aid you in using the window system. If you have questions, criticisms, or suggestions for improvement, please use the User Response Sheet provided at the back of the manual.

Notational Conventions

The following paragraphs describe the notation for keystroke sequences, mouse clicks, and Lisp syntax, as well as flavors, initialization options, methods, and instance variables.

Keystroke Sequences

Many of the commands used with the Explorer system are executed by a combination or sequence of keystrokes. Keys that should be pressed at the same time, or *chorded*, are listed with a hyphen connecting the name of each key. Keys that should be pressed in a particular sequence are listed with a space separating the name of each key. The following table explains the conventions used in this manual to describe keystroke sequences.

Keyboard Sequence	Interpretation
META-CTRL-D	Hold the META and CTRL keys while pressing the D key.
CTRL-X CTRL-F	Hold the CTRL key and press the X key, release the X key, and then press the F key. Alternately, press CTRL-X, release both keys, and press CTRL-F.
META-X Find File RETURN	Hold the META key while pressing the X key, release the keys, type the letters find file and then press the RETURN key.
TERM - SUPER-HELP	Press the TERM key and release it, press the minus key (-) and release it, then press and hold the SUPER key while pressing the HELP key.

Mouse Clicks

The mouse has three buttons that enable you to execute operations from the mouse without returning your hand to the keyboard. Pressing and releasing a button is called *clicking*. The following table lists abbreviations used to describe clicking the mouse.

Abbreviation	Action
L	Click the left button (press the left button once and release).
M	Click the middle button (press the middle button once and release).
R	Click the right button (press the right button once and release).
L2, M2, R2	Click the specified button twice quickly. Alternately, you can press and hold the CTRL key while you click the specified button once.
LHOLD, MHOLD, RHOLD	Press the specified button and hold it down.

Lisp Language Notation The Lisp language notational convention helps you distinguish Lisp functions and arguments from user-defined symbols. The following table shows the three fonts used in this manual to denote Lisp code.

Typeface	Meaning
boldface	System-defined words and symbols, including names of functions, macros, flavors, methods, variables, keywords, and so on—any word or symbol that appears in the system source code.
<i>italics</i>	Example names or an argument to a function, such as a value or parameter you would fill in. Names in italics can be replaced by any value you choose to substitute. (Italics are also used for emphasis and to introduce new terms.)
monowidth	Examples of program code and output are in a monowidth font. System-defined words shown in an example are also in this font.

For example, this sentence contains the word **setf** in boldface because **setf** is defined by the system.

Some function and method names are very long—for example, **get-ucode-version-of-band**. Within the text, long function names may be split over two lines because of typographical constraints. When you code the function name **get-ucode-version-of-band**, however, you should not split it or include any spaces within it.

Within manual text, each example of actual Lisp code is shown in the monowidth font. For instance:

```
(setf x 1 y 2) => 2
(+ x y) => 3
```

The form `(setf x 1 y 2)` sets the variables `x` and `y` to integer values; then the form `(+ x y)` adds them together.

In this example of Lisp code with its explanation, `setf` appears in the monowidth font because it is part of a specific example.

For more information about Lisp syntax descriptions, see the *Explorer Lisp Reference* manual.

Flavor Notation The window system is composed of various flavors and associated symbols, such as methods, initialization options, and instance variables. Some flavors require certain symbols to operate properly. For example, any window built with the `w:mouse-blinker-mixin` flavor also requires the `w:blinker` flavor to be in the component list. In this manual, such requirements are shown as follows:

w:mouse-blinker-mixin

Flavor

Required flavor: `w:blinker`

Makes a blinker suitable for use as the mouse blinker. Not all blinkers....

You can also find the requirements for a flavor by using the **describe** function. For example, if you typed the form (describe 'w:mouse-blinker-mixin) in a Lisp Listener, the system returns the following:

```
Symbol W:MOUSE-BLINKER-MIXIN is in the W package.
W:MOUSE-BLINKER-MIXIN has property SYS::DOCUMENTATION-PROPERTY: (DEFFLAVOR "Blinker that is
capable of being MOUSE-BLINKER A MIXIN Flavor.")
W:MOUSE-BLINKER-MIXIN has property SYS:FLAVOR: #<FLAVOR W:MOUSE-BLINKER-MIXIN 61200463>

Flavor #<FLAVOR W:MOUSE-BLINKER-MIXIN 61200463> directly depends on flavors: none
and is directly depended on by W::MOUSE-MULTIPLE-RECTANGLE-BLINKER,
W::MOUSE-FOLLOWING-ARROW-BLINKER, W:MOUSE-BOX-STAY-INSIDE-BLINKER,
W:MOUSE-BOX-BLINKER, W:MOUSE-HOLLOW-RECTANGULAR-BLINKER, W:MOUSE-RECTANGULAR-BLINKER,
W:BITBLT-BLINKER, W::MOUSE-BLINKER-FAST-TRACKING-MIXIN
Not counting inherited methods, the methods for #<FLAVOR W:MOUSE-BLINKER-MIXIN 61200463>
are:
:TRACK-MOUSE
:SET-OFFSETS
:OFFSETS
Instance variables that may be set by initialization: W:X-OFFSET, W:Y-OFFSET
Defined in package W
Properties:
:DOCUMENTATION: (:MIXIN "Blinker that is capable of being MOUSE-BLINKER")
:REQUIRED-FLAVORS: (W:BLINKER)
SYS::INSTANCE-AREA-FUNCTION: NIL
SYS::REQUIRED-INIT-KEYWORDS: NIL
SYS::REMAINING-INIT-KEYWORDS: NIL
SYS::REMAINING-DEFAULT-PLIST: NIL
SYS::ALL-INITTABLE-INSTANCE-VARIABLES: NIL
SYS::ALL-SPECIAL-INSTANCE-VARIABLES: NIL
SYS::INSTANCE-VARIABLE-INITIALIZATIONS: NIL
SYS::MAPPED-COMPONENT-FLAVORS: NIL
SYS::UNMAPPED-INSTANCE-VARIABLES: (W:X-POS W:Y-POS W:SHEET ...)
COMPILE-FLAVOR-METHODS: NIL
SYS::ALL-INSTANCE-VARIABLES-SPECIAL: NIL
SYS::ADDITIONAL-INSTANCE-VARIABLES: NIL
Flavor #<FLAVOR W:MOUSE-BLINKER-MIXIN 61200463> does not yet have a method hash table

This is an ART-Q type array.
It is 20 long.

W:MOUSE-BLINKER-MIXIN has property :SOURCE-FILE-NAME: ((DEFFLAVOR # FS::REMOTE-LM-PATHNAME
"DLS: WINDOW; WDEFS.#" # FS::LOGICAL-PATHNAME "SYS: WINDOW; WDEFS.#" ))
W:MOUSE-BLINKER-MIXIN
```

Note the following parts of the description:

- ① Methods defined for this flavor. These methods do not include the methods inherited from component flavors.
- ② Initialization options for this flavor. Another part of the flavor description, not shown here, consists of the instance variables that are gettable and settable.
- ③ The documentation string for this flavor.
- ④ The required flavors for this flavor.
- ⑤ The pathname of the source file. You can examine the source file to see the component list for this flavor.

Other important information, which in the case of this flavor is **nil**, includes required initialization options (the `REQUIRED-INIT-KEYWORDS`), the default values for instance variables (the `REMAINING-DEFAULT-PLIST` or the `INSTANCE-VARIABLE-INITIALIZATIONS`), and required methods.

Flavor Naming Conventions Conventions are different for *instantiable* flavors (which are complete and can support instances of themselves) and *mixin* flavors (which are incomplete and supply only one particular aspect of behavior).

In the following convention examples, the word *frobboz* is used to stand for any feature, attribute, or class of windows that would appear in a flavor name.

frobboz Flavor

An instantiable flavor whose most distinguishing characteristic is that it is a complete Lisp object. As a complete Lisp object, **frobboz** has the minimum attributes needed to make it instantiable. The **frobboz** name is preferred to the **frobboz-window** name except when it is necessary to make a distinction between a window having **frobboz** attributes and the **frobboz** flavor.

frobboz-mixin Flavor

Provides the **frobboz** feature when mixed into other flavors, but **frobboz-mixin** is not instantiable by itself. Such mixins may have components such as other mixins; may have no components, only *required flavors*; or may not have required flavors because the mixin is a general-purpose mixin used by several flavors.

Required flavors are flavors that must be present as a component of the flavor at some level. This manual uses the following convention to indicate that a mixin requires a flavor:

baz-mixin Flavor

Required flavor: **frobboz-mixin**

Requires the **frobboz-mixin** flavor as a component flavor, either directly or through inheritance.

Initialization Options, Methods, and Instance Variables Rather than separately listing each method, this manual uses the following convention to document initialization options and **:get-** and **:set-** methods. In most cases, the initialization option is documented rather than the instance variable. For example, the following syntax line documents the **:hysteresis** initialization option, the **:hysteresis** method, and the **:set-hysteresis** method. Methods so documented may simply access the instance variable, or the methods may do other things as well.

:hysteresis *hysteresis* Initialization Option of **w:hysteretic-window-mixin**
Gettable, settable. Default: 25.

The value, in pixels, of the hysteresis...and so on.

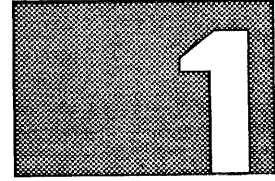
If the initialization option and both methods were listed explicitly, they would appear as follows:

:hysteresis *hysteresis* Initialization Option of **w:hysteretic-window-mixin**
Default: 25.

:hysteresis Method of **w:hysteretic-window-mixin**
:set-hysteresis *hysteresis* Method of **w:hysteretic-window-mixin**

In cases where a **:set-** method uses more than one argument, that method is listed separately, as in the following example.

- | | |
|-------------------------|---------------------------------------------------------------------------------|
| :offsets | Method of w:mouse-blinker-mixin |
| | Returns two values: the x and y offsets of the blinker. The values...and so on. |
| :set-offsets x y | Method of w:mouse-blinker-mixin |
| | Sets the offsets of the blinker to the values specified by x and y...and so on. |



WINDOW SYSTEM CONCEPTS

General Concepts

1.1 This section is designed to introduce you to the concepts that form the basis of the window system. You must understand these concepts to understand the discussions in this manual. After the discussion of the very basic concepts, this section then briefly describes the contents of each of the other sections and gives a simple example of creating a window.

The final portion of the section provides a simple decision tree to help you identify the features you want for a particular window. This tree lists where to find more information about a feature.

Screens, Windows, and Panes

1.1.1 The *window system* is a hierarchical input/output (I/O) interface between the Explorer system, its peripherals, and the user. *Windows* are generally thought of as areas shown on the video display; however, this is not necessarily so. Windows can also exist as areas in memory waiting to be displayed.

A *screen* is the topmost node of the window system hierarchy; it defines an area of the physical display. A screen has no superior windows, and windows in one screen cannot be extended into another screen. Each screen object usually corresponds to a particular location in the Explorer memory.

A process is often associated with a window and uses that window for input and output. Thus, a window can act as a stream. Each window is an instance of a specific flavor. A *frame* is a collection of inferior windows, or *panes*, that can be manipulated as a single window. An example of a frame is the Inspector. The Inspector frame contains three history panes, an interaction pane, a command menu pane, and a summary pane. You specify the relative size and location for each of the panes in a set of *constraints*. The constraint frame editor, or WINIFRED™, is a window-driven interface that enables you to develop constraint frames easily.

The Explorer video display shows *windows* and *screens*. A window may or may not appear on the video display, but a screen always appears on the video display. Screens are divided into *windows*, and the windows can be subdivided into *inferior windows* or *panes*. Historically, windows and screens collectively are called *sheets*. The term sheet frequently appears in the source code. By convention (and in common usage) frames, panes, sheets, and screens are often referred to simply as windows.

WINIFRED is a trademark of Texas Instruments Incorporated.

The Explorer window system can be used on either a monochrome system or on an optional color system. This manual describes the features that are common to both and also the differences. Your application can be coded to work correctly on either a monochrome or a color system. However, certain guidelines must be followed. The following list describes the main places in this manual (and other manuals) where color is discussed:

- Functions and concepts that are specific to color are described in Section 19, Using Color.
- Problems that may arise when converting applications to color are discussed in Appendix B, Converting Applications to Color.
- Several text output functions now have a new optional color argument, as described in Section 7, Output of Text.
- The color argument in the graphics methods allows you to use color on a color system, as described in Section 12, Graphics.
- A broad overview on color concepts is presented in the Color Concepts section of the *Explorer Programming Concepts* manual.
- Refer to the description of Edit Attributes in the Customizing Your Environment section of the *Introduction to the Explorer System* to merely change the foreground, background, label, background label, and border color of a window.
- To edit the colors that can be displayed in a window, refer to the Color Map editor section in the *Explorer Tools and Utilities* manual.

The Main Screen

1.1.1.2 You have encountered windows many times in your use of the Explorer system. Sometimes there is only one window visible on the video display.

Using the System menu's Create, Edit Screen, or Split Screen options, a user can make windows of various sizes and flavors and position them on the video display. Similarly, you can write programs to control the size, position, and behavior of windows using the information in this manual.

**The Window System
as a User Interface**

1.1.2 The *window system* refers to a large body of software used to manage communications between the user and programs in the Explorer, via the Explorer system console, sometimes called the display unit. The console consists of a keyboard, a mouse, and a monitor.

Windows can function as streams by accepting all the operations that streams accept. Input operations on windows read from the keyboard or mouse; output operations on windows type out characters or graphics on the screen. The value of **terminal-io** is normally a window, so I/O functions on the Explorer system send their I/O to windows by default. Other standard streams, such as **standard-output**, **standard-input**, and so on are synonym streams of **terminal-io** so they also send I/O to windows by default.

The window system controls the keyboard, encoding the shifting keys, interpreting special commands from the TERM and SYSTEM keys, and directing input to the correct place. The window system also controls the mouse, tracking it on the screen, interpreting clicks on the buttons, and routing its effects to the correct places. The most important part of the window system is its control of the screens, which it subdivides into windows so that many programs can coexist and even run simultaneously without getting in each other's way, sharing the screen space according to a set of established rules.

States of a Window

1.1.3 A window can be in different states:

- A *visible* window is a window that is currently displayed on a screen. A window is fully visible if no other window is covering any part of it; a window is partially visible if another window(s) is covering part of it.
- An *exposed* window is a window that is enabled to accept output from a process. An exposed window need not be visible. For a window to be exposed, the window's superior must have some place to put its output (that is, it must have a *bit-save* array).
- The *selected* window is the window that receives the input from the keyboard. Only one window can be selected at one time, and when a window becomes selected, it is exposed and made fully visible automatically. The currently selected window is usually indicated by a blinking keyboard cursor.
- A *deexposed* window is a window that had been exposed but is no longer exposed. This can happen either when a program or user explicitly deexposes a window, or when another window is selected.

- An *active* window is a window that is capable of being exposed. Active windows are part of a list of windows (a *stack*) that can be exposed without allocating more memory or creating another window. All visible windows and all exposed windows are active. In addition, other windows (windows that are not visible or that are not exposed) are active if they are resident in memory.
- A *deactivated* window is a window that is not exposed nor selected and that is not on the stack. When a window is first created, it is deactivated until it is exposed or selected.

You can *bury* an active window—put it logically at the end of the stack of windows, and make the next window(s) in the stack visible. A buried window is still active, but it is not selected. The buried window may or may not be partially visible, depending on the sizes of the other windows in the stack.

You can *kill* a window—make it both inactive and deexposed, remove it from the stack, and delete it from memory. If a process is associated with a window, when you kill the window you also kill the associated process.

Windows as Instances of Flavors

1.2 In the Explorer system, each window is a flavor instance. Many different window flavors are available; those that you can use are described in this manual. Because windows are flavors, they have certain characteristics. You should be aware of the facilities available on the Explorer system for using, mixing, and examining flavors.

Characteristics of Window Flavors

1.2.1 Because windows are flavors, they share the following characteristics with all flavors:

- Have instance variables, which can be gettable, settable, and inittable. Many instance variables can also be accessed by macros or defsubst.
- Have methods that perform various operations on the windows. These methods can include `:before` and `:after` daemons and wrappers to customize the methods as you desire.
- Can be mixed as needed.
- Are instantiated with `make-instance`.
- Contain the `sys:vanilla-flavor` methods.

If you are unfamiliar with flavors, you should refer to the explanation of flavors in the *Explorer Programming Concepts* manual and in the *Explorer Lisp Reference* manual.

This manual describes the initialization options and methods of various flavors of windows. They are grouped in each section of this manual by the functions they perform.

Mixing Flavors

1.2.2 Many programs never need to create any new windows. Often you can use the standard input, output, and graphics methods on an existing window, such as a Lisp Listener, which is the value of `*terminal-io*` when your program is called. For example, the following code defines a function that, when executed, draws a pattern of XORed circles on any window that has `w:graphics-mixin` (such as a Lisp Listener).

```
(defun green-hornet (&optional (window *terminal-io*) (separation 40))
  (send window :clear-screen)
  (send window :home-down)
  (multiple-value-bind (iw ih)
    (send window :inside-size)
    (let ((center-x1 (- (truncate iw 2)
                       (truncate separation 2)))
          (center-x2 (+ (truncate iw 2)
                       (truncate separation 2)))
          (center-y (truncate ih 2)))
      (do ((i (- (min center-y center-x1) 10.) (1- i)))
          ((=< i 5))
          (send window :draw-circle
                  (if (bit-test #o20 i) center-x1 center-x2)
                  center-y i))))
    (read-char window)
  t)
```

Often you must mix flavors to make windows behave as you want. When mixing flavors, you must pay attention to the correct ordering of flavor components. The earlier components override later ones. For example, if you want to make a window that prints out notifications on itself by mixing in `w:notification-mixin`, you must put `w:notification-mixin` before `w>window`:

```
(deflavor my-window () ; Correct ordering
  (w:notification-mixin
   w>window))
```

However, if you put `w>window` in front of `w:notification-mixin`, you get something equivalent to `w>window`:

```
(deflavor my-window () ; Incorrect ordering
  (w>window
   w:notification-mixin))
```

In the second example, the effect of `w:notification-mixin` is completely lost. The whole purpose of `w:notification-mixin` is to override some methods of `w>window` (inherited from `w:delay-notification-mixin`), and, in fact, it defines the same methods in a different way. If `w:notification-mixin` comes last, it is overridden instead.

Methods and Their Flavors

1.2.3 Methods are usually associated with a particular flavor. For example, the `:print-notification-on-self` method is a method of the `w:notification-mixin`. However, because some flavors are very basic, this manual documents some methods and initialization options differently:

- All the methods, instance variables, and initialization options of `w:minimum-window` are documented as being of *windows* rather than of any specific flavor.
- All the methods, instance variables, and initialization options of `w:sheet` are documented as being of *windows and screens*.

Features Common to All Windows

1.3 All windows share some features, such as borders, names, blinkers, input and output operations, and so on. The following sections discuss the features that can be used on almost all windows.

Outside Edges of Windows

1.3.1 Windows usually have a border (a thin black line around the edges of the window), and they frequently have a label at the top or in the lower left corner. The border helps you see where all the windows are, what kind of windows they are, and what parts of the screen they are using.

A window's area of the screen is divided into two parts. Around the edges of the window are the four *margins*. While margins can have zero size, there usually is a margin on each edge of the window, holding a border and sometimes other things, such as a label. The rest of the window is called the *inside*; regular character output and graphic drawings all occur on the inside part of the window. The margins and the inside of the window are managed separately so mixins that add things to the margins can be independent of the program that draws in the window's inside.

Mixins usually override similar attributes in flavors to which the mixins are added. One exception occurs with flavors of margin items in which the ordering controls the spatial position of the margin items.

Section 3, Outside Edges of Windows, discusses the flavors, methods, and variables that produce margins, borders, and labels.

Sizes and Positions

1.3.2 Every window has a size and position. You can specify these in several ways: specify numerically, use the mouse, cause a window to be displayed near some point or some other window, and so on.

Section 4, Sizes and Positions, discusses the flavors, methods, and variables affecting the size and position of a window.

Visibility and Exposure

1.3.3 A window can be fully *visible* (you can see the entire window) or only partially visible (some of the window is covered by another window). A window may or may not be *exposed*; an exposed window can have output drawn on it.

Section 5, Visibility and Exposure, discusses the flavors, methods, and variables affecting visibility and exposure of windows. This section also discusses bit-save arrays.

Selection

1.3.4 Before a window can receive input from the keyboard, the window must be selected. Windows can be selected by programs, or a user can select a window either with keystroke sequences (for example, the user can select the Inspector window by pressing SYSTEM I), by using the System menu, or by moving the mouse blinker into a partially visible window and clicking the left button.

Section 6, Selection, discusses how to select a window, and how selection can be controlled when several windows are involved.

Output of Text 1.3.5 Windows handle the standard output stream operations and can be passed as the output stream to functions such as **print** and **format**. You can output characters at a cursor position, move the cursor around, selectively clear parts of the window, insert and delete lines and characters, and so on, by means of stream operations. Output of text on windows provides additional features. For example, characters can be drawn in any of a large set of *fonts* (typefaces), and you can switch from one font to another within a single window. Windows can define their own actions for exceptional conditions that affect output, such as attempting output beyond the right or bottom edge of the window or printing more than a window-full without pausing.

Section 7, Output of Text, discusses how output is actually performed, how to cause output to appear or disappear from a window, and what exception conditions are handled by windows.

Input 1.3.6 A window whose flavor incorporates the **w:stream-mixin** flavor supports all the standard input stream operations and can be passed as the input stream to functions such as **read** and **read-line**. Each such window has an *input buffer* that holds input for the window that has not been read yet. You can *force keyboard input* into a window's input buffer; frequently two processes communicate by one process forcing keyboard input into an input buffer that another process is reading.

Asynchronously intercepted characters (such as CTRL-ABORT) take effect immediately when typed and are handled by the selected window. Each window can specify its own asynchronously intercepted characters as well as synchronously intercepted characters.

Section 8, Input, discusses input, I/O buffers, and intercepted characters.

Fonts 1.3.7 Fonts control the appearance of text on the window.

Section 9, Fonts, discusses how fonts are specified and the attributes and format of fonts.

Blinkers 1.3.8 Blinkers are used to add either visual cues to a window or temporary modifications to a window's normal display. Blinkers are flavor instances with their own standard operations. Each window can have many *blinkers*. Most windows have one blinker that follows the window's cursor position; this blinker normally appears as a blinking rectangle. However, blinkers need not follow the cursor and need not actually blink (some do and some do not).

Blinkers are not always rectangular. The arrow that follows the mouse is a blinker, but the mouse blinker is not always an arrow. For example, the Zmacs editor shows you the character the mouse is pointing to by changing the mouse blinker to a hollow rectangle.

Section 10, Blinkers, discusses various types of blinkers and how to use them.

The Mouse 1.3.9 Windows are the standard interface to the mouse. Both mouse motion and mouse clicks are normally handled by messages sent to the window over which the mouse is positioned.

Section 11, The Mouse, discusses different ways of controlling the mouse, operations to use to find out the current status of the mouse, and the scrolling facility.

Graphics 1.3.10 In addition to characters from fonts, you can also draw graphics (pictures) on windows. You can use functions, methods, and subprimitives to draw lines, circles, triangles, rectangles, arbitrary polygons, circle sectors, and cubic splines.

Section 12, Graphics, describes the functions, methods, and subprimitives that draw graphics on windows.

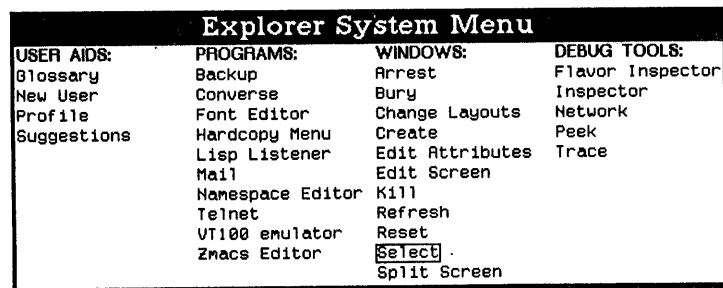
Typeout Windows 1.3.11 Typeout windows allow windows such as scroll windows and editor windows to print output in response to individual commands. The typeout window is an inferior of the other window, and exposes itself when output is drawn on it.

Section 13, Typeout Windows, discusses how to use a typeout window. The section includes an example of a standalone typeout window.

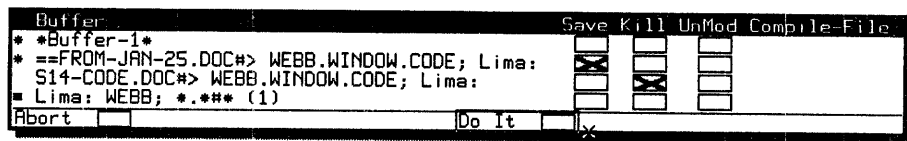
Types of Windows 1.4 Sections 14 through 18 discuss windows that are of different types. In general, these types of windows are mutually exclusive. For example, if a window is a choice facility, it is not a general scroll window. These windows use many of the features discussed in paragraph 1.3, Features Common to All Windows.

Choice Facilities 1.4.1 Choice facilities consist of menus, multiple choice windows, and choose-variable-values windows.

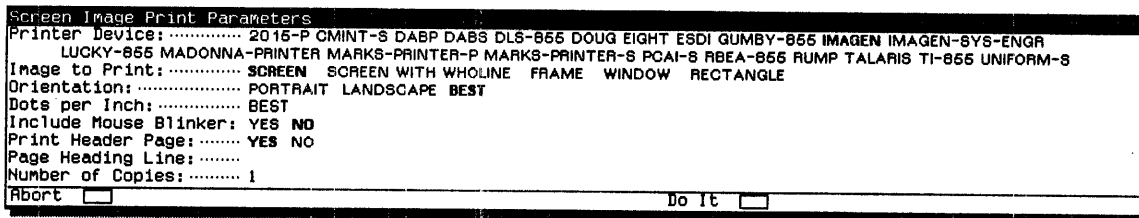
Menus allow the user to choose one or several of a fixed set of items. The System menu produced by the double-click-right sequence is an example of such a menu.



Multiple-choice windows allow the user to specify an answer to each of a set of similar multiple-choice questions. The Kill or Save Buffers window from the Zmacs editor is an example of a multiple-choice menu.



Choose-variable-values windows allow the user to view and modify the values of a set of variables. Each variable is printed and read according to its own range of possible values. One variable might allow only numbers, while another variable's value might be restricted to a list of pathnames. The Hard Copy choice in the System menu invokes a choose-variable-values window.



Section 14, Choice Facilities, discusses how to use choice facilities. For each choice facility, the window system includes an easy-to-use interface.

Frames 1.4.2 For greater flexibility in subdividing a window into multiple areas for different uses, you can create *inferior windows* or *panes* within the window. The main window is then called a *frame*. Each pane can be of a different flavor suitable to its own purpose. For example, Peek uses a menu frame and a scrolling window as its two panes.

Section 15, Frames, discusses how to use frames by specifying the panes that comprise them, and shows examples of different configurations. This section also describes the constraint frame editor, which enables you to quickly generate the code for a constraint frame.

Text Scroll Windows 1.4.3 Text scroll windows provide a simple way to display and scroll a number of lines of similar material on a window. For example, editor windows and menus can scroll.

The lines displayed by text scroll windows are Lisp objects. How these Lisp objects are displayed depends on how certain methods are defined.

Section 16, Text Scroll Windows, discusses the uses of text scroll windows and how to specify them.

**General Scroll
Windows**

1.4.4 General scroll windows are an independent facility from text scroll windows. Although these windows are called scroll windows, the windows do not actually scroll. The contents of the windows are updated as items in the windows change. Peek is a good example of a general scroll window.

General scroll windows display continuously maintained items, which can vary in size. Each item in the window takes up an entire line, and each item can be composed of one or more subitems arranged vertically. Each subitem can fill one or more lines and can contain subsubitems. The lowest level of subdivision of subitems consists of entries that are arranged in a horizontal sequence. New subitems, at any level, can be added and deleted automatically, and the display is updated automatically by moving lines around on the window.

Section 17, *General Scroll Windows*, discusses the uses of general scroll windows and how to specify them.

**Miscellaneous
Features**

1.4.5 Section 18 discusses miscellaneous topics:

- Notification windows are used to notify a user of the occurrence of a particular event.
- Beeps can provide audible feedback to the user.
- You can also program the Explorer system to create sounds or to record and play back sounds.
- Some programs, such as the Lisp Listener or the Zmacs editor, have flavors that can be used to include their facilities in a window.
- The user can specify some aspects of the appearance of the mouse documentation window and status line.
- The System menu provides a menu interface to most of the programs and features of the Explorer system.
- Window resources help conserve system memory and speed execution.
- The Explorer system includes functions to help you find existing windows, resources, flavors, and methods.

Optional Color

1.4.6 Section 19 discusses the concepts and functions for using an optional color system. Appendix B, *Converting Applications to Color*, discusses how to convert from a monochrome load band to one that uses color, and lists problem areas that may arise when you convert an existing monochrome application to use color.

Obsolete Symbols

1.5 This release of the Explorer window system replaces some code with improved code of similar names. The obsolete symbols are documented in Appendix A and are marked with an *[o]* symbol in the syntax line.

Designing a Window

1.6 When you start to design a new window for your application, you should use the simplest mechanism that meets your needs. The steps are presented as follows in order of increasing complexity:

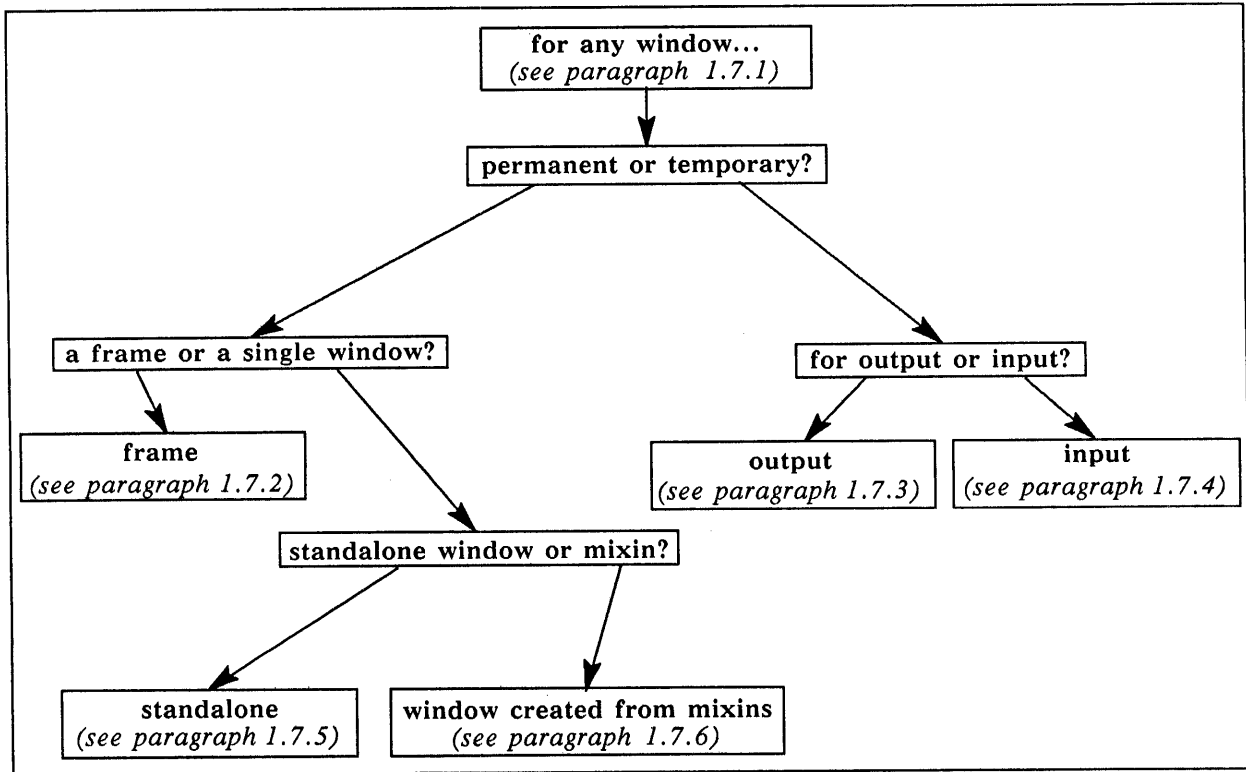
1. If possible, choose a functional interface that creates a particular window automatically. Many temporary windows, such as menus and notifications, can be created using function calls.
2. Use a utility that generates the window code for you. The frame editor, described in paragraph 15.3, Constraint Frame Editor, is one such utility.
3. Create a new window instance by using existing flavors and mixins.
4. Customize the methods of an existing flavor or mixture of flavors to produce special results.
5. Build your own window flavor.

The remainder of this section includes the following:

- A simple decision tree that may help you to decide what kind of window you want, including a general explanation of each of the choices of the decision tree
- Short explanations of the functions and flavors available to implement each leaf of the decision tree

General Choices Among Windows

1.7 The following tree summarizes the general choices you should make when you design a window. The following pages describe these choices in each leaf in more detail and refer you to the portion of the manual that describes that feature. Some references are to other manuals.



For any Window...

Answer the questions listed in paragraph 1.7.1 for any window or pane.

Permanent or Temporary?

- Use *permanent* windows when you want the window to (almost) always appear when the user is interacting with your software. If you have cases when you need to display or gather information, you can associate various temporary windows with your main permanent window.
- Use *temporary* windows when you want to display information briefly, notify the user of some condition that may require attention, or infrequently get information from the user.

A Frame or a Single Window?

- Use a *frame* to display a group of panes as a single window. Examples of panes are the Inspector and Peek. Note that you can include Suggestions on a window without the window being a frame.
- Use a *single window* if one window can serve all your needs. For example, the new user utility uses a single window.

Existing Standalone Window or Mixin?

- Use an instance of an *existing standalone window*, if possible, to serve your needs. You can customize many features of standalone windows using existing initialization options and methods. For example, any initialization option or method marked as being of *windows and screens* works on all windows based on the *w:sheet* flavor, which includes virtually all window flavors.
- Create your own window flavor using the *w:minimum-window* flavor (or another flavor based on these) with various mixins to create a specialized window.

Whether you use an existing window or you create your own window flavor, you can redefine existing methods to customize window behavior.

For Output Or Input?

You can use a window as a stream by incorporating *w:stream-mixin* in its flavor. You can also use other, more specialized windows for more specific purposes:

- Use a window for *output* if you want the user to be able to see information but not change it. Some output windows can be mouse-sensitive; that is, a user can select a portion of the output to display more information, and so on.
- Use a window for *input* if you want to gather information from or otherwise interact with the user. Input windows include notifications, menus, choose variable values windows, and so on.

For Any Window 1.7.1 For any window, there are some decisions you should make before you begin. There are other decisions you should consider after you have created and tested your window.

Before You Begin 1.7.1.1 Before you start building your window, decide whether your application is going to include a command loop. If it is, you should consider using the Universal Command Loop (UCL), described in the *Explorer Tools and Utilities* manual. The UCL offers various menu and help interfaces that make writing code simpler. In addition, by using the UCL, you provide a user interface that is similar to most utility interfaces already on the Explorer system.

After You Finish 1.7.1.2 After you have developed and tested the window for your application, you should consider these questions:

**Choose As Many
As Desired**

Explanation and Reference

Should this window be listed in one of the columns of the System menu?

After you make the window selectable as described in Section 6, Selection, use the `w:add-to-system-menu-column` function described in paragraph 18.6, The System Menu, to add your application to the System menu.

How often is the window used, and how quickly should the window appear?

If this window is used frequently, you may want to create a resource for it. As a resource, the program allocates memory for the window once rather than allocating memory for the window each time it is invoked and then discarding the memory after it is deactivated. If you want to use a resource, examine the list of resources contained in the `w:window-resource-names` variable. Use one of the existing resources if possible; if not, create your own resource using the function described in paragraph 18.7, Window Resources.

Should this window include Suggestions menus?

See the *Explorer Tools and Utilities* manual for a description of Suggestions menus.

-
- Using a Frame** 1.7.2 If your application calls for a frame, use the constraint frame editor (WINIFRED, discussed in Section 15, Frames) to create it. Before you do so, however, you should:
1. Make certain decisions about the frame as a whole.
 2. Decide on the characteristics of each pane by answering the questions in the decision tree for each separate pane.
 3. Finally, use WINIFRED to create the constraint frame.

Questions About the Frame as a Whole

1.7.2.1 Consider each of the following features for the frame as a whole.

Choose As Many As Desired

Explanation and Reference

Should it include a label?

If so, create a flavor that includes one of the constraint frame flavors (described in paragraph 15.4, Constraint Frame Flavors) and `w:label-mixin` (described in paragraph 3.4.2).

Should it include a border?

Use one of the bordered constraint frame flavors described in paragraph 15.4, Constraint Frame Flavors.

Should the panes of the frame share an input buffer?

Use one of the constraint frame flavors that explicitly share an input buffer, as described in paragraph 15.4, Constraint Frame Flavors.

Should the panes of the frame share a selection substitute?

First, be sure that you include `w:select-mixin` as part of your frame flavor. Then, set a selection substitute as described in paragraph 15.10.1, The Selected Pane.

Questions About a Non-Constraint Frame as a Whole

1.7.2.2 If you decide not to use constraint frames, answer the following questions for your frame:

Choose As Many As Desired

Explanation and Reference

Should the screen manager not interfere with inferiors?

If not, use the `w:no-screen-managing-mixin` flavor, described in paragraph 5.9.2, Autoselection.

Should the inferiors of the window not be in the Select menu of the System menu?

If so, use the `w:inferiors-not-in-select-menu-mixin` flavor, described in paragraph 6.3.1, The System Menu Select Command.

**Types of
Output Windows**

1.7.3 Certain features can be included in a window to allow output in a special window or buffer that appears. You can also associate these features with a particular pane of a constraint frame.

From the following questions, choose the one that best describes what you want the window to do.

Choose One**Explanation and Reference**

Is the output simple and relatively brief (only a line or two)?

Use notifications. Include either the **w:notification-mixin** flavor or the **w:delay-notification-mixin** flavor as part of your window, and then use either the **w:careful-notify** function or the **w:notify** function to produce the notification. These flavors and functions are described in paragraph 18.2, Notifications.

Do you want to allow scrolling in only one direction?

Use a typeout window. These windows are typically used to display help and error messages. Use **w:essential-window-with-typeout-mixin** or an associated flavor, described in paragraph 13.1, Using Typeout Windows and in paragraph 13.3, Windows With Inferior Typeout Windows.

Do you want to allow scrolling in both directions?

This type of window displays information that a user may want to look at several times. Use the **zwei:view-file** function.

Do you want a feature similar to Peek, where each line lists a separate item or subitem, but that is not mouse-sensitive?

Use a general scroll window.

- If you *do not* want a typeout window associated with the window, use either the **w:basic-scroll-window** mixin or the **w:scroll-window** flavor, described in paragraph 17.3, Using a Scroll Window.
- If you *do* want a typeout window associated with the window, use either the **w:scroll-window-with-typeout-mixin** or the **w:scroll-window-with-typeout** flavor, described in paragraph 17.3, Using a Scroll Window.

To provide mouse-sensitivity, use the flavors described in paragraph 17.7, Mouse-Sensitive Scroll Windows

Do you want a feature similar to the Inspector, where you can display a number of similar lines with scrolling?

Use a text scroll window, such as the **w:text-scroll-window** mixin (described in paragraph 16.1, Using Text Scroll Windows) or the **w:text-scroll-window-typeout-mixin** flavor (described in paragraph 16.3, Function Text Scroll Windows). If you want to dynamically define functions to display windows, use the **w:function-text-scroll-window** flavor, described in paragraph 16.3, Function Text Scroll Windows.

Do you want a temporary window that displays text?

Use one of the pop-up text window resources or flavors described in paragraph 18.4.3, Window Flavors for Other Programs.

Do you want a window whose lines truncate rather than wrap?

Use either the **w:line-truncating-mixin** or the **w:truncating-window** flavor, described in paragraph 7.4.4, End-of-Line Exceptions.

**Types of
Input Windows**

1.7.4 In general, input windows are of two types: those that simply require confirmation (your application may do a certain action depending on whether the user confirms), and those that actually require the user to make a choice or otherwise supply information. The following functions and flavors typically use the mouse as the primary input device rather than the keyboard, although menus also support keystrokes as well as mouse clicks.

*Obtaining
Confirmation*

1.7.4.1 Do you simply want the user to confirm or not confirm an action? If so, use one of the following:

Choose One

Explanation and Reference

Is the output simple and relatively brief (only a line or two)?

Use notifications. Include either the `w:notification-mixin` flavor or the `w:delay-notification-mixin` flavor as part of your window, and then use either the `w:careful-notify` function or the `w:notify` function to produce the notification. These flavors and functions are described in paragraph 18.2, Notifications.

Should the user just select yes or no with the mouse from a pop-up window?

Use the `w:mouse-y-or-n-p` function, described in paragraph 14.2.4.3, Other Special Functions, of the Functions That Create Menus paragraph. This function requires the user to click on either a yes item or a no item to make the window disappear.

Should the user confirm either by selecting an item with the mouse or by pressing a key?

Use the `w:mouse-confirm` function, described in paragraph 14.2.4.3, Other Special Functions, of the Functions That Create Menus paragraph. This function requires the user, to specify yes, to either click on a yes item or press the END key. To specify no, the user must either click on a no item or press the ABORT key.

Should the confirmation be requested through the Listener or minibuffer rather than through a pop-up window?

See the *Explorer Input/Output Reference* manual for descriptions of functions that obtain confirmation without using a pop-up window, such as the `y-or-n-p` function.

Gathering Information 1.7.4.2 Do you want to obtain information from the user? If so, use one of the following:

- Menus
- Choose variable values windows
- Mouse-sensitive typeout
- Zmacs-like functions

Menus Menus provide a series of choices from which the user can choose. What kind of choices do you want the user to make, and what features do you want to provide?

Choose One

Explanation and Reference

Should the user choose from a menu?

Use the `w:menu-choose` function, described in paragraph 14.2.4.1, The Most General Function, of the Functions That Create Menus paragraph. `w:menu-choose` provides the general features of menus and can be used to produce highlighting, dynamic, multicolumn, permanent or temporary menus. This function also enables you to specify the geometry of a menu, specify an abort item to be executed if the user aborts from a menu, specify alignment of menu items, specify sorting of menu items (alphabetically, ascending or descending, or with another sorting algorithm), and enable scrolling, among others.

You can also use `w:menu`, the analogous resource or flavor, described in paragraph 14.2.6, Flavor and Initialization Options That Define Menus.

Should the user select boxes in columns to indicate choices?

Use the `w:multiple-choose` function, described in paragraph 14.3.1, Multiple-Choice Functional Interface. `w:multiple-choose` lets you specify a label for each column, where the menu appears, how many lines are displayed before scrolling is enabled, and the superior of the menu. This menu is always temporary. An example of this type of menu is the Kill or Save Buffers menu in the Zmacs editor.

If the function does not give you the features you want, try using the `w:temporary-multiple-choice-window` resource or flavor, the `w:multiple-choice` flavor, or the `w:basic-multiple-choice` mixin flavor. These flavors are described in paragraph 14.3.2, Making Your Own Multiple-Choice Windows.

Choose-Variable-Values Windows Choose-variable-values windows allow a user to supply values for various variables. Depending on how you program the window, the user can choose one of a list of values, toggle between yes and no, choose from a menu, or enter a new value. You can also program the window to accept only values within a certain limit or values of a particular type. A choose-variable-values window can be either temporary or permanent.

For most purposes, you should be able to use the `w:choose-variable-values` function, described in paragraph 14.4.2, Choose-Variable-Values Functional Interface. If this function does not provide the features you want, you can use one of the flavors described in paragraph 14.4.4, Making Your Own Window, of the Choose-Variable-Values Facility paragraph.

Mouse-Sensitive Typeout Some typeout can be mouse-sensitive, such as the List Buffers display of Zmacs. This enables the user to choose an item with the mouse after the item is displayed. Use one of the following features to provide mouse-sensitive typeout:

Choose One	Explanation and Reference
<i>Is the output brief enough that you can draw it using a format statement?</i>	Use the <code>-M</code> format directive, described briefly in paragraph 14.5.2. This feature provides mouse sensitivity within the output of a <code>format</code> statement. For more information about the general features of a <code>format</code> statement, see the <i>Explorer Lisp Reference</i> manual.
<i>Do you want a feature similar to Peek, where each line can be described separately?</i>	Use a general scroll window (described in Section 17) with either <code>w:essential-scroll-mouse-mixin</code> or <code>w:scroll-mouse-mixin</code> as a component of the flavor. If you actually want a Peek window, use the <code>tv:peek-frame</code> flavor.
<i>Do you want a feature similar to the Inspector?</i>	Use the <code>w:mouse-sensitive-text-scroll-window</code> flavor (described in Section 16). If you actually want an Inspector window or frame, use either the <code>w:inspect-frame-resource</code> or the <code>w:inspect-frame</code> flavor.

Zmacs-Like Functions The following are specialized functions that offer features similar to features in the Zmacs editor:

Choose One	Explanation and Reference
<i>Is there a string that the user should edit?</i>	Use the <code>zwei:pop-up-edstring</code> function, described in paragraph 18.4.3, Window Flavors for Other Programs.
<i>Are you asking the user for a pathname that could be partially supplied by default?</i>	Use the <code>zwei:read-defaulted-pathname-near-window</code> function, described in paragraph 18.4.3, Window Flavors for Other Programs.

Types of Standalone Windows

1.7.5 The following are some of the kinds of standalone windows that are available. You can either use an instantiation of the flavor by itself, or you can mix other flavors with these windows to provide particular features.

Choose One

Explanation and Reference

Do you want a window that is the simplest available?

Use `w:minimum-window`, described in paragraph 2.4, Basic Window Flavors.

Do you want a window that allows you to use fonts, perform graphics operations, and create labels and borders?

Use `w>window`, described in paragraph 2.4, Basic Window Flavors.

Do you want a window similar to the one just described but whose lines truncate rather than wrap?

Use `w:truncating-window`, described in paragraph 7.4.4, End-of-Line Exceptions.

Do you want a Lisp Listener?

Use `w:lisp-listener` for a Lisp Listener than can be selected with the SYSTEM L keystroke sequence. Use `w:lisp-interactor` for a Lisp Listener than *cannot* be selected with the SYSTEM L keystroke sequence. These flavors are described in paragraph 18.4.1, Lisp Listeners.

Do you want a Peek window?

Use the `tv:peek-frame` flavor (described in paragraph 17.8, Peek Flavors). Similar features are available with a general scroll window; use either `w:essential-scroll-mouse-mixin` or `w:scroll-mouse-mixin` as a component of the flavor. These flavors are described in paragraph 17.7, Mouse-Sensitive Scroll Windows.

Do you want an Inspector frame?

Use either the `w:inspect-frame-resource` or the `w:inspect-frame` flavor (described in paragraph 16.5, Inspector Flavors). Similar features are available with a text scroll window; use the `w:mouse-sensitive-text-scroll-window` flavor, described in paragraph 16.5, Mouse-Sensitive Text Scroll Windows.

Do you want a Telnet window?

Use either the `supdup:telnet` flavor or the `supdup:telnet-windows` resource, described in paragraph 18.4.3, Window Flavors for Other Programs.

Do you want a Zmacs editor?

- For a Zmacs editor that uses the same process as all Zmacs windows in the system, use the `zwei:zmacs-frame` flavor.
- For a Zmacs editor with a separate process and a permanently visible minibuffer, use the `zwei:standalone-editor-window` flavor.
- For a Zmacs editor with a separate process that pops up a mode line or a minibuffer when needed, use the `zwei:standalone-editor-frame` flavor.

All these flavors are described in paragraph 18.4.2, Editor Windows.

Types of Mixins 1.7.6 Using the following questions, decide what features you want for your window or pane. Note each feature that you want. When you have identified all the features, create your flavor using those mixins or initialization options.

Choose As Many As Desired

Explanation and Reference

Include borders?

- To specify the width of a border, use the `:borders` initialization option or the `:set-borders` method, described in paragraph 3.3, Borders.
- To specify the width of border margins, use the `:border-margin-width` initialization option or the `:set-border-margin-width` method, described in paragraph 3.3, Borders.
- To use a fancy border (one that is not a simple rectangle), create a special drawing function. This process is described in paragraph 3.3.1, Border Functions.

Don't include borders?

If your window is the full size of the main screen and you do not want borders, include the `w:full-screen-hack-mixin` flavor, described in paragraph 3.3.2, Deleting Borders on Full-Screen Windows.

Symbols related to borders

Constraint frames have their own flavors and mixins that enable or disable borders.

Include labels?

- The default label for the window is its name with a numeric counter. You can set a window name by using the `:name` initialization option, described in paragraph 3.4.1, Names of Windows.
- To specify a label of a particular size, font, vertical spacing, or position, use the `:label` initialization option or the `:set-label` method, described in paragraph 3.4.1, Names of Windows.
- To position a label automatically, use the appropriate keyword for the `:label` initialization option or use either the `w:top-label-mixin` or the `w:centered-label-mixin` flavor. The mixins are described in paragraph 3.4.3, Positioning the Label.
- By default, the label of a window is not boxed. To box the label for the window, use one of `w:box-label-mixin`, `w:top-box-label-mixin`, or `w:centered-box-label-mixin` flavor (all described in 3.4.4, Boxing the Label), depending on where the label should appear.

Symbols related to labels.

Labels on menus and temporary windows are usually defined by a keyword or argument of the functional interface or flavor. See the appropriate function or flavor for the window you are creating.

Include a bit-save array?

Include the `:save-bits` initialization option or the `:set-save-bits` method, described in paragraph 5.6, Bit-Save Arrays.

Include shadow borders?

Shadow borders are typically used only on temporary windows. To include them on your window, use the `w:shadow-borders-mixin`, described in paragraph 5.8.1, Flavors and Methods, of the Temporary Windows paragraph.

Continued

(Continued)
Choose As Many
As Desired

Explanation and Reference

What should happen if the window is not fully visible?

Typically, a window that is not fully visible is displayed if it or one of its superiors has a bit-save array. However, you can also choose one of the following behaviors:

- To display the contents of the partially visible window that does not have a bit-save array, use the **w:show-partially-visible-mixin**.
- To display only the borders of a window with the inside of the window a shade of gray, use either the **w:gray-deexposed-right-mixin** or the **w:gray-deexposed-wrong-mixin**.
- To never display the window (even it could become partially visible), until it is first explicitly exposed, use the **w:initially-invisible-mixin**.

These mixins are all described in paragraph 5.9.3, Control of Partial Visibility.

Be selectable?

The user cannot select a window unless you make that window selectable. In general, you can choose to make a window selectable in one of three ways:

- To make the window selectable in and of itself, use the **w:select-mixin**, described in paragraph 6.2, How Programs Select Windows.
- To make a window selectable by selecting its superior, use **w:not-externally-selectable-mixin**, described in paragraph 6.3.2, Selection With TERM and SYSTEM Keys.
- To make a window serve as the selected window for all its inferiors, use **w:alias-for-inferiors-mixin**, described in paragraph 6.3.2, Selection With TERM and SYSTEM Keys.

The last two choices are called selection substitutes.

Be associated with a process?

To associate a process with a window, use **w:process-mixin**, described in paragraph 6.6.2, Process-Related Methods and Flavors.

Should the process, if any, be reset when it tries to write on a window that is not exposed?

Use **w:reset-on-output-hold-flag-mixin**, described in paragraph 6.6.2, Process-Related Methods and Flavors.

Should the window accept input?

Use **w:stream-mixin**, described in paragraph 8.2, Input Buffers. Typically, input windows are appropriate only for windows such as I/O buffers. For windows that can be less restricted than interaction windows or I/O buffers, consider using a temporary input window.

Use line truncation rather than wrapping?

Use **w:line-truncating-mixin**, described in paragraph 7.4.4, End-of-Line Exceptions.

Continued

(Continued)
Choose As Many
As Desired

Explanation and Reference

What kind of mouse handling?

When the user clicks the mouse, you can specify whether the system sees a mouse click or a mouse blip (a list that includes the click, the mouse position, and the window).

- To include a mouse click, use `w:kbd-mouse-buttons-mixin`.
- To include a mouse blip, use `w:list-mouse-buttons-mixin`.

Both of these flavors are described in paragraph 11.4.2, Encoding Mouse Clicks as Characters.

Should the window scroll?

Use `w:scroll-bar-mixin`, described in paragraph 11.8, Scrolling.

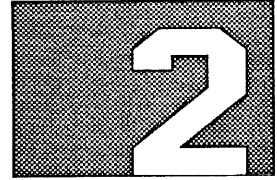
Include graphics?

Use `w:graphics-mixin` for drawing simple graphics. Use the GWIN flavors for graphic objects and more complicated graphics. These flavors are described in paragraph 12.4, Methods and Flavors to Draw Graphics Images.

If the window is smaller than the main screen, should it be greedy about giving up the mouse?

If so, use the `w:hysteretic-window-mixin` and set the sensitivity of relinquishing the mouse with either the `:hysteresis` initialization option or the `:set-hysteresis` method, described in paragraph 11.5, Ownership of the Mouse.

BASIC WINDOWS



Introduction

2.1 This section briefly describes some of the basics underlying the Explorer window system: its package structure, how to create an instance of a window, and the basic window flavors.

Window System Packages

2.2 The window system code is in three packages:

- The W package contains new code for this release and inherits code from the TV package.
- The TV package contains code from earlier releases.
- The GWIN package contains the code that handles the graphics objects and worlds.

Some symbols have definitions in both the W and the TV packages. In these cases, you should use the code in the W package; it is updated and typically more efficient to use. If desired, you can access the version of the symbol defined in the TV package by including an explicit package prefix. (For example, invoking `tv:menu`.)

Typically, you can avoid explicitly specifying a package name within code by including the appropriate package name in the file attribute line. Thus, within a file that contains code in the W package, you can omit all the W and TV package prefixes except when you want to use the obsolete (that is, the TV) version of a symbol.

Creation of Windows

2.3 When you want to create a flashy and sophisticated user interface, especially involving mouse sensitivity or automatic updating, you should consider creating your own windows (and your own window flavors, perhaps).

To create a window, you should use the `make-instance` function.

`make-instance` *flavor-name* &rest *initialization-options* Function

Creates, initializes, and returns a new instance of the specified flavor. The *initialization-options* argument contains alternating keywords and values; the keywords must be initialization options accepted by the flavor you are using. The initialization options accepted by various window flavors are described in this manual.

When executed, this example prompts you for the upper left and lower right corners of the window, then creates an instance of a Lisp Listener with large characters and lots of vertical space between lines.

```
(setq my-window (make-instance 'w:lisp-listener
                              :borders 4
                              :font-map (list fonts:bigfnt)
                              :vsp 6
                              :edges-from :mouse))
```

However, this code only creates an instance; it does not expose or select that instance. (Technically, the window instance is *deactivated*.) To select the window, you must send a `:select` message to the window instance, as follows:

```
(send my-window :select)
```

You can also expose the window instance at the time of creation by including an `:expose-p` initialization option with a value of `t`, as shown in the following code:

```
(setq my-window (make-instance 'w:lisp-listener
                              :borders 4
                              :font-map (list fonts:bigfnt)
                              :vsp 6
                              :edges-from :mouse
                              :expose-p t))
```

For more information on the `make-instance` function, see the *Explorer Lisp Reference* manual.

You can also obtain a window from a resource rather than explicitly creating your own. (A *resource* is a pool of interchangeable objects that can be used temporarily and then returned to the pool.) Windows as resources are explained more fully in paragraph 18.7, Window Resources.

**Basic Window
Flavors**

2.4 All window flavors are ultimately based on the **w:minimum-window** flavor, which includes the **w:sheet** flavor component. A simple instantiable flavor often used for testing is the **w:window** flavor. Applications typically use more special-purpose windows than instances of **w:window**.

w:minimum-window

Flavor

Provides the minimum functionality needed for a window. All existing window flavors are built on the **w:minimum-window** flavor; you should include this component or a mixin that contains this component in any window flavors that you define. This flavor itself is made of the following components:

w:essential-window
w:essential-activate
w:essential-expose

w:essential-set-edges
w:essential-mouse

The **w:minimum-window** flavor has no methods of its own; all are inherited from these components. At times—such as in the debugger—you may encounter methods of these component flavors. You may also encounter methods of **w:sheet**, a component of **w:essential-window**. As a programmer, you usually need not pay attention to the distinctions between these flavors. In this manual, all the methods, instance variables, and initialization options of **w:minimum-window** are documented as being of *windows* rather than of any specific flavor.

w:sheet

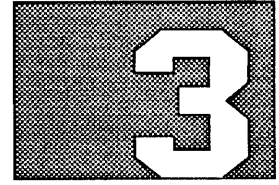
Flavor

Provides the structure required by the low-level display primitives. **w:sheet** is a component of **w:essential-window**. In this manual, methods defined by **w:sheet** are documented as being of *windows and screens*.

w:window

Flavor

Includes several mixins that provide most of the generally useful features, including the ability to use fonts, perform graphics operations, and create labels and borders. This flavor is normally used only for testing purposes because it contains everything but the kitchen sink.



OUTSIDE EDGES OF WINDOWS

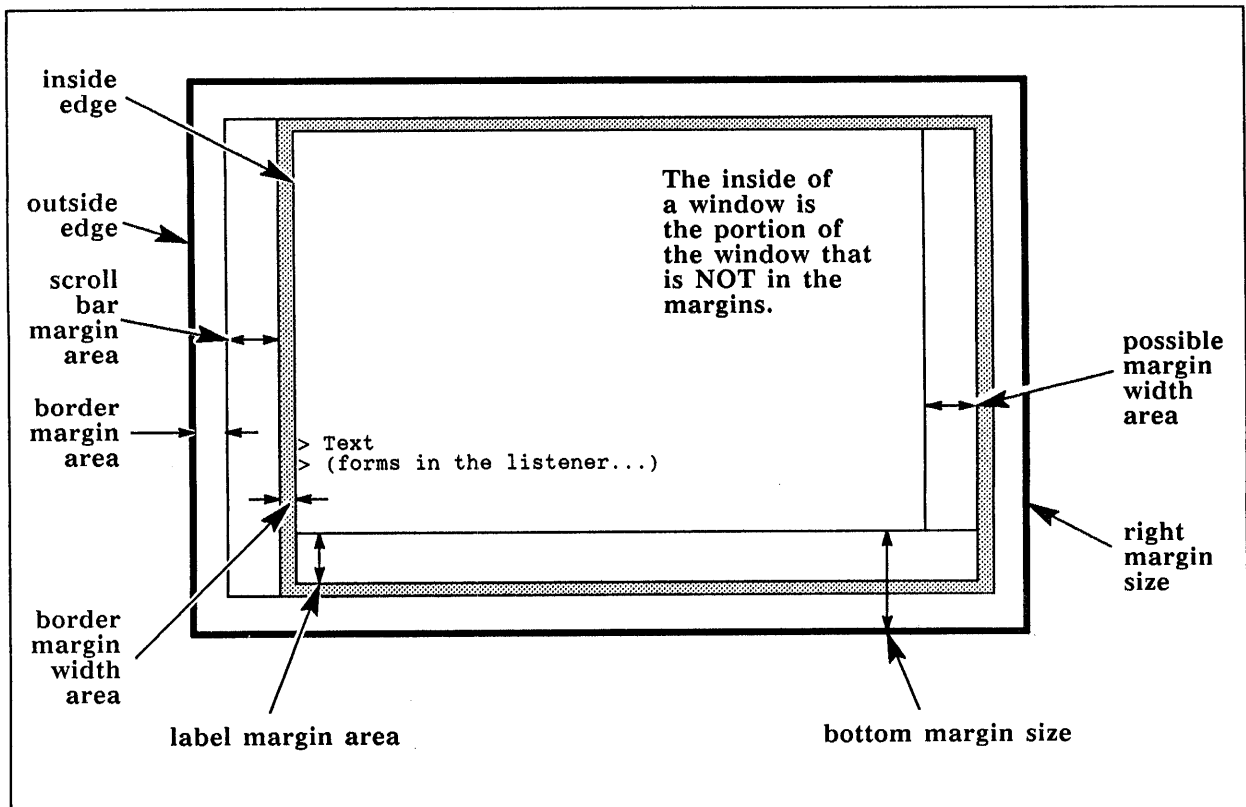
Introduction

3.1 *Margins* are located at the edges of a window. Each edge of a window has a separate margin. Each margin can contain an item such as the following:

- A *border*, a rectangular box drawn around the window. Borders delineate where a window is drawn on the screen.
- A text string called a *label*, which helps identify the window. Although labels can be inside or outside the borders, the label is usually inside.

These margin items are discussed in this section. Another typical margin item is a scroll bar.

To prevent text from being obscured by the border or by margin items, windows leave a small space—typically one pixel wide—between the border and the text. The width of this space is called a *border margin width*.



Margins

3.2 In a window, margins are the area between the *inside* and *outside* edges of the window. Margins may contain margin regions, which in turn may contain margin items. In any flavor of window, one of the margin items is the innermost; this item is the item closest to the inside part of the window. Each successive margin item is outside the previous one; the last margin item is just inside the edges of the window. Each margin item is created by mixing in a flavor. The flavors you mix in and the order that the flavors are mixed in control what items are in the margins and the order of the items. Margin item flavors closer to the beginning of the component flavor list are closer to the outside of the margins. The `w:window` flavor has `w:borders-mixin` and `w:label-mixin` for components, in that order, so the label is inside the border.

`:margins`

Method of *windows*

Returns four values: the sizes—measured in pixels—of the left, top, right, and bottom margins, respectively. Each value is the sum of the sizes of the borders, labels, and anything else that pertains to that margin. For a window with no margins, all four values are zero.

`:left-margin-size`

Method of *windows*

`w:sheet-left-margin-size window`

Macro

`:top-margin-size`

Method of *windows*

`w:sheet-top-margin-size window`

Macro

`:right-margin-size`

Method of *windows*

`w:sheet-right-margin-size window`

Macro

`:bottom-margin-size`

Method of *windows*

`w:sheet-bottom-margin-size window`

Macro

Returns the size of the respective margin in pixels. There are no methods to set these sizes nor initialization options to initialize them. The margin sizes are always computed from the labels, borders, and other margin items as described later in this section.

The macros return the value of the respective sizes of *window*. The `:outside-accessible-instance-variables` option of the `defflavor` function creates these macros.

Borders

3.3 A *border* is the rectangular box drawn around a window to show where the edges of the window are located.

Borders also include space left between the borders and the inside of the window. The thickness of this space is called the *border margin width*. This space prevents characters and graphics next to the edge of the inside of the window, or the next-innermost margin item, from merging with the border.

You can control the thickness of each of the four borders separately or of all of them together. You can also specify your own function to draw the borders if you want something more elaborate than simple lines.

If your application is running in a color environment, you can also set the color of the border by using the `:border-color` operation, as explained in paragraph 19.4, Initialization Options and Methods Used With Color Windows.

w:borders-mixin

Flavor

Creates the borders around windows that you often see when using the Explorer system.

:borders *argument*Initialization Option of **w:borders-mixin**

Gettable, settable. Default: 1.

Initializes the **w:borders** instance variable to the parameters of the borders. *argument* can have any of the following values:

- **nil**, which means there are no borders at all.
- Symbols or numbers that apply to each of the four borders. These can be either of the following:
 - A symbol — The function that draws and defines the default thickness of the border in pixels.
 - A number — The thickness of the border in pixels.
- A list of the form (*left top right bottom*) that specifies each of the borders at the four edges of the window. The items of the list can be any of the following:
 - A symbol — The function that draws and defines the default thickness of the border in pixels.
 - A number — The thickness of the border in pixels.
 - **nil** — The edge should not have a border.
 - **t** — The border should be drawn by the default function that also defines the thickness of the border in pixels.
- A list of the form (*keyword1 spec1 keyword2 spec2...*) — The borders at the edges selected by the keywords, which can be **:left**, **:top**, **:right**, and **:bottom**.
- A cons (*function . thickness*) — The function that draws the border and the thickness the border will be.

Note that setting border specifications to a border width of 0 is not the same as giving **nil** as the argument to this option. When 0 is the specified border width, space is allocated for the border margin width. When **nil** is the specified border width, space is not allocated.

:border-margin-width *n-pixels*Initialization Option of **w:borders-mixin**

Gettable, settable. Default: 1.

Initializes the **w:border-margin-width** instance variable to the width of the white space in the margins between the borders and the inside of the window. *n-pixels* is specified in pixels. If the specification for the border is **nil**, then **:border-margin-width** does not produce a border margin, regardless of the value of *n-pixels*; if the specification for the border is 0, however, **:border-margin-width** does produce a border margin.

Border Functions 3.3.1 `w:draw-rectangular-border` is the only border function supplied by the system.

`w:draw-rectangular-border` *window alu left top right bottom* Function
 Draws a one-pixel wide border around a window by default.

To define your own border function, create a Lisp function that takes six arguments: the window on which to draw the label, the ALU argument used to draw the label, and the left, top, right, and bottom edges of the area that the border should occupy. The returned value is ignored. The function runs inside the `w:sheet-force-access` macro. Place a `w:default-border-size` property on the name of the function whose value is the default thickness of the border; the default thickness is used when a specification is a non-nil symbol.

If you specify a border function in the argument to the `:borders` message, you should specify both the name of the function and its width, as in the following example that uses the existing border function:

```
(make-instance 'w:window
  :expose-p t
  :edges-from :mouse
  :borders '((w:draw-rectangular-border . 8)
            (w:draw-rectangular-border . 6)
            (w:draw-rectangular-border . 4)
            (w:draw-rectangular-border . 2))
  :name "Text Window"
  :label :top
)
```

This code prompts you to specify the upper left and lower right corners of the window, then creates and exposes it with a top border 8 pixels thick, a left border 6 pixels thick, a bottom border 4 pixels thick, and a right border 2 pixels thick.

Deleting Borders on Full-Screen Windows 3.3.2

`w:full-screen-hack-mixin` Flavor
 Required flavor: `w:borders-mixin`

Offers the user the option of requesting that these windows have no borders when they occupy the full screen. This flavor is included in many system flavors, such as Lisp Listeners and Zmacs frames.

`w:flush-full-screen-borders` &optional (*flush-p* t) Function
 Eliminates (when *flush-p* is t) or reinstates (when *flush-p* is nil) the borders of all full-screen-sized windows that use the `w:full-screen-hack-mixin` flavor.

Labels

3.4 A *label* is a text string within the margin of a window. It is possible to have more than one line of text within the margin. The default for the label of a window is its name. For example, the label of the initial Lisp Listener is Lisp Listener 1.

Names of Windows 3.4.1**:name** *name*Initialization Option of **w:minimum-window***Gettable*. Default: The flavor name and a counter ID

Sets the name of a window to *name*, which can be either a symbol or a string. The main use of the name is for the default value for the label, if there is a label.

The w:label-mixin Flavor 3.4.2**w:label-mixin**

Flavor

Creates the labels in the margins of windows that you often see when using the Explorer system. **w:label-mixin** allows you to control the text of the label, the font in which the label is displayed, and whether the label appears at the top or bottom of the window.

w:labelInstance Variable of **w:label-mixin**

Describes the label of the window. It is either **nil** (for no label) or a list of length eight whose elements are one of the following:

- **w:label-left**, **w:label-top**, **w:label-right**, and **w:label-bottom** — The rectangle allocated to the label. All four edges are relative to the window's outside upper left corner.
- **w:label-font** — The font to use for the label.
- **w:label-string** — The string to display in the label.
- **w:label-vsp** — The separation between lines in the label.
- **w:label-centered** — Whether the label should be centered. When **w:label-centered** is non-**nil**, the label text is horizontally centered.

:label-sizeMethod of **w:label-mixin**

Returns the width and height, in pixels, of the area occupied by the label.

:label *specification*Initialization Option of **w:label-mixin***Gettable, settable*.

Sets the string displayed as the label, the font in which the label is displayed, and whether the label is at the top or the bottom of the window. If you do not specify every parameter, the parameters not specified default to standard values for **:label**. The **:set-label** method overwrites any previous specifications. Any parameters not specified use the default values.

By default, the string is the same as the name of the window, the font is the screen's standard font for `:label`, and the label is at the bottom of the window.

specification can be one of the following:

- `nil` — No label.
- `t` — The label is given all the default characteristics.
- `:top` — The label is put at the top of the window.
- `:bottom` — The label is put at the bottom of the window.
- A string — The text displayed in the label.
- A font — The font for the text displayed in the label.
- A list of the form (*keyword1 arg1 keyword2 ...*) — The attributes corresponding to the keywords are set; the rest of the attributes default. For keywords that do not require arguments, you do not supply one.

The following keywords can be given:

Keyword and Argument	Description
<code>:top</code>	The label is put at the top of the window.
<code>:bottom</code>	The label is put at the bottom of the window.
<code>:centered</code>	The label is printed horizontally centered, rather than starting at the left edge.
<code>:string <i>string</i></code>	The text in the label is the text specified by <i>string</i> .
<code>:font <i>font-specifier</i></code>	The font used for the text in the label is specified by <i>font-specifier</i> , which can be any font specifier.
<code>:vsp <i>n-pixels</i></code>	When the label has multiple lines, the number of pixels separating the lines is specified by <i>n-pixels</i> .
<code>:color <i>color</i></code>	In a color environment, this is the foreground color of the label (that is, the color of the text of the label).
<code>:background <i>color</i></code>	In a color environment, this is the background color of the label (that is, the color behind the text of the label).

For example, consider the following labels:

(label-example2)

This is a string.
DEFAULT-LABEL

This is a string.█
Window 10

This is a string.█
A string label

This is a string that extends
over several lines

```
(defun label-example2 ()
  (setq l1 (make-instance 'w:window
    :label 'default-label
    :top 50 :left 50
    :height 50 :width 300))
  (setq l2 (make-instance 'w:window
    :label fonts:bigfnt
    :top 125 :left 50
    :height 50 :width 300))
  (setq l3 (make-instance 'w:window
    :label "A string label"
    :top 200 :left 50
    :height 50 :width 300))
  (setq l4 (make-instance 'w:window
    :label ; A list of keyword-value pairs
    '(:centered
      :string "A label that extends
over several lines" ; Note the explicit carriage return
      :font fonts:bigfnt
      :vsp 5)
    :top 275 :left 50
    :height 50 :width 300))
  (send l1 :expose)
  (send l1 :string-out "This is a string.")
  (send l2 :expose)
  (send l2 :string-out "This is a string.")
  (send l3 :expose)
  (send l3 :string-out "This is a string.")
  (send l4 :expose)
  (send l4 :string-out "This is a string.")
  )
```

Positioning the Label 3.4.3

w:top-label-mixin

Flavor

Causes the label to appear in the top margin of the window by default instead of at the bottom. The **w:top-label-mixin** flavor does not override an explicit specification of the label position.

w:centered-label-mixin

Flavor

Required flavor: **w:label-mixin**

Centers the label string horizontally within the width of the window. Unlike **w:top-label-mixin**, the **w:centered-label-mixin** flavor *does* override an explicit specification of the label position.

Boxing the Label 3.4.4 These boxing mixins make the label appear to be in a box by drawing a line, one pixel in width, just on the inside of the label. The other three sides of the box are drawn by the window's borders. Because the existence of a label adjusts the size of the margin and thus the border, the calculations for the border must be made before the adjustment for the label. Therefore, the **w:borders-mixin** must precede the various boxing mixins or the label appears on top of the border.

w:box-label-mixin

Flavor

Required flavor: **w:label-mixin**

Makes the label appear to be in a box by drawing a line just on the inside of the label. This combines with the window's borders, which surround the other three sides of the label, to make a box. The extra line is present only if the

label is turned on. Menus use `w:box-label-mixin`. For example, any menu you get from the Split Screen option in the System menu shows the label with a box around it.

`:label-box-p` *t-or-nil* Initialization Option of `w:box-label-mixin`
 Default: `t`

Puts a box around the label if *t-or-nil* is `t`. If this option is `nil`, the box around the label is inhibited.

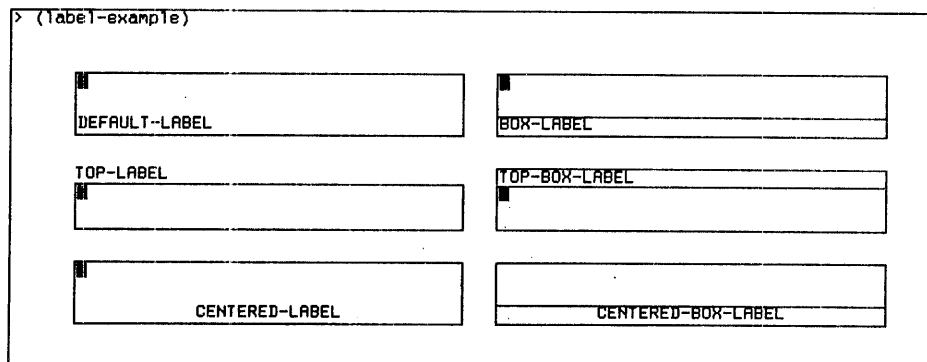
`w:top-box-label-mixin` Flavor
`w:bottom-box-label-mixin` Flavor

Makes the label appear to be in a box, either at the top or the bottom of the window, respectively, by drawing a line just on the inside of the label. These flavors must be combined with the `w:borders-mixin` flavor.

For example, consider the following labels:

```
; The first window uses the default label for w:window, so it does not have a separate defflavor.
(defflavor wd-top-label () (w:top-label-mixin w:window)
 (:default-init-plist :height 50 :width 300))
(defflavor wd-centered-label () (w:centered-label-mixin w:window)
 (:default-init-plist :height 50 :width 300))
(defflavor wd-bottom-box-label () (w:borders-mixin w:bottom-box-label-mixin w:window)
 (:default-init-plist :height 50 :width 300))
(defflavor wd-top-box-label () (w:borders-mixin w:top-box-label-mixin w:window)
 (:default-init-plist :height 50 :width 300))
(defflavor wd-centered-box-label () (w:borders-mixin w:box-label-mixin
 w:centered-label-mixin w:window)
 (:default-init-plist :height 50 :width 300))

(defun label-example ()
  (setq w1 (make-instance 'w:window
                          :label 'default-label
                          :top 50 :left 50
                          :height 50 ; For the others, these options
                          :width 300) ; are in the default-init-plist.
        w2 (make-instance 'wd-top-label
                          :label 'top-label
                          :top 125 :left 50)
        w3 (make-instance 'wd-centered-label
                          :label 'centered-label
                          :top 200 :left 50)
        w4 (make-instance 'wd-bottom-box-label
                          :label 'box-label
                          :top 50 :left 375)
        w5 (make-instance 'wd-top-box-label
                          :label 'top-box-label
                          :top 125 :left 375)
        w6 (make-instance 'wd-centered-box-label
                          :label 'centered-box-label
                          :top 125 :left 375))
  (send w1 :expose) (send w2 :expose) (send w3 :expose)
  (send w4 :expose) (send w5 :expose) (send w6 :expose)
  )
```



Delaying Redisplay of a Label 3.4.5

w:delayed-redisplay-label-mixin

Flavor

Required flavor: **w:label-mixin**

:delayed-set-label *specification*

Method of **w:delayed-redisplay-label-mixin**

:update-label

Method of **w:delayed-redisplay-label-mixin**

Adds the **:delayed-set-label** and **:update-label** methods to your window. The **:delayed-set-label** method changes the label in such a way that the label is not actually displayed until the **:update-label** method executes. This is especially useful for programs that suppress redisplay when there is type-ahead; the user's commands can change the label several times, and you may want to suppress the redisplay of the changes in the label until there is no type-ahead. Paragraph 8.6.1, I/O Buffers and Type-Ahead, discusses type-ahead.

w:sheet-label-needs-updating *window*

Macro

Determines whether the **:update-label** method should update the label. **w:label-needs-updating** is non-nil if **:delayed-set-label** has executed but the updated label has not been displayed yet. The **:outside-accessible-instance-variables** option of the **defflavor** function creates this macro.

Margin Regions

3.5 *Margin regions* are a general facility for allocating space in a window's margin for specific purposes. Each region can display text or graphics and can be mouse-sensitive. Margin choices are implemented using margin regions.

w:margin-region-mixin

Flavor

Required flavors: **w:essential-mouse**, **w:essential-window**

Gives a window the ability to have margin regions. This allows the window to have separate mouse handling in parts of the margins.

:set-region-list *new-region-list*

Method of **w:margin-region-mixin**

Sets the list of margin regions. The new list should be a list of margin region descriptors as described under **w:region-list**, but only the first three elements of each descriptor need be filled in. The rest are set up automatically.

:region-list *list-of-descriptors*

Initialization Option of **w:margin-region-mixin**

Sets the list of margin region descriptors. Each descriptor specifies one margin region and is a list of this form:

(function margin size left top right bottom)

The list can be longer than seven, but you must define the meaning of the extra elements. The following table gives the meaning of the seven standard elements and names of the macros provided to access them. Each macro

takes the current region as an argument and returns the margin region object for the current region.

Standard Element	Macro	Description of Macro
<i>function</i>	<code>w:margin-region-function</code>	Handles various operations on the margin region. It is called with a method name as the first argument, so it could be a flavor instance, but no flavors are predefined for the purpose. See paragraph 3.5.1, About <code>w:margin-region-function</code> , for more details.
<i>margin</i>	<code>w:margin-region-margin</code>	The name of the margin that this region lives in; either <code>:left</code> , <code>:top</code> , <code>:right</code> , or <code>:bottom</code> .
<i>size</i>	<code>w:margin-region-size</code>	The thickness in pixels of the margin region, perpendicular to the edge it is next to. (The other dimension is controlled by the size of the window, which can be diminished by space already reserved for other margin items.)
<i>left, top, right, and bottom</i>	<code>w:margin-region-left</code> , <code>w:margin-region-top</code> , <code>w:margin-region-right</code> , and <code>w:margin-region-bottom</code>	The edges of the rectangle assigned to the margin region. If positive, they are relative to the outside upper left corner of the window. If negative, they are relative to the outside lower right corner. These values are computed by the <code>:redefine-margins</code> method, which divides the margin space; they are recorded here so that the margin region can be displayed and found by the mouse.

For example, the `:mouse-click` method of `w:margin-region-mixin`, shown following, processes mouse clicks other than double-click right.

```
(defmethod (margin-region-mixin :mouse-click (button x y)
  (cond ((and current-region (not (= button #\mouse-r-2))))
    (funcall (margin-region-function current-region) :mouse-click x y
              current-region
              button)
    t)))
```

The margin region descriptor can be longer than seven. Additional elements are not used by `w:margin-region-mixin` and therefore can be used by higher-level facilities to record their own information with each margin region.

`w:margin-region-area` *descriptor*

Function

Returns the four edges of the rectangle allocated to the margin region specified by the *descriptor* argument, all relative to the window's outside upper left corner. The `w:margin-region-area` function can only be used inside of methods of the window whose margin region is being operated on. For example:

```
(defmethod my-flavor :my-method ()
  (multiple-value-bind (left top right bottom)
    (margin-region-area
      (assoc 'my-region-function region-list :test #'eq))
    ...
```


About
w:margin-region-
function

3.5.1 The **w:margin-region-function** macro, mentioned in the table on the preceding page, handles various operations on the margin region.

The *function* of a margin region should handle the following methods (keywords). For all of these, the *descriptor* argument is described above in the discussion of **w:region-list**.

- **:refresh** *descriptor* — The **:refresh** method should draw this region on the screen in the position specified.
- **:mouse-enters-region** *descriptor* — Moving the mouse blinker into a region invokes the **:mouse-enters-region** method.
- **:mouse-leaves-region** *descriptor* — Moving the mouse blinker out of a region invokes the **:mouse-leaves-region** method.
- **:mouse-moves** *x y descriptor* — Moving the mouse blinker while within a region invokes the **:mouse-moves** method. Moving the mouse blinker into a region also invokes the **:mouse-moves** method after invoking the **:mouse-enters-region** method. The *x* and *y* arguments specify the new mouse blinker position, relative to the outside of the window.
- **:mouse-click** *x y descriptor mouse-char* — With the exception of double-click-right, clicking a mouse button on a region invokes the **:mouse-click** method. If the method does nothing, the mouse click has no effect. The argument *mouse-char* is like that of the **:mouse-click** window method.
- **:who-line-documentation-string** *descriptor* — This method returns a string or list for the mouse documentation window explaining the meaning of mouse clicks when a mouse button is clicked on the item where the mouse blinker is located. The structure of the list is described with the **:who-line-documentation-string** method in paragraph 11.6, How Windows Handle the Mouse.

Usually, the function is defined to have a **case** statement with each element being the same as a method name. The margin region descriptor itself is always one of the arguments, to identify the region being operated on.

For example, consider the following code. Note that this code does not implement all the needed code for line areas.

```
(defun line-area-region (op &optional ignore y ignore bd)
  (declare (:self-flavor w:line-area-text-scroll-mixin))
  (case op
    ((:refresh :mouse-moves) nil)
    (:mouse-enters-region
     ;; Change the mouse to a rightward pointing arrow.
     (w:mouse-set-blinker-definition :character 13. 6. :on
                                     :set-character
                                     w:mouse-glyph-thick-right-arrow))
    (:mouse-leaves-region
     ;; Leaving the line area region; change the mouse back.
     (w:mouse-standard-blinker))
    (:mouse-click
     ;; If the mouse is near an item, send a mouse blip indicating
     ;; which item it is.
     (let (item line)
       (if (and (>= y (w:sheet-inside-top))
                (setq line (+ top-item (w:sheet-line-no nil y)))
                (< line (array-active-length items))))
         (progn
          (setq item (aref items line))
          (send self :force-kbd-input
                  `(:line-area ,item ,self ,bd)))
          ;; else
          (beep))))
    (:who-line-documentation-string
     ;; Display default mouse documentation for line areas.
     (send self :line-area-mouse-documentation))))
```

Defining Margin Item Flavors

3.5.2 Assume that you want to define something called a *frobboz* that goes in a window's margins in the same way that labels and borders do. You create a flavor called `frobboz-margin-mixin` that implements the feature.

Your `frobboz-margin-mixin` flavor should have certain instance variables, which will be used only by the methods of `frobboz-margin-mixin`, so the instance variables' precise format is up to you. The following examples describe these instance variables:

- The `current-frobbozes` instance variable should be the specification of what frobbozes this window should have. It might record text to display for the frobbozes, a font to use, and so on.
- The `frobboz-margin-area` instance variable should record the rectangle within the window where the frobbozes should go. Everything that deals with the location of the frobbozes on the screen should act on the basis of the value of this variable. It is recommended that you use a list of four values: the left, top, right, and bottom edges of the rectangle, all relative to the upper left outside corner of the window.

Some margin mixins have only a single variable whose value is a list containing both the contents and the position of the margin item.

The following example shows how to define your `frobboz-margin-mixin` flavor. The `:before :init` method verifies the values of the `current-frobbozes` instance variable.

```
(defflavor frobboz-margin-mixin
  ((current-frobbozes nil) frobboz-margin-area)
  ()
  (:required-flavors w:minimum-window)
  (:inittable-instance-variables current-frobbozes))

(defmethod (frobboz-margin-mixin :before :init) (ignore)
  (setq current-frobbozes
    (canonicalize-and-validate-frobboz-spec current-frobbozes)))
```

Now you must create methods for at least two standard operations to perform margin computation and display, and to interface `frobboz-margin-mixin` to the rest of the system. These methods are `:compute-margins` and `:refresh-margins`. You also may wish to provide the user with a method to change the window's frobbozes. This method should use the `:redefine-margins` method.

`:compute-margins` *lm tm rm bm*

Method of *windows*

The system uses `:compute-margins` to find out how much space is needed in each margin of the window by borders, labels, and anything else. Each flavor that implements a kind of margin item must define a method for it. `:compute-margins` uses the `:pass-on` method combination so the values from one method become the arguments to the next. (See the *Explorer Lisp Reference* manual for a description of the `:pass-on` method.) The *lm*, *tm*, *rm*, and *bm* arguments are interpreted as the width in pixels for each margin. Each method increments one or more of these arguments by the amount of space needed by that mixin.

`:refresh-margins`

Method of *windows*

The `:refresh` method of `tv:minimum-window` erases the margins, then calls the `:refresh-margins` method. The `:refresh-margins` method and its `:before` and `:after` methods redraw the contents of the window margins.

Assume that the frobbozes always go in the left margin. Consequently, it is always the left margin's width that is incremented, and the others are returned just as they were passed. Also assume that `frobboz-margin-width` is a function you have defined that computes the width of space that the frobbozes need.

In addition to returning modified versions of its arguments, the `:compute-margins` method also sets up the value of `frobboz-margin-area`. This is the only place it is necessary to set that variable. By recording the position of each margin item this way, you take into account how one margin item affects the position of the others. For example, the frobbozes might fall inside the borders, and then the *lm*, *tm*, *rm*, and *bm* values will already contain the width of the borders. In this case, `frobboz-margin-area` describes a rectangle that is within the borders.

The following is one way to define the `:compute-margins` method:

```
(defmethod (frobboz-margin-mixin :compute-margins)
  (lm tm rm bm)
  (let ((wid (frobboz-margin-width current-frobbozes)))
    (setq frobboz-margin-area
      (list lm tm (+ lm wid) (- w:height bm)))
    (values (+ lm wid) tm rm bm)))
```

Usually an additional mixin-specific method is introduced into this method, as follows:

```
(defmethod (frobboz-margin-mixin :compute-margins)
  (lm tm rm bm)
  (send self :recalculate-frobboz-margins lm tm rm bm))

(defmethod (frobboz-margin-mixin :recalculate-frobboz-margins)
  (lm tm rm bm)
  (let ((wid (frobboz-margin-width current-frobbozes)))
    (setq frobboz-margin-area (list lm tm (+ lm wid) (- w:height bm)))
    (values (+ lm wid) tm rm bm)))
```

This way, other mixins can be defined to modify where the frobbozes go by replacing the `:recalculate-frobboz-margins` method.

You must also provide a method for `:refresh-margins` to draw the frobbozes in the margin. You can assume that the margin is clear to start with.

```
(defmethod (frobboz-margin-mixin :after :refresh-margins) ()
  (w:sheet-force-access (self)
    (draw-frobbozes current-frobbozes frobboz-margin-area)))
```

You may wish to provide the user with a method to change the window's frobbozes. This method should use the `:redefine-margins` method.

:redefine-margins

Method of *windows*

Recomputes how much margin space is needed for all of the margin items by invoking the `:compute-margins` method and then actually changes the window margin sizes, if necessary.

If the margin sizes have changed, then the window is erased and `:refresh-margins` is invoked. The `w:restored-bits-p` instance variable (present in all windows) is left set to `nil`. If the margin sizes have not changed, no output is done, and `w:restored-bits-p` remains set to `t`. All this is done using the `:refresh` method. For example:

```
(defmethod (frobboz-margin-mixin :set-frobbozes) (new-frobbozes)
  (setq current-frobbozes
    (canonicalize-and-validate-frobboz-spec new-frobbozes))
  (send self :redefine-margins)
  (when w:restored-bits-p
    (w:sheet-force-access (self)
      (erase-frobboz-area frobboz-margin-area)
      (draw-frobbozes current-frobbozes frobboz-margin-area))))
```

The frobbozes are explicitly erased and drawn when the total sizes of the margins have not changed (and therefore no screen updating has been done), in case the *contents* of the frobbozes have changed.

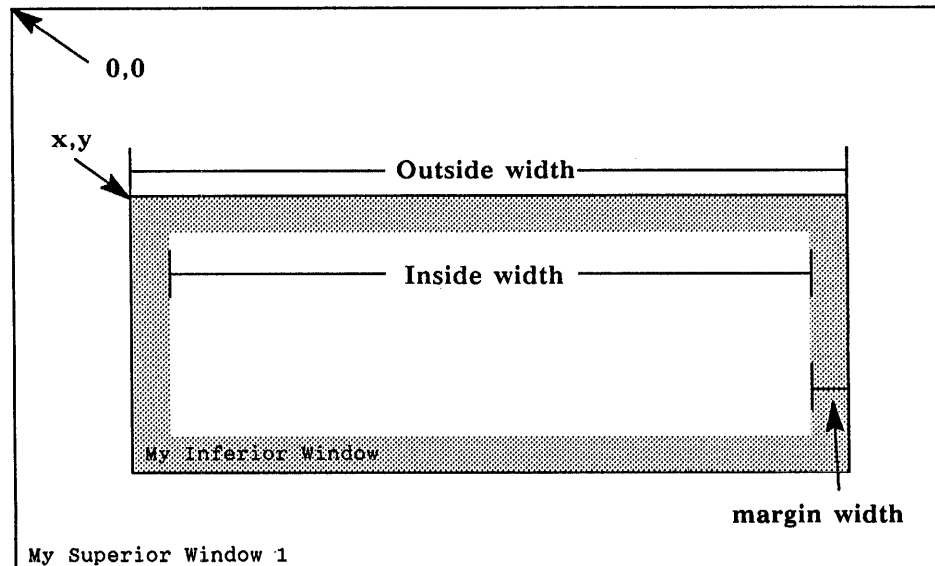
4

SIZES AND POSITIONS

Introduction

4.1 This section discusses how to examine and set the sizes and positions of windows. Many different methods let you express things in different forms that are convenient in varying applications. Sizes are usually specified in units of pixels. Widths are sometimes referred to in units of characters, and heights are sometimes expressed in units of lines. The number of horizontal pixels in one character is called the *character width*, and the number of vertical pixels in one line is called the *line height*; these two quantities are explained in the introduction to Section 7, Output of Text.

A window has two parts: the inside and the margins. Some of the methods in the following paragraphs deal with the outside size (including the margins), and some deal with the inside size. Margins include borders and labels, while the inside is used for drawing characters and graphics.



Initialization Options for Sizes and Positions

4.2 Because a window's size and position are usually established when the window is created, this paragraph begins by discussing the initialization options that let you specify the size and position of a new window. For your convenience, there are many ways to express what you want. The idea is that you specify various things, and the window system computes whatever you leave unspecified. For example, if you specify the left edge and the width of a window, the position of the right edge is computed using the left edge and width.

If you do not specify some parameters, defaults are used. Each edge defaults to the corresponding inside edge of the superior window. For example, if you specify the position of the left edge but do not specify the width or the position of the right edge, then the right edge defaults to the inside right edge of

the superior. If you specify the width but neither edge position, the left edge defaults to the inside left edge of the superior; the same goes for the height and the top edge.

Before a window can be exposed, its position and size must be such that it fits within the *inside* of the superior window. If a window is not exposed, there are no constraints on its position and size; it may overlap its superior's margins or even be outside the superior window altogether.

:left *left-edge* Initialization Option of *windows*
:x *left-edge* Initialization Option of *windows*

Sets the left edge of the window relative to the left edge of the window's superior. *left-edge* specifies the offset of the window's left edge in pixels.

:top *top-edge* Initialization Option of *windows*
:y *top-edge* Initialization Option of *windows*

Sets the top edge of the window relative to the top edge of the window's superior. *top-edge* specifies the offset of the window's top edge in pixels.

:position *position-list* Initialization Option of *windows*

Sets the left and top edge of the window relative to the left and top edges of the window's superior. *position-list* is a list of the form (*left-edge top-edge*) where *left-edge* and *top-edge* specify the offset of the window's left and top edges, respectively, in pixels.

:right *right-edge* Initialization Option of *windows*

Sets the right edge of the window relative to the left edge of the window's superior. *right-edge* specifies the offset of the window's right edge in pixels.

:bottom *bottom-edge* Initialization Option of *windows*

Sets the bottom edge of the window relative to the top edge of the window's superior. *bottom-edge* specifies the offset of the window's bottom edge in pixels.

:edges *edges-list* Initialization Option of *windows*

Sets the edges of the window. *edges-list* is a list of the form (*left-edge top-edge right-edge bottom-edge*) where the values of *left-edge*, *top-edge*, *right-edge*, and *bottom-edge* specify the new edges of the window in pixels.

NOTE: All edge parameters are relative to the *outside* of the superior window.

:width *outside-width* Initialization Option of *windows*
:height *outside-height* Initialization Option of *windows*
:size *size-list* Initialization Option of *windows*

Sets the outside width and/or height of the window to the value of the argument(s). The arguments are expressed in pixels. *size-list* is a list of the form (*outside-width outside-height*).

- :inside-width** *inside-width* Initialization Option of *windows*
:inside-height *inside-height* Initialization Option of *windows*
:inside-size *size-list* Initialization Option of *windows*
- Sets the inside width and/or height of the window to the value of the argument(s). The arguments are expressed in pixels. *size-list* is a list of the form (*inside-width inside-height*).
- :character-width** *spec* Initialization Option of *windows*
:character-height *spec* Initialization Option of *windows*
- Another way of specifying the width or height of the window. *spec* is either a number of characters or a character string. The inside width or height, respectively, of the window is made wide or high enough to display those characters in the current font.
- :integral-p** *t-or-nil* Initialization Option of *windows*
 Default: **nil**
- Specifies whether the inside dimensions of the window can contain a number of lines (no fractional parts) and characters. If *t* is specified, the inside dimensions of the window are made to be an integral number of characters wide and lines high by moving the margins to make the window larger.
- :edges-from** *source* Initialization Option of *windows*
- Specifies that the window is to use the edges (position and size) specified by *source*.
- The *source* argument can be one of the following:
- A string — The inside size of the window is made large enough to display the string in the current font.
 - A list of the form (*left-edge top-edge right-edge bottom-edge*) — These edges, relative to the superior, are used exactly as if you had used the **:edges** initialization option.
 - **:mouse** — The user is asked to point the mouse to where the top left and bottom right corners of the window should go. (This is what happens when you use the Create command in the System menu.)
 - A window — That window's edges are copied.
- :minimum-width** *n-pixels* Initialization Option of *windows*
:minimum-height *n-pixels* Initialization Option of *windows*
- These initialization options, in combination with the **:edges-from** **:mouse** initialization option, specify the minimum size of the rectangle that can be accepted from the user. If the user tries to specify a size smaller than one or both of these minimums, the system sounds a beep and prompts the user to start over again with a new top left corner. *n-pixels* is the minimum width or height in pixels.

Methods for Sizes and Positions

4.3 The methods discussed in the following paragraphs examine or change the size or position of a window.

The Option Argument

4.3.1 Many methods that change the window's size or position use an argument called *option*. This argument deals with new sizes or positions that are not valid. A size may not be valid because it may be so small that there is no room for the margins. If the new width is smaller than the sum of the sizes of the left and right margins, then the new width is not valid. A new setting of the edges is also invalid if the window is exposed and the new edges are not enclosed inside its superior. In all of the methods that use the *option* argument, *option* can be either *nil* or *:verify*:

- *nil* means that you want to set the edges, but if the new edges are not valid, an error should be signaled.
- *:verify* means that, rather than changing the edges, you want to check whether the new edges are valid. If the edges are valid, the method with *:verify* returns *t*; otherwise, it returns two values: *nil* and a string explaining what is wrong with the edges.

NOTE: It is valid to set the edges of a deexposed inferior window in such a way that the inferior is not enclosed inside the superior; however, you cannot expose the inferior until it is enclosed inside the superior. This requirement makes it more convenient to change the edges of a deexposed window and all of its inferiors sequentially, because you do not have to be careful about the order in which you change the edges.

The Methods

4.3.2

:height
:width

Method of *windows*
Method of *windows*

Returns the window's outside height or width, respectively, in pixels.

w:sheet-width window
w:sheet-height window

Defsubst
Defsubst

Returns the outside width or height of the window identified by *window*.

You should usually use higher-level methods to deal with edges, since higher-level methods perform error checking to ensure that you do not inadvertently introduce problems.

When used without an argument, *w:sheet-width* refers directly to the *w:width* instance variable; *w:sheet-height* refers directly to the *w:height* instance variable. Therefore, these defsubsts must be called from methods or functions that use `(declare (:self-flavor ...))`.

:size Method of *windows*
:inside-size Method of *windows*

Returns two values: the width and height in pixels. **:size** returns the outside measurements; **:inside-size** returns the inside measurements.

:set-size *new-width new-height* &optional *option* Method of *windows*
:set-inside-size *new-inside-width new-inside-height* &optional *option* Method of *windows*

Sets the width and height of the window without changing the position of the upper left corner. *new-width* and *new-height* are specified in pixels. **:set-size** sets the outside size. **:set-inside-size** sets the inside size, and the margin sizes are recomputed according to their contents. In simple cases, the margin sizes stay the same.

:inside-height Method of *windows*
w:sheet-inside-height &optional (*window self*) Macro
:inside-width Method of *windows*
w:sheet-inside-width &optional (*window self*) Macro

Returns the window's inside height or width, respectively, in pixels. *window* specifies the window whose inside height or width, respectively, is to be determined.

When used without an argument, these macros refer directly to the instance variables containing the locations of the window margins and size, and therefore they must be called from methods or functions that use (declare (:self-flavor ...)).

:position Method of *windows*

Returns two values: the x and y positions of the upper left corner of the window, in pixels, relative to the superior window.

:set-position *new-x new-y* &optional *option* Method of *windows*

Sets the x and y position of the upper left corner of the window, in pixels, relative to the superior window's coordinate system.

w:sheet-inside-left &optional (*window self*) Macro
w:sheet-inside-top &optional (*window self*) Macro
w:sheet-inside-right &optional (*window self*) Macro
w:sheet-inside-bottom &optional (*window self*) Macro

Returns the position of the respective inside edge of the window, relative to the top left outside corner of the window. If used with no argument, these macros expand into direct references to instance variables and therefore can be used only within methods or (declare (:self-flavor ...)) functions.

:edges Method of *windows*
:inside-edges Method of *windows*

Returns four values: the left, top, right, and bottom edges, in pixels. The **:edges** method returns the outside edges relative to the superior window; the **:inside-edges** method returns the inside edges relative to the top left corner of this window. The **:inside-edges** method can be useful for clipping, that is, truncating the portions of images that would appear outside a certain boundary.

NOTE: See paragraph 4.3.1, The *option* Argument, for an explanation of the *option* argument.

:set-edges *new-left new-top new-right new-bottom* Method of *windows*
&optional option

:inferior-set-edges *window new-left new-top new-right* Method of *windows*
new-bottom &optional option

Sets the edges of the window to the values of the arguments, in pixels. For **:set-edges**, the values are relative to the superior window. For **:inferior-set-edges**, the window specified by *window* is an inferior of the window being sent the **:inferior-set-edges** message; the edge values are relative to *window*'s superior.

:center-around *x y* Method of *windows*

Positions the window so that its center is as close to the point (*x,y*) as possible without hanging off an edge. **:center-around** does not change the size of the window. The coordinates are in pixels relative to the superior window.

:change-of-size-or-margins *&rest options* Method of *windows*

The primitive method for changing a window's size or the size of its margins. All the other methods that do this call **:change-of-size-or-margins** after the other methods have done error checking.

You should not use the **:change-of-size-or-margins** method to manipulate a window directly. Instead, you should use **:set-size** or another higher-level method to change size and use flavors that compute the margins (such as **w:borders-mixin**) to manage margin sizes. However, the **:change-of-size-or-margins** method is a good place to add **:after** methods to recompute other data structures or change the size of inferiors according to the window's new size. In the **:after** method, the window's size and margins will already be altered to their new values.

Two related symbols, the **:expose-near** method and the **w:position-window-next-to-rectangle** function, are described in paragraph 5.7.3, Symbols That Manipulate Screen Arrays and Exposure.

Low-Level Edges Functions

4.4 The following low-level instance variables, macros, and functions should be used with care. Although these primitives may be very fast, they may only do one thing—which may not be sufficient to complete a task. You should typically use the methods discussed previously to examine or alter the sizes and positions of windows.

w:x-offset Instance Variable of *windows*
w:y-offset Instance Variable of *windows*

Sets the x or y position of the window's outside left edge relative to the window's superior.

w:sheet-x-offset *window* Defsubst
w:sheet-y-offset *window* Defsubst

Returns the x or y offset of the window identified by *window*.

You should usually use higher-level methods to deal with edges, because higher-level methods perform error checking to ensure that you do not inadvertently introduce problems.

w:sheet-calculate-offsets *window superior* Function

Returns, as two values, the x and y positions of a window's upper left corner in its superior. If *window* and *superior* are the same window, the values are 0. *window* must be an indirect inferior of *superior* zero or more levels down.

w:sheet-number-of-inside-lines *&optional (window self)* Macro

Returns the number of lines (of height equal to **w:line-height**) that fit in the inside height of *window*.

When used without an argument, this function refers directly to the instance variables containing the locations of the window margins and size, and therefore it must be called from methods or functions that use `(declare (:self-flavor ...))`.

w:sheet-overlaps-p *window left top width height* Function
w:sheet-overlaps-edges-p *window left top right bottom* Function

Calculates the position of the window edges using the last four arguments. Both functions return **t** if *window* overlaps the specified rectangle. The edges specified are relative to the superior of *window*.

w:sheet-overlaps-sheet-p *window-a window-b* Function

Returns **t** if *window-a* and *window-b* overlap. This is a geometric test; it does not matter where the two windows are in the hierarchy.

w:sheet-within-p *window left top right bottom* Function

Calculates the position of the window edges using *left*, *top*, *right*, and *bottom* and returns **t** if *window* is contained within the specified rectangle relative to the superior of *window*.

w:sheet-within-sheet-p *window outer-window* Function

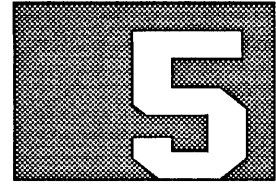
Returns **t** if *window* is within *outer-window*. This is a geometric test; it does not matter where the two windows are in the hierarchy.

w:sheet-bounds-within-sheet-p *left top width height outer-window* Function

Calculates the position of the window edges using *left*, *top*, *width*, and *height* and returns **t** if the specified rectangle is within *outer-window*. The edges are specified relative to the superior of *outer-window*.

w:sheet-contains-sheet-point-p *window top-window x y* Function

Returns **t** if *window* contains the point specified by *x* and *y* in the superior identified by *top-window*.



VISIBILITY AND EXPOSURE

Introduction

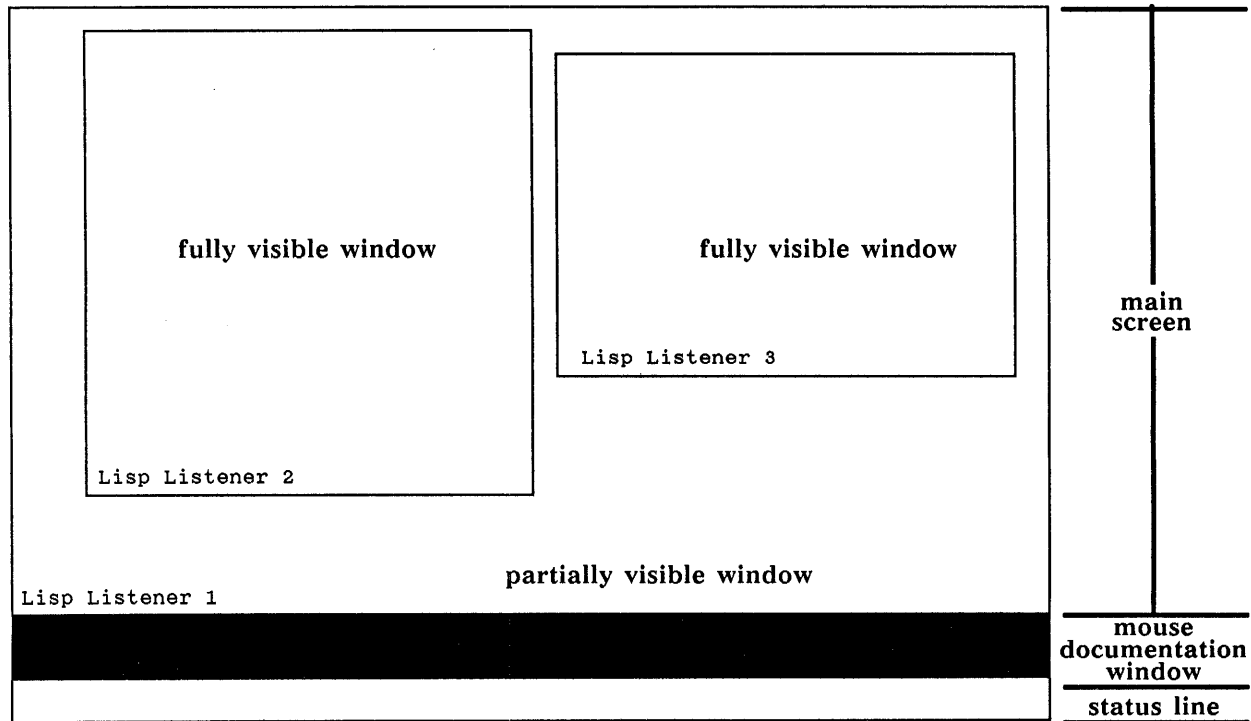
5.1 This section explains the two most important aspects of a window: whether it is visible on the screen and whether it is exposed. Understanding these concepts is essential if you use the window system.

- A *visible* window is displayed, either completely or partially, on the video display. Windows that are only partially visible are *overlapped* by another window.
- An *exposed* window can accept output. All fully visible windows are exposed. A window that is not fully visible can be exposed if it has a *bit-save array*, an array that saves the *contents* of the window.

If a fully visible window does not completely cover the video display, the *screen manager* determines what window is partially visible according to a strict hierarchy. The hierarchy takes into account things such as the *priority* assigned to a window and which window was last visible.

Figure 5-1 shows an example of overlapping windows. Both Lisp Listener 2 and Lisp Listener 3 windows are exposed, and both overlap Lisp Listener 1.

Figure 5-1 Overlapping Windows



The part of the hierarchy that is immediately inferior to the main screen includes the frames and windows that, when visible, usually cover the main portion of the video display. This part of the hierarchy is commonly referred to as the *stack*. Each frame in the stack includes its panes as inferiors.

You can show the hierarchy of windows by examining the Windows option of the Peek utility. For example, the following is part of the hierarchy for the system that contains the three Lisp Listeners shown in Figure 5-1. Note that the topmost window in the stack is currently the Peek window, followed by Lisp Listener 3, Lisp Listener 2, and then Lisp Listener 1. Other windows are automatically created by the Explorer system when you boot.

```

Screen Main Screen
  Peek Frame 1
    Basic Peek 1
    Dynamic Highlighting Command Menu Pane 1
    Dynamic Highlighting Command Menu Pane 2
  Lisp Listener 3
  Lisp Listener 2
  Lisp Listener 1
  Suggestions Frame Off 1
    Menu Manip Strip 3
  ...

```

The screen manager continually updates the frames in the stack as the user selects different frames.

Screens

5.2 Screens, which are at the top of the window hierarchy, are instances of the `w:screen` flavor. Each screen object usually represents an individual piece of display hardware. However, the main black-and-white video display that all Explorers have is logically divided into two screens, with different screen objects: the `w:main-screen` at the top and the `w:who-line-screen` on the bottom. An example of these is shown in Figure 5-1. Because these are separate screens, windows on the main screen cannot be extended onto the who-line screen, the mouse cannot move onto the who-line screen, and so on.

`w:screen` Flavor

Screens are also flavor instances whose flavors incorporate `w:screen`. Screens are not windows, but they have much in common with windows because both incorporate the `w:sheet` flavor. There is only one `w:sheet-area` for each hardware display.

`w:sheet-area` Variable

The area in which windows are created by default. For a discussion of areas, see the section about storage management in the *Explorer Lisp Reference* manual.

`w:sheet-get-screen window &optional highest` Function

Returns *window* if *window* is a screen; otherwise, returns the screen that is the ultimate superior of *window*. In other words, `w:sheet-get-screen` searches for the highest level superior of *window* that is either a screen or is not active. *window* identifies the window or screen where `w:sheet-get-screen` begins the search for the screen. If *highest* is non-nil, inferiors of *highest* are treated like screens.

w:main-screen	Variable
The screen object representing the Explorer black-and-white display, except for the who-line screen.	
w:who-line-screen	Variable
The screen object representing the who-line screen. Each field of the who-line is a separate window on this screen.	
w:default-screen	Variable
The default superior for windows created with the make-instance function. It is initialized to be the main screen.	
w:all-the-screens	Variable
A list of all screen objects. Using the contents of w:all-the-screens , you can search through all the active windows, much like the w:map-over-sheets function does. The following is a typical value for w:all-the-screens :	
<pre>(#<STANDARD-SCREEN Main Screen 3340126 exposed> #<WHO-LINE-SCREEN Who Line Screen 4300000 exposed>)</pre>	

Hierarchy of Windows

5.3 Windows are arranged in a hierarchy, each window having one direct superior and a list of inferiors. The top of the window hierarchy is a screen. Windows can have several indirect superiors; however, the window has only one *direct* superior.

Each superior keeps a list of all of its active inferiors using the **w:inferiors** instance variable. Each inferior window keeps track of its superior using the **w:superior** instance variable. Superior windows do not keep track of their inactive inferiors; this allows unused windows to be reclaimed by the garbage collector. When a window is deactivated, the window system does not use it until it is activated again.

If you deactivate a window, the pointer to the window is removed from the **w:inferiors** instance variable kept by the window's superior. If there is no other reference to the window, the window can be reclaimed by the garbage collector. To prevent the garbage collector from reclaiming a deactivated window, put a pointer to the window in a variable. In this way, you can reactivate the window should you need to.

Inactive windows are never visible; a window must become active to be visible.

For example, when the Inspector window is invoked using the **SYSTEM I** keystroke sequence, the superior of the Inspector window is the screen, and the subwindows in the Inspector window are inferiors of the Inspector window. The screen itself has no superior (if you ask for its superior, **nil** is returned). You can examine the window hierarchy from the Peek utility. If

you first select the Inspector and then Peek, Peek displays something similar to the following:

```

Screen Main Screen
  Peek Frame 1
    Basic Peek 1
    Dynamic Highlighting Command Menu Pane 1
    Dynamic Highlighting Command Menu Pane 2
  Inspect Frame 2
    Inspect Window 4
    Inspect Window 3
    Inspect Window With Typeout 2
    Inspector Interaction Pane 2
    Inspector Pane Menu 2
    Inspect History Window 2
...

```

Normally, the Edit Screen command from the System menu edits the arrangement of the windows on a screen, but it can also edit the arrangement of inferiors (panes) of a window in the same fashion.

The Edit Screen item, then, manipulates a set of inferiors of a specific superior, which may or may not be a screen. The set of inferiors is called the *active inferiors set*; that is, each inferior in this set is said to be active. The active inferiors can be visible on their superior. If no two active inferiors overlap, there is no problem; they can all be exposed. This is the case for the panes within the Peek frame and the Inspector frame. However, when two inferiors overlap, only one of them can be exposed. The screen manager chooses which inferior to expose. The Edit Screen item of the System menu lets you override this choice.

The following symbols affect the hierarchy of windows.

- :superior** *superior* Initialization Option of *windows and screens*
Gettable, settable. Default: `w:default-screen`, which is initially the main screen.
- w:sheet-superior** *window-or-screen* Macro
Returns either the superior of this window, or `nil` for a screen. The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.
- :inferiors** Method of *windows and screens*
Default: `nil`
- w:sheet-inferiors** *window-or-screen* Macro
Returns a list of the active inferiors. The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.

```

> (w:sheet-inferiors w:who-line-screen)
(#<WHO-LINE-SHEET Who Line Sheet 5 4300063 exposed>
 #<WHO-LINE-FILE-SHEET Who Line File Sheet 1 4300140 exposed>
 #<WHO-LINE-SHEET Who Line Sheet 4 4300220 exposed>
 #<WHO-LINE-SHEET Who Line Sheet 3 4300275 exposed>
 #<WHO-LINE-SHEET Who Line Sheet 2 4300352 exposed>
 #<WHO-LINE-SHEET Who Line Sheet 1 4300427 exposed>)

```


w:sheet-me-or-my-kid-p *window me* Function
 Returns **t** if *window* is equal to the window *me* or if *window* is an inferior of *me* (that is, if *window* is in the **w:inferiors** instance variable for *me*). Inferiors are sometimes referred to as *descendants*, hence the **kid** in **w:sheet-me-or-my-kid-p**.

:activate Method of *windows*
 Makes the window active in its superior by putting a pointer to the window in the **w:inferiors** instance variable of the window's direct superior.

:activate-p &optional *t-or-nil* Initialization Option of *windows and screens*
 Specifying this option as non-**nil** when you create a window causes the created window to be active. *t-or-nil* specifies whether the window will be activated when the window is created. The following cases are possible:

Value	Action
No value	The window is created but not activated. You can use :expose-p to force activation.
t	The window is activated when it is created.
nil	The window is created but not activated. You can use either :activate-p or :expose-p to activate the window.

:active-p *t-or-nil* Method of *windows and screens*
 Returns **t** if this window is active in its superior. A screen is always considered active.

:deactivate Method of *windows*
 Deactivates a window by removing any reference to the window from the **w:inferiors** instance variable of the window's superior.

:kill Method of *windows*
 Deactivates and also makes a positive effort to get rid of other entities that may be associated with the window (such as processes or network connections). If a window has associated entities, garbage collection alone may not be satisfactory. Therefore, **:kill** is preferable to **:deactivate**.

Lists of Windows 5.4 The following functions provide ways to manipulate lists of windows (windows are also called sheets). The **w:map-over-exposed-sheets** and **w:map-over-exposed-sheet** functions map only exposed windows. The **w:map-over-sheets** and **w:map-over-sheet** functions perform in a similar manner, except they map over active windows.

w:map-over-exposed-sheets *function* Function
 Calls the function identified by *function* on every exposed sheet, starting with the screens, their inferiors, and so on down the hierarchy.

The following example defines a function that prints out the name, height, and width for each window. The argument to this new function is a single

window. The `w:map-over-exposed-sheets` function calls this function once for each exposed window.

```
(w:map-over-exposed-sheets
 #'(lambda (a-window)
      (format t "~%Window is ~A, height=~A, width=~A"
              a-window
              (send a-window :height)
              (send a-window :width))))
```

If Lisp Listener 1 is the only exposed window, the example returns the following. If other windows are exposed, you get a different output.

```
Window is Lisp Listener 1, height=734, width=1024
Window is Main Screen, height=754, width=1024
Window is Who-Line Sheet 5, height=37, width=1024
Window is Who-Line File Sheet 1, height=14, width=512
Window is Who-Line Sheet 4, height=14, width=168
Window is Who-Line Sheet 3, height=14, width=80
Window is Who-Line Sheet 2, height=14, width=104
Window is Who-Line Sheet 1, height=14, width=160
Window is Who-Line Screen, height=54, width=1024
```

Note that this example was defined using lambda notation. The special notation, sharp-sign single quote (`#'`), tells the Lisp interpreter or compiler that the next expression is a function that can be evaluated.

<code>w:map-over-exposed-sheet</code>	<i>function window</i>	Function
<code>w:map-over-sheet</code>	<i>function window</i>	Function

Calls *function* on *window* and on every exposed inferior (for `w:map-over-exposed-sheet`) or on every active inferior (for `w:map-over-sheet`) of *window* to all levels. *window* is the window or screen level where the function begins execution.

For example, if you print the height and width of the exposed windows using the following code:

```
(w:map-over-exposed-sheet
 #'(lambda (aw)
      (format t "~%Window is ~A, height=~A, width=~A" aw
              (send aw :height) (send aw :width))) w:main-screen)
```

Depending on the windows active in your system, the output may look like the following:

```
Window is Menu Manip Strip 3, height=18, width=1022
Window is Suggestions Frame Off 1, height=20, width=1024
Window is Lisp Listener 1, height=734, width=1024
Window is Main Screen, height=754, width=1024
```

<code>w:map-over-sheets</code>	<i>function</i>	Function
--------------------------------	-----------------	----------

Calls *function* on every active window, starting with the screens, their inferiors, and so on down the hierarchy.

For example, suppose you print the names of the active windows using the following code:

```
(w:map-over-sheets
 #'(lambda (aw)
      (format t "%-A" aw) w:main-screen))
```

Then, depending on the windows active in your system, the output may look like the following:

```
Lisp Listener 1
Menu Manip Strip 3
Suggestions Frame Off 1
Editor Typeout Window 3
Edit: MH: WEBB; *.*## (1)
Typein Window 4
Zwei Mini Buffer 6
Mode Line Window 3
Menu Manip Strip 1
Menu Manip Strip 2
Label Pane Manip Strip 1
Nice Scrolling Suggestions Window 1
Suggestions Frame 1
Main Screen
Who Line Sheet 5
Who Line File Sheet 1
Who Line Sheet 4
Who Line Sheet 3
Who Line Sheet 2
Who Line Sheet 1
Who Line Screen
```

Pixels

5.5 A screen displays an array of picture elements (pixels). Each pixel is a little dot of some brightness; a screen displays a big array of these dots to form a picture. Everything you see on the screen, including borders, graphics, characters, and blinkers, is made from pixels.

Each physical screen has a display memory that stores the values of all the pixels. On regular black-and-white screens, each pixel has one of only two values: lit or unlit. Thus, the pixel is represented in memory by one bit. Usually 0s are used for the background of a window, and the characters or lines on it are made of 1s. So 1 can be considered on and 0 off.

Black-and-white screens have a hardware flag that controls the visual appearance of 1 and 0 pixels. In *black-on-white* mode, 1 is dark and 0 is bright, so windows appear with dark text on a white background. This mode is the default. In *white-on-black* mode, 1 represents white and 0 represents black. Users can switch between these modes using the TERM C keystroke sequence.

An individual window can specify 1 for background and 0 for text; this is independent of *white-on-black* mode (which applies to the whole screen) and is requested with the `:reverse-video-p` initialization option or the `:set-reverse-video-p` method. These work by controlling the arithmetic logic unit (ALU) arguments used for drawing and erasing characters. Programs that use the window's recommended ALU arguments for their drawing and erasing will automatically display in reverse video. The mouse documentation window is an example of a window that uses reverse video.

On a color monitor running in a color environment, each pixel can have one of 256 values, ranging from 0 to 255. Thus, a pixel is no longer just on or off, black or white, or 0 or 1, but rather it is an integer with a range of values.

The color controller actually generates 24 bits of color information per 8-bit pixel to drive the color monitor. These 24 bits are composed of 8 bits for RED, 8 bits for GREEN, and 8 bits for BLUE.* So, each of the three colors (R, G, and B) can take on one of 256 values, with 0 being completely off and 255 being full strength. Thus, taken in all possible combinations, there are over 16 million possible colors. Since each pixel can take on one of 256 possible values, each 8-bit pixel value is translated by the hardware into a 24-bit RGB value by indexing into a data table called the Color Look-Up Table (LUT). Thus, up to 256 different colors can be displayed on the color monitor.

Note that the color associated with an integer may change as the contents of the LUT change. This integer value is referred to as either the *logical color* of a pixel or the *pixel value*. The value generated by the LUT that corresponds to the logical color is the *physical color*.

In a color environment, reverse video is achieved by transposing the values of the foreground and background colors instead of by manipulating ALUs. For details on color, refer to Section 19, Using Color.

- w:black-on-white** Function
Displays 1-bits as black, and 0-bits as white. (This is the default mode.) This function works by setting a bit in the display hardware.
- w:white-on-black** Function
Makes the monitor display 1-bits as white and 0-bits as black.
- w:complement-bow-mode** Function
In a monochrome environment, this function switches between displaying 1-bits as white or as black. (The **bow** in the name is an abbreviation of **black-on-white**.) The TERM C keystroke sequence uses this function.
- :complement-bow-mode** Method of *windows*
In a color environment, this method transposes the values of the foreground and background colors for the window.
- w:bits-per-pixel** Instance Variable of **w:screen**
For a black-and-white screen, one bit per pixel.

* The RGB model for representing color is based on the three color photoreceptors in the human eye, which detect either red, green, or blue. The RGB model is *not* based on the three primary colors red, blue, and yellow, which many people might expect.

:reverse-video-p *t-or-nil* Initialization Option of *windows*
Gettable, settable. Default: **nil**

Sets the use of reverse video display. When *t-or-nil* is **t** in a monochrome environment, the background of the window is black; when *t-or-nil* is **nil**, the background of the window is white.

When *t-or-nil* is **t** in a color environment, the background color of the window is the value of the window's background color instance variable; when *t-or-nil* is **nil** in a color environment, the background color of the window is the value of the window's foreground color instance variable.

The **:reverse-video-p** initialization option is separate from the whole screen's reverse video mode. In a monochrome environment, the TERM C keystroke sequence sets the whole screen's reverse video mode. In a color environment, the **:complement-bow-mode** method sets the whole screen's reverse video mode.

w:buffer Instance Variable of **w:screen**

The address of the screen memory; the address is stored as a fixnum.

NOTE: The **w:buffer** and **w:buffer-halfword-array** instance variables are used during system initializations and are not intended to be changed by the user.

w:buffer-halfword-array Instance Variable of **w:screen**

An art-16b array containing the screen memory.

Bit-Save Arrays

5.6 The pixel values that make up a window's screen image are called its *contents*. When a window is fully visible, its contents are displayed on a screen so they can be seen. When a window is not fully visible, its contents are lost unless there is a place to save them. Such a place is called a *bit-save array*.

A bit-save array is an array of bits large enough to hold a copy of the contents of the window. If a window has a bit-save array, its contents are copied into the array when the window ceases to be fully visible. If the window becomes fully visible again, the contents are copied from the bit-save array back onto the screen. During the interim, programs can use the **w:sheet-force-access** macro to output to the bit-save array while the window is not visible, and the window's inferiors, if any, can be exposed and do output. (See paragraph 5.7, Screen Arrays and Exposure.)

When a window with a bit-save array is partially visible, the visible parts can be displayed correctly by copying them from the bit-save array. This occurs when a small window is completely surrounded by a full-screen window, such as the initial Lisp Listener, as shown in Figure 5-1. The visible parts of the larger window are copied from the window's bit-save array.

If a window does not have a bit-save array, there is no place to put its contents when it is not visible, so they are lost. When the window becomes visible again, it tries to redraw its contents, that is, to regenerate the contents

from state information in the window. This is done by the `:refresh` method documented later. Some windows can do this; for example, editor windows can regenerate their contents based on the editor buffers they are displaying. Other windows, such as Lisp Listeners, cannot regenerate their previous contents. Such windows simply leave their contents blank, except for the margins, which all windows can regenerate.

The advantage of having a bit-save array is that losing and regaining visibility does not require the contents to be regenerated; this feature is desirable because regeneration may be computationally expensive, or even impossible. The disadvantage is that copying the bit-save array to disk and returning it to memory may take too much time.

Furthermore, bit-save arrays on a color system are eight-bit arrays, which require more memory, so their use should be limited.

When a frame is in use, giving the frame a bit-save array enables the contents of the frame and all the panes to be preserved if the frame ceases to be fully visible. Bit-save arrays for the panes would be needed only if panes were shuffled or substituted within the frame. In most applications, this situation rarely happens, but if it does, it is accompanied by a complete redisplay. Thus, the frame normally gets a bit-save array and the panes do not.

w:bit-array Instance Variable of *windows*
w:sheet-bit-array *window* Macro

The window's bit array if the window has a bit array. If the window does not have a bit array, the `w:bit-array` instance variable is `nil`. The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.

:save-bits *flag* Initialization Option of *windows*
Gettable, settable. Default: `nil`

Determines whether the bits are saved and when the bit-save array is created, if the bits are to be saved. *flag* can be `t`, `nil`, or `:delayed`:

- `t` causes the bits to be saved.
- `nil` causes the bits not to be saved.
- `:delayed` causes the window to create a bit-save array the first time it is deexposed, but not before.

:refresh &optional (*type* `:complete-redisplay`) Method of *windows*

Restores the saved contents of the window or regenerates the contents, according to the value of *type* and whether the window has a bit-save array.

If you define your own window flavor—or redefine a standard window flavor—you should include `:after` methods so the `:refresh` method can complete the job of redrawing a window. Unless you do this, the system may redraw the window in a manner you do not want. When these `:after` methods run, the `w:restored-bits-p` instance variable is non-`nil` if the window contents were restored from a bit-save array. If this is so, the `:after` methods need not do anything unless the window's inside size has changed. *type* determines

how the window uses its bit-save array. The following keywords are the possible values of *type*:

- **:complete-redisplay** specifies that window's current bit image be completely discarded and regenerated from scratch. The margins are redrawn by invoking the **:refresh-margins** method. The default definition of **:refresh** leaves the inside portion of the window blank except for refreshing any exposed inferiors. If the window has no bit-save array, *type* is ignored and the actions for **:complete-redisplay** are always used.
- **:use-old-bits** specifies that the complete contents are restored from the bit-save array. This is specified by the system when a window is exposed.
- **:size-changed** is a flag that indicates that the window size has been changed. The contents are restored from the bit-save array, and then the margins are refreshed with the **:refresh-margins** method.
- **:margins-only** is a flag that indicates that only the borders have changed. You should use this keyword when the inside portion of the window is completely undisturbed and only the margins need to be refreshed. The system treats **:margins-only** just like **:size-changed**.

w:restored-bits-p

Instance Variable of *windows*

Indicates whether the contents of the window have been restored. If **w:restored-bits-p** is **nil**, the inside of the window was left blank and must be regenerated to whatever extent possible. In the **:after** methods of the **:refresh** method (and therefore also of the **:expose** method), **w:restored-bits-p** is **t** if the contents were restored from a bit-save array.

Screen Arrays and Exposure

5.7 Screen arrays and exposure control how the system decides where to put a window's contents (its pixels), how the notion of visibility on the screen is extended into a hierarchy of windows, and how programs can control which windows are visible.

Concepts of Screen Arrays

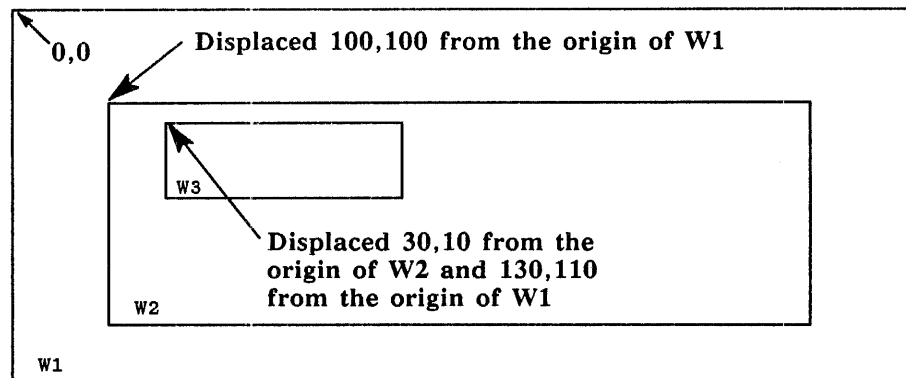
5.7.1 Each window or screen can have a *screen array*, which is where output to the window should be drawn. Drawing characters or graphics is done by changing pixels of the window's screen array. The screen array is stored in the **w:screen-array** instance variable. If the variable is **nil**, the window does not have a screen array now.

A screen array is normally put in the special memory used by the hardware to display the screen. A visible window has a screen array; the array is an indirect array that points into the area of the superior's screen array where the inferior is displayed on the superior. For example, consider a visible window whose superior is a screen and whose upper left corner is at location 100,100 in the screen. The window's screen array would be an indirect array whose 0,0 element is the same as the 100,100 element of the screen; that is, every pixel of the visible window is displaced by 100,100. If you set a pixel at the visible window coordinates *x,y*, the system adds 100 to each coordinate and sets the pixel corresponding to *x + 100,y + 100* in the screen array. The following figure shows this displacement.

A visible window more than one level down from the screen has a screen array that is displaced more than once. If the visible window is displayed on its superior at the superior's coordinates of 50,50 and the visible window's

superior is displayed at its superior's—assume the superior is the screen—coordinates 100,100, the system puts all pixels located at coordinates x,y in the visible window at $100 + 50 + x, 100 + 50 + y$ on the screen. The window's screen array points into the middle of its superior's screen array, which points into the middle of the superior's superior's screen array, and so on until the main screen is reached. When typeout is done on the window, the typeout appears on the screen, offset by the combined offsets of all the ancestors, so that it appears in the correct absolute position on the screen.

The following figure illustrates the idea of displacement by showing the screen and two windows: the first window is displaced 100 pixels down and 100 pixels to the right of the screen's top left corner; the second window is displaced 10 pixels down and 30 pixels to the right of its superior's top left corner.



Concepts of Exposure

5.7.2 An *exposed* window can accept output. Most visible windows are also exposed, but not all exposed windows are visible. If an exposed window is not visible, you can still output to its bit-save array. If that window does not have a bit-save array, but an ancestor of that window does have one, output goes to the ancestor's bit-save array. If an exposed window is not visible, output to that window is not visible either until the window becomes visible.

The `:expose` method makes a window exposable; it does not automatically expose the window. If, at the time a window is made exposable, its superior has a screen array (that is, if its superior is visible), the window actually becomes exposed.

If the superior of a window that is exposable, but not currently exposed, later acquires a screen array, the window becomes exposed at that time. The superior can acquire a screen array by being exposed, or the superior can be given a bit-save array with the `:set-save-bits` method.

Having a superior exposable but not exposed is simple to demonstrate. To show this, follow these steps:

1. Select the Inspector by pressing SYSTEM I or by selecting it from the System menu.
2. Select the Peek utility by pressing SYSTEM P or by selecting it from the System menu.

3. Select the window option of the Peek utility by pressing W (for window) or by selecting the Window item from the command menu.

The system displays all active windows, including the Inspector frame and each of its inferior panes.

4. Click on the name of one of the Inspector windows in the list of windows.

The system displays a menu of actions that can be performed on this object.

5. Select the Describe item.

In the first line of the display, the system describes the pane as exposed. The pane itself, however, is not visible. Thus, the pane is exposable but not exposed.

The panes actually become exposed when their ancestor (the Inspector frame) acquires a screen array.

The `:deexpose` method makes a window not exposed; such a window must be explicitly exposed again with an `:expose` method. What happens when you attempt to output to a deexposed window depends on the window's deexposed timeout action (described in paragraph 7.4.1). You can override this action by using the `w:sheet-force-access` macro.

**Symbols That
Manipulate
Screen Arrays
and Exposure**

5.7.3

`w:sheet-force-access` (*window ignore*) &body *body* Macro

Executes *body*, outputting to *window* whether *window* is exposed or not. Thus, code within *body* can output even if *window* does not ordinarily allow output while deexposed. If *window* is deexposed and has no bit array, then *body* is not executed at all, but *window* refreshes completely when it is exposed.

`:expose` &optional *inhibit-blinkers bits-action* Method of *windows and screens*
new-left new-top

Makes the window exposable and exposed, if possible. If the window is not active in its superior, it is first activated. A window cannot be made exposable unless its full size fits within the superior.

This is a very useful method to attach `:before` and `:after` methods to, but remember that this method can be performed only on a window that is already exposable. The `:before` and `:after` methods must not make the assumption that the window is becoming exposable.

The arguments to the `:expose` method are supplied by the system and are usually of interest only to the system's methods. User invocations of this method usually should supply no arguments.

:expose-near *mode* &optional (*warp-mouse-p* *t*) Method of *windows*
w:expose-window-near *window mode* &optional (*warp-mouse-p* *t*)
(expose-p t) Function

If the window is not exposed, changes the window's position according to *mode* and exposes it using the **:expose** method. If the window is already exposed, does nothing.

Arguments: *mode* — A list that can be one of the following:

- **:point** *x y* positions the window so that its center is at the point *x,y*, in pixels, relative to the upper left corner of the superior window, or as close as possible without hanging off an edge of the superior. The mouse blinker moves to the superior window's *x,y* coordinates.
- **:mouse** is like the **:point** mode, but the *x* and *y* come from the current mouse blinker position instead of the arguments. This keyword behaves like pop-up windows (such as the System menu). In addition, if *warp-mouse-p* is non-nil, the mouse blinker is warped to the center of the window. (The mouse blinker moves only if the window is near an edge of its superior; otherwise, the mouse blinker is already at the center of the window.)
- **:window** *window-1 window-2 window-3...* positions the window next to but not overlapping the rectangle that is the bounding box of all the arguments to **:window**. You must provide at least one argument. Usually, you specify only one window; thus, the window is positioned touching one edge of that specified window. In addition, if *warp-mouse-p* is non-nil, the mouse blinker is warped to the center of the window.
- **:rectangle** *left top right bottom* specifies a rectangle, in pixels, relative to the superior window. The window is positioned near but not overlapping the rectangle. In addition, if *warp-mouse-p* is non-nil, the mouse blinker is warped to the center of the window.

warp-mouse-p — Determines whether the mouse blinker moves to the center of the window on which the **:expose-near** method operates, according to the value of *mode*.

expose-p — For the **w:expose-window-near** function, determines whether the window is exposed. If *expose-p* is non-nil, the window is exposed.

w:position-window-next-to-rectangle *window position* Function
left top right bottom

Moves *window* near the rectangle specified by *left*, *top*, *right*, and *bottom*. *left*, *top*, *right*, and *bottom* specify a rectangle using the inside pixel coordinates of *window*'s superior.

Unlike the **:expose-near** method, this function aligns the window with the rectangle, and other alternatives are attempted depending on the suggested position. The **:expose-near** method does not allow a suggested position.

position is a keyword or a list of keywords that indicates where to put the *window*. Possible values for *position* are **:above**, **:below**, **:left**, or **:right**; or a list of these keywords. If *position* is a single keyword and there is not enough room to place the window without obscuring the rectangle, then the alter-

natives shown in the following table are attempted in order of most preferable to least. If *position* is a list, then only those alternatives are attempted.

Initial Preference	Alternative Preferences		
	First	Second	Third
:above	:left	:right	:below
:below	:right	:left	:above
:left	:below	:above	:right
:right	:above	:below	:left

:expose-p *t-or-nil* Initialization Option of *windows and screens*
Gettable. Default: **nil**

w:sheet-exposed-p *window-or-screen* Macro

Initializes the **w:exposed-p** instance variable that determines whether to expose the window or leave it deexposed. If non-**nil**, the window is made exposable after it is created. The default is to leave the window deexposed.

The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.

:exposable-p Method of *windows and screens*

Returns **t** if the window is exposable.

:exposed-inferiors *list-of-inferiors* Method of *windows and screens*
 Default: **nil**

w:sheet-exposed-inferiors *window-or-screen* Macro

Returns a list of all exposable inferiors of this window or screen. The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.

:screen-array *array* Method of *windows and screens*
 Default: **nil**

w:sheet-screen-array *window-or-screen* Macro

Returns the window or screen's screen array, or **nil** if the screen has no array. Normally, you do not need to access the **w:screen-array** instance variable directly. **w:screen-array** is a displaced array that points to a special memory that is the bit-mapped memory for the Explorer monitor. (See the section about arrays in the *Explorer Lisp Reference* manual for details about displaced arrays and the general-purpose functions that manipulate arrays.) The Explorer system displays the **w:screen-array** on the video display. When you change an element in the **w:screen-array**, the change is immediately visible on the video display.

Each window has a **w:screen-array**, and the 0,0 element of a window's **w:screen-array** corresponds to the upper left corner of the video display. The dimensions of the **w:screen-array** are the dimensions of the screen and have no correlation to the window's **w:width** and **w:height** instance variables. (These variables contain the dimensions of the *active* part of the array, not the dimensions of the entire array.) It is possible, then, to access parts of the **w:screen-array** that are outside this instance of the window. This feature is normally important only when the **:bitblt** method and variants of the **:bitblt** method are in use. When executing a **:bitblt** method, you must ensure that only those bits that are within the window are transferred. Also,

you must ensure that the array elements that are the destination of the `:bitblt` method are correct.

The `aref` function can access individual elements of the `w:screen-array`. Remember, the upper left corner of the video display corresponds to the 0,0 element of `w:screen-array`, and the useful parts of the array are kept in the `w:height` and `w:width` instance variables.

The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.

`:deexpose` &optional (*save-bits-p* **`:default`**) Method of *windows and screens*
(screen-bits-action **`:noop`**) *(remove-from-superior t)*

Makes the window not exposed and not exposable when this method is sent. This is a useful method to add `:before` and `:after` methods to.

Arguments: The arguments to the `:deexpose` method are supplied by the system and are usually of interest only to the system's methods.

save-bits-p — One of the following:

- **`:default`** means the bits are saved if the window has a bit-save array. This is the default value.
- **`:force`** gives the window a bit-save array if it does not already have one, so the bits are always saved.
- **`nil`** does not save the bits.

screen-bits-action -- What to do to the bits on the screen. *screen-bits-action* can be either **`:noop`** (does nothing to the bits) or **`:clean`** (erases the bits from the screen). The default is **`:noop`**.

remove-from-superior — If you specify *remove-from-superior*, you should always use `t` (the default). If *remove-from-superior* is `nil`, the window remains exposable. The window system uses `nil` as part of implementing deexposure of an exposable window whose superior loses its screen array. Using `nil` at any other time leads to incorrect results.

`w:with-sheet-deexposed` *window body* Macro

Executes the code in *body* with *window* deexposed. If *window* had been exposed, it is reexposed when the code in *body* completes execution. Methods that change things about the window often make use of this macro to reduce the complicated case of an exposed window to the simpler case of a deexposed window.

Temporary Windows

5.8 Normally, when a window is exposed in an area of the screen already occupied by other exposed windows, the windows covered by the newly exposed window are deexposed automatically by the window system. This occurs because the window system does not normally leave two overlapping windows exposed. (Temporary windows are the common exception.)

Sometimes, though, windows appear on the screen for a very short time. The most obvious examples of these are the pop-up menus that appear only long enough for you to select an item. If windows were deexposed every time a pop-up menu appeared, the windows without bit-save arrays would have their

screen images destroyed, forcing them to regenerate their screen images or to reappear empty. The ones with bit-save arrays would not be damaged in this way, but they would have to be deexposed, and deexposure is a relatively expensive operation, in terms of both of memory and CPU time.

This problem is solved for pop-up menus by making them temporary windows. Temporary windows work differently from other windows in the following way. When a temporary window is exposed, it saves the pixels that it covers up, then restores these pixels when the temporary window is deexposed. These pixels may come from several different windows. In this way, temporary windows do not change the area of the screen that they use, even if they cover up some windows that do not have bit-save arrays.

Also, a temporary window, unlike a normal window, does not deexpose the windows that it covers up. In this way, the covered windows need not save their bits in their bit-save arrays (if they have bit-save arrays) nor regenerate their contents (if they do not have bit-save arrays). Regular windows never notice that the temporary window was there.

Flavors and Methods **5.8.1**

- w:temporary-window-mixin** Flavor
 Makes a temporary window.
- w:shadow-borders-mixin** Flavor
 Required flavor: **w:borders-mixin**
 Implements a window shadowing effect that makes a window appear as if it is raised away from the screen. This produces a three-dimensional effect. **w:shadow-borders-mixin** is mixed in with most flavors that produce temporary windows. The window has a default border width of 3.
- :right-shadow-width *new-width*** Initialization Option of **w:shadow-borders-mixin**
Settable. Default: 6.
- :bottom-shadow-width *new-width*** Initialization Option of **w:shadow-borders-mixin**
Settable. Default: 6.
 Sets the width of the shadow effect for the right or bottom edge of the window, respectively, to the size specified by *new-width*. The value of *new-width* is specified in pixels.
- :shadow-draw-function *function*** Initialization Option of **w:shadow-borders-mixin**
Settable. Default: **w:draw-shadow-border**
 Sets the function that draws the shadow border around temporary windows, such as the System menu.
- :temporary-bit-array** Method of *windows*
 Returns non-nil if the window is a temporary window.
- w:temporary-shadow-borders-window-mixin** Flavor
 Defines a mixin that is similar to **w:temporary-window-mixin** but that includes shadow borders.

Temp Locking

5.8.2 Problems could occur if temporary windows were this simple. Suppose that a temporary window appears over a normal window; some of the contents of the normal window are saved in an array inside the temporary window. Now, if the normal window were moved somewhere else, possibly became deexposed or overlapped by other windows, and then the temporary window was deexposed, the temporary window would restore its saved bits where the normal window used to be. This restoration would overwrite some other window.

Furthermore, even though the normal window is still exposed, output on it must not be permitted while the temporary window is exposed. Such output could overwrite the temporary window.

Because of problems like these, when a temporary window is exposed on top of some other windows, all the windows that it covers up (fully or partially) become *temp-locked*. While a window is temp-locked, any attempt to type out on it is delayed until it is no longer temp-locked. Furthermore, any attempt to deexpose, deactivate, move, or reposition a temp-locked window is delayed until the window is no longer temp-locked. The temp-locking is undone when the temporary window is deexposed.

Because of temp-locking, you should never write a program that puts a temporary window on the screen for a long time. Some action by the user, such as moving the mouse, should make the temporary window deexpose itself. While the temporary window is in place, it blocks many important window system operations over its area of the screen. The windows it covers cannot be manipulated, and programs that try to manipulate them must wait until the temporary window goes away.

Two or more temporary windows can be exposed at the same time without causing problems. If you expose a temporary window and then expose another temporary window, and they do not overlap each other, they can be deexposed in either order. Any windows that both temporary windows cover up are temp-locked until both temporary windows are deexposed. If temporary window B covers up temporary window A, then A is temp-locked exactly like any other window, so A cannot be deexposed until B has been deexposed.

`w:lock-sheet` *window* &body *body*

Macro

Implements temp-locking of windows. That is, executes *body* while *window* is locked against output and refresh. Typically, *body* includes methods rather than functions. You do not need to enclose functional interfaces such as the `w:menu-choose` function or `w:notify` within a `w:lock-sheet` macro.

The Screen Manager

5.9 Fully visible windows do not always use the entire screen. This situation does not happen in elementary use of the Explorer system, because initial windows in the system are all the same size as the screen. If you create a small Lisp Listener with the System menu Create command, the rest of the screen is unclaimed by any fully visible window. The part of the window system responsible for dealing with unclaimed parts of the screen is called the *screen manager*.

The screen manager fills unclaimed areas by looking for deexposed windows that fall entirely or partly within them. Only active immediate inferiors of the

screen are considered, and they are considered in a specific priority order described in paragraph 5.9.4, Priority Among Windows for Exposure.

Autoexposure

5.9.1 A window that falls entirely within unclaimed areas can be made visible without deexposing any other windows. This is called *autoexposure*. Because the window is a direct inferior of the screen, exposing it always makes it visible. The screen manager continues to consider the remaining deexposed windows, but with less screen area unclaimed.

A window located in an area of the screen that no other window uses can be visible to the user. However, if the window is also located in a portion of the screen used by another window, the window cannot be fully visible to the user; only that portion of the window that is outside the area used by another window can be visible. Overlapping windows are shown in Figure 5-1. The window is not treated as visible or exposed in any other sense. The windows appear to be overlapping pieces of paper on a desktop. The deexposed window is partially covered by the visible windows, but you can still see those parts that are not covered. The contents are copied from the window's bit-save array. Windows without bit-save arrays are by default ineligible for partial visibility, so other windows with a lower priority get a chance for the same screen area. It is possible, however, to arrange for windows without bit-save arrays to be partially visible (though the displayed contents may not be accurate).

A window whose size or position precludes it from fitting entirely within the bounds of its superior cannot be exposed. The screen manager does not try to autoexpose this type of window. The portion of the window that lies within its superior's bounds can be displayed as a partially visible window.

Autoselection

5.9.2 Besides controlling autoexposure, the screen manager can also select a window if no window is selected. This is called *autoselection*. A window is a candidate for autoselection if it is an exposed inferior of the screen and its `:name-for-selection` method returns a non-nil value. (This method is described in paragraph 6.3.1, The System Menu Select Command.)

The screen manager manages the inferiors of windows or sheets as well as the inferiors of screens. The system invokes the screen manager on a sheet's inferiors by sending the sheet a `:screen-manage` message. This happens for all visible sheets regardless of their flavors.

:screen-manage Method of *windows and screens*
Autoexposes and displays partially visible windows among the active inferiors of this window or screen, as described previously.

:screen-manage-autoexpose-inferiors Method of *windows and screens*
Autoexposes the active inferiors of this window or screen. This method is used in the default definition of the `:screen-manage` method.

w:no-screen-managing-mixin Flavor
Prevents the screen manager from dealing with the inferiors of a window by redefining the `:screen-manage` method to do nothing.

When a single program uses a frame, that program usually has sole control over exposure of panes. `w:no-screen-managing-mixin` prevents the screen manager from interfering. This mixin is normally not used with constraint

frames because constraint frames avoid problems while changing configurations by deactivating any panes that do not belong in the configuration. Zmacs frames use this mixin so that the screen manager does not autoexpose various editor windows that belong to the frame.

Control of Partial Visibility 5.9.3 This paragraph discusses flavors, methods, and variables that affect partial visibility of windows.

:screen-manage-deexposed-visibility Method of *windows*
 Returns non-nil if parts of this window can be displayed when the window is partially visible.

w:show-partially-visible-mixin Flavor
 If a window has this flavor mixed in, the screen manager attempts to show the window to the user when the window is partially visible, even if the window does not have a bit-save array. If the window does not have a bit-save array, there are no saved contents to display, and the screen manager must give the window a screen array temporarily, send it a **:refresh** message so it can draw itself on the screen array, and then display whatever is found there. This often means that you see the label and borders of the window, but not the inside.

w:gray-deexposed-right-mixin Flavor
w:gray-deexposed-wrong-mixin Flavor

Makes any visible parts of the window appear gray if the window is not fully visible. Thus, these flavors provide something other than blank space when the window should be partially visible. **w:gray-deexposed-right-mixin** works for all windows; **w:gray-deexposed-wrong-mixin** is faster but does not work for windows that have inferiors.

You can use either mixin in windows that have no bit-save arrays as a quicker alternative to **w:show-partially-visible-mixin**.

:gray-array array Initialization Option of **w:gray-deexposed-right-mixin**
Gettable, settable. Default: **w:12%-gray**

:gray-array array Initialization Option of **w:gray-deexposed-wrong-mixin**
Gettable, settable. Default: **w:12%-gray**

Initializes the percentage of gray displayed. The value must be a two-dimensional array of bits that **bitblt** can replicate; the array's width must be a multiple of 32. Useful values for **w:gray-array** include **w:100%-black**, **w:66%-gray**, and **w:12%-gray**. See the description of the **w:make-gray** function in paragraph 12.4.2, Bit Block Transferring, for a complete list and examples of the predefined gray patterns.

w:initially-invisible-mixin Flavor

Stops a window from appearing through screen management, even partially, until it has first been explicitly exposed. Specifically, this flavor creates window instances that have a priority of -2. This flavor is used in some window flavors (such as editor windows and others) of which instances are present in the saved system environment even without the user's ever having requested them. These windows can be active and can be selected using the **SYSTEM** key, but they do not become partially visible if some other window is made smaller.

Recall that if a deexposed window has its deexposed timeout action set to `:permit`, output on the window can proceed but is sent to the bit-save array rather than to the screen. If the window is partially visible, such output could modify the visible parts of the window. You can request that the screen manager check periodically for such output and copy the changed contents to the screen.

w:screen-manage-update-permitted-windows

Variable

Controls whether the screen manager looks for partially visible windows with deexposed timeout actions of `:permit` and updates the visible portion of their contents on the screen. If this variable is `nil`, as it is initially, the screen manager does not update the visible portion of the window. Otherwise, the value should be the interval between screen updates, expressed in 60ths of a second.

**Priority
Among Windows
for Exposure**

5.9.4 Suppose there is a portion of the screen where there are no exposed windows, and more than one active, deexposed window could be exposed to fill this area. However, two windows overlap so both cannot be exposed. The screen manager decides which window gets exposed on the basis of a priority ordering. All of the active inferiors of a window are maintained in a specific order, from highest to lowest priority. When there is a portion of the screen on which more than one active inferior might be displayed, the inferior with the highest priority is the window that gets displayed. This priority ordering is like the relative heights of pieces of paper on a desk; the highest piece of paper at any point on the desk is the one that you see, and the rest are covered.

:order-inferiors

Method of *windows*

Sorts the list of active inferiors of this window or screen into the proper order for considering them for autoexposure or partial visibility. The `w:inferiors` instance variable is the list of active inferiors. The inferior at the beginning of `w:inferiors` has the highest priority.

The default definition of `:order-inferiors` uses a complicated algorithm that puts the most recently exposed windows first but also allows the programmer to specify priorities explicitly.

The algorithm involves a value assigned to each window, called its *priority*, that can be a fixnum or `nil`. Thus, windows with higher numerical priority values have higher priority to appear on the screen. The default value for the priority is `nil`, which is considered less than any positive numeric value.

The standard ordering of inferiors puts all exposable inferiors first, followed by the unexposable inferiors in order of decreasing priority. Each group of unexposable inferiors with the same priority is ordered by how recently they were exposable. The longer an inferior has gone without being exposable, the lower the inferior's priority.

Computation of the current ordering is based on the past ordering as stored in the old value of `w:inferiors`. When the window system does anything to change the ordering, such as making a window exposable or not exposable, it invokes the `:order-inferiors` method to update the recorded ordering.

The ordering is updated by moving the exposable windows to the front and sorting the unexposable ones by priority. The sort is stable; that is, unexposable windows with the same priority value keep their previous ordering. Because numerical priorities are not typically used (that is, the priorities of most windows are `nil`), the ordering generally changes only as a result of exposing and deexposing windows. When a window becomes exposable, the window's priority increases; when other windows become exposable instead, this window's priority becomes lower. Thus, the ordering simply shows how recently each window was exposable.

:priority *priority* Initialization Option of *windows*
Gettable, settable. Default: `nil`

Sets the window's priority value. This can be a number or `nil`.

:bury Method of *windows*

Buries the window, which deexposes the window and puts it at the end of its priority grouping in the ordering. A program typically buries its window when it thinks that the user is not interested in that window and would prefer to see some other windows. The user can bury a window using the Bury command in the Edit Screen mode of the System menu.

Also, the `w:deselect-and-maybe-bury-window` function is a convenient interface to the `:bury` method. This function is described in paragraph 6.2, How Programs Select Windows.

Negative Priorities

5.9.5 Negative priorities have a special meaning. If the value of a window's priority is `-1`, then the window is never visible, even if it is only partially covered; however, it is still autoexposed. If the value of priority is `-2` or less, then the window is not autoexposed, so it is never seen unless sent an explicit `:expose` message.

Delaying Screen Management

5.9.6 At times, you may want to inhibit automatic reexposure by the screen manager. The screen manager can potentially interfere with the actions of a program that explicitly deexposes windows. Suppose you send a `:deexpose` message to an exposed window. The exposed window certainly does not overlap any still visible windows, and it is the most exposed window, so it is the first candidate for autoexposure. The screen manager runs and probably autoexposes that very window, canceling the effect of the `:deexpose` method.

Explicit deexposure is usually performed at the beginning of a sequence of window rearrangements. For example, moving an exposed window deexposes it, changes its position (which is easy when it is deexposed), and reexposes it. The screen manager should run only when the whole sequence is complete; it should not consider the transient intermediate states. Even if the screen manager did not directly interfere with the program's deliberate actions, it would waste time and confuse the user by displaying partially visible windows in temporarily unclaimed screen areas for which the program is already preparing a new use.

You can shut the screen manager off within the `w:delaying-screen-management` macro. While the body of this macro is being executed, events that would normally bring about screen management are recorded on a queue instead. After exiting the macro (whether normally or by throwing), the screen manager looks at the queue and performs all necessary screen management at once.

Sometimes screen management cannot be done when the **w:delaying-screen-management** form is exited because relevant windows are locked by other processes. Then the entries are left on the queue. They are handled at some later time when the necessary locks are freed by a background process called Screen Manager Background. The necessary screen management, then, always gets done eventually.

When **w:delaying-screen-management** forms are nested, only the outermost one does any screen management when it is exited.

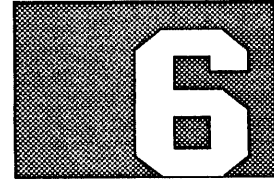
w:delaying-screen-management *body* Macro

Evaluates the forms in *body* sequentially with screen management delayed. The value of the last form is returned. For example, the system delays screen management while the user is resizing a window with the Edit Screen commands. By delaying screen management, the system redisplay the contents of the screen only once, after the user has completed all the changes.

w:without-screen-management *body* Macro

Evaluates the forms in *body* sequentially with screen management delayed. Moreover, if the body completes normally, the entries put on the queue by the body's execution are removed from the queue, on the assumption that the body has done all appropriate screen redisplay. If the body terminates abnormally with a throw, the queued entries remain on the queue and eventually are processed by the screen manager.

SELECTION



Introduction

6.1 At any given time, programs (or the system) responding to user commands can select only one window and send keyboard input to only one window. This window is called the *selected* window. A process trying to input through another window normally waits until that window is selected.

A window's cursor marker is a blinker that normally blinks only when the window is selected; thus, this cursor-following blinker usually indicates which window is selected. You can control what happens to each blinker when its window becomes selected. (See paragraph 10.2, *Visibility and Deselected Visibility of Blinkers*.)

A user can change the selected window using the **TERM** and **SYSTEM** keys or the System menu. Also, moving the mouse blinker to a window and clicking the left button one time selects that window if the window can be selected.

For example, suppose there are several Lisp Listeners displayed on the main screen at the same time. One of the windows displays a flashing blinker and echoes keyboard input there. The processes in the other screens suspend keyboard input; you can see this if one of the windows on the screen is a Peek window. The mouse or the **TERM O** keystroke sequence can be used to select a different window.

The selected window must handle some operations that windows in general do not have to handle. The `w:select-mixin` flavor defines these methods and should be used in flavors of windows that are going to be selected. The `w>window` flavor includes the `w:select-mixin` flavor. A window can be useful without being selectable.

If two processes try to read from the same window (or windows sharing an input buffer), you cannot predict which process will get the input. If you are designing an application where this might occur, you must make sure that you do not have two processes actually active and reading input from the same source at the same time. In most applications, only one process can read input from any one window or input buffer. In these applications you should use `w:process-mixin` in the window flavor to tell the window which process is associated with it.

The selected window controls the actions performed by the system at the instant a character is typed on the keyboard. Because typed-ahead commands (such as **END** in the editor) switch windows, there is no way to know for certain which window will eventually read a character being typed at a given moment. Letting the selected window determine asynchronous processing for the character avoids this problem. Asynchronous processing options include asynchronous intercepted characters (as described in paragraph 8.7.2) and case conversion of control characters (see paragraph 8.7.3, *Global Asynchronous Characters*).

The selected window determines which process the `:process` method acts on when asynchronously intercepted characters (such as the **CTRL-ABORT** keystroke sequence) are intercepted. The status line normally does the same

thing to find the process whose run state should be displayed. If you use `w:process-mixin`, the `:process` method returns the process associated with the window; otherwise, a default definition of `:process` is inherited from `w:select-mixin` and returns whichever process last read input from the window—or from any other window sharing the same input buffer. You should use `w:process-mixin` whenever possible. Not using `w:process-mixin` can lead to unexpected results in the CTRL-ABORT keystroke sequence (this is, however, acceptable for the status line).

If a process tries to get input from a window whose input buffer is empty and is not selected, it cannot get any input and must wait. The input buffer is selected if this window, or any other window sharing the same input buffer, is the selected window. The wait ends when input appears in the buffer, or when the buffer becomes selected and there is keyboard input available. If the window is not even exposed, the program can also specify a notification. The window's `deexposed` type-in action controls whether notifications can happen. See paragraph 18.2, Notifications.

How Programs Select Windows

6.2

`w:selected-window`

Variable

The selected window. The system maintains the variable; you should not set it. Note that constraint frames do not use this variable; instead, they use the `w:selected-pane` method.

`w:select-mixin`

Flavor

Required flavor: `w:essential-window`
Required instance variable: `w:io-buffer`

To be selected, the flavor for a window must include the `w:select-mixin` flavor. `w:select-mixin` is part of the `w>window` flavor but not part of the `w:minimum-window` flavor.

A window whose flavor does not contain this mixin flavor can be sent the `:select` message only if the window has designated another window as a selection substitute. The last substitute selected must have `w:select-mixin`.

`:select` &optional (*remember-previous* t)

Method of *windows*

Makes this window (or its selection substitute, if any) the selected window. Many application window flavors define `:before` and `:after` methods for this method. The `:before` and `:after` methods run whenever this method is invoked, even if the window is already selected. To test for a window being selected, use the `:self-or-substitute-selected-p` method described in paragraph 6.4, Selection Substitutes.

If *remember-previous* is not `nil` (and only then), the previously selected window is entered on the list of previously selected windows for the TERM and SYSTEM keys to use. The default is `t`.

Neither the `:select` nor the `:mouse-select` methods should be called in the mouse process. If you want to use these methods in a `:mouse-click`, `:mouse-buttons`, or `:handle-mouse` method, you must include the following in your code:

```
(process-run-function "Select" window-to-select :select)
```

This causes the `:select` method to be executed in its own process.

`:mouse-select` *args* Method of *windows*

Selects a window for mouse click or asynchronous keyboard input such as is done when the TERM key is pressed. The *args* are sent to the `:select` method.

While the `:mouse-select` method is generally the same as sending a `:select` message to the window's alias for selected windows (as explained in paragraph 6.3, Teams of Windows), the `:mouse-select` method does not change which window gets type-ahead input. (See the description of `w:*terminal-keys*` in paragraph 8.7.3, Global Asynchronous Characters, for more details about specifying where type-ahead input should go.)

Neither the `:select` nor the `:mouse-select` methods should be called in the mouse process. If you want to use these methods in a `:mouse-click`, `:mouse-buttons`, or `:handle-mouse` method, you must include the following in your code:

```
(process-run-function "Select" window-to-select :mouse-select)
```

This causes the `:mouse-select` method to be executed in its own process.

`:deselect` &optional (*restore-selected* *t*) Method of *windows*

Invoked automatically when a window ceases to be selected, whether because the window is no longer visible, or because another window is being selected. Many application window flavors define `:before` and `:after` methods for the `:deselect` method.

restore-selected controls whether this window is stored in the `w:previously-selected-windows` array used by the TERM S and SYSTEM keys. The `:deselect` method determines whether to select this or some other window found in that array. The possible values for *restore-selected* are the following:

- `:dont-save` — Does not put the window being deselected in the array, and no other window is selected.
- `:beginning` or `nil` — Puts the window being deselected at the front of the array, and no other window is selected.
- `:end` — Puts the window being deselected at the end of the array, and no other window is selected.
- `:first` — Selects the window at the front of the array, then puts the window being deselected at the front of the array. This is similar to what the TERM S keystroke sequence does.
- `:last` or `t` — Puts the window being deselected at the end of the array and selects the window at the front of the array. This is the default.

`w:deselect-and-maybe-bury-window` *window* *deselect-mode* Function

Deselects *window* and selects the previously selected window. If this deexposes the deselected window, then the deselected window is buried. *deselect-mode* is passed to the `:deselect` method and is used as its *restore-selected* argument. See the previous discussion of `:deselect` for its possible values.

w:window-call (*window-to-select* &optional *exit-method* *exit-args*) &required *body* Macro

w:window-mouse-call (*window-to-select* &optional *exit-method* *exit-args*) &required *body* Macro

Executes *body* on the window specified as *window-to-select*. **w:window-call** uses the `:select` method; **w:window-mouse-call** uses the `:mouse-select` method. On exit, both macros reselect the window that had been selected before and send the selected window the specified arguments.

exit-method is invoked after *body* has been executed. The value used for *exit-method* is often `:deactivate`. If *exit-method* is omitted, nothing is done to *window-to-select* except for deselecting it because some other window is selected. *exit-args* are passed, along with *exit-method*, to *window-to-select*.

For example, the following code creates an instance of a mouse-sensitive window along with its item types. The **w:window-call** form exposes and selects the window, outputs a mouse-sensitive sentence, waits for the user to make a choice, returns the value of that choice, deactivates the mouse-sensitive window, and selects the previous window.

```
(defflavor mouse-sensitive-window ()
  (w:basic-mouse-sensitive-items w:window))

(setq my-window (make-instance 'mouse-sensitive-window
  :item-type-alist
  `((word-item left-click-word "a word type item.")
    (phrase-item left-click-phrase "a phrase type item.")
    (sentence-item left-click-sentence "a sentence type item.))))

(w:window-call (window :deactivate)
  (format my-window "this -M contains -VM mouse-sensitive -1M."
    "sentence"
    'word-item "some"
    "words and phrases")
  (send my-window :any-tyi)
  )
```

You may find that using selection substitutes is better than using **w:window-call** for controlling selection among windows of a team. (Selection substitutes are described in paragraph 6.4.)

Teams of Windows

6.3 The principle of selecting a single window is based on the concept of windows that are independent competitors for the user's input, such as a pair of Lisp Listeners. Normally a team of windows is a single frame and its panes, managed by a single process. The windows of a team often share an input buffer to make it easier for one process to read input from all of the windows at one time. This is an important procedure, which you should read about if you are designing a team of windows. Note that a constraint frame is a specific type of team.

Teams are not actual Lisp objects but merely concepts understood by the user and programmer. The window system cannot have a selected team; one window of the team must be selected. Each team's program selects a window of the team as the team's selection representative. The selected window should then be the selection representative of your selected team. The selected window can change when the user selects a new team, or when your selected team chooses a new selection representative.

To implement this, the programmer of the team first selects one window of the team to be the leader. This is not the same as the selection representative. The selection representative can change from moment to moment, but the leader must be fixed. When the team is a frame and its panes, it is natural to make the frame be the leader. Standard mixins are provided to make this easy to do. These mixins and the procedures for using them are described in the following paragraphs.

The selection representative is implemented as the leader's selection substitute. Then the team can be selected using the `:select` method on its leader window.

Even when the team allows the user to select from the windows of the team (such as when a Zmacs frame in two-window mode allows the user to click a mouse button on either of the editor windows to select the window), this selection is best represented by the idea of a team that does all selection under program control. The appropriate mouse clicks are defined as commands that tell the team's program to change the team's selection representative.

You usually want only a single item representing the team to appear in the Select menu, invoked from the System menu. If the team consists of a frame—which is the leader—and its panes, selection can be done with `w:inferiors-not-in-select-menu-mixin` in the frame's flavor. More complex behavior is also possible. For example, Zmacs frames in two-window mode allow each editor window to have its own entry in the Select menu.

Also, the `TERM` and `SYSTEM` keys should reselect the team by selecting its current selection representative. This is done by making them record and reselect the team's leader. If the team consists of a frame—which is the leader—and the frame's panes, selection can be done with `w:alias-for-inferiors-mixin` in the frame's flavor. (Zmacs frames follow this pattern exactly. The frame is the alias for any editor windows inside the frame.)

The following paragraphs describe the details of how these things are done.

The System Menu Select Command

6.3.1 When the Select command in the System menu is used, it gets the list of alternatives by invoking the `:selectable-windows` method on each screen. This method travels down the window hierarchy and determines whether each window should be included. Each window is sent a `:name-for-selection` method. The value sent should be either `nil` (meaning omit this window) or a string (to be used as the window's name). If the value is a string, the string is displayed in the menu of windows.

w:inferiors-not-in-select-menu-mixin Flavor
Required flavor: `w:basic-frame`

Redefines the `:selectable-windows` method to ignore the window's inferiors. The inferiors are not asked whether they should be included.

:selectable-windows Method of *windows*

Returns an association list of strings versus windows that become part of the association list displayed in the Select menu. The association list returned should describe this window and its inferiors, or whichever of them ought to appear in that menu.

Normally this method uses the string returned by a window's `:name-for-selection` method as the first element of the association list, or it omits this window if its `:name-for-selection` returns `nil`. `:selectable-windows` then appends the values returned from the window's inferiors.

:name-for-selection

Method of *windows*

Either returns a string to display in the Select menu for this window, or returns `nil` (meaning do not list this window in the menu).

The default definition uses the window's label string, if any, or its name. Many applications redefine `:name-for-selection`. For example, `w:not-externally-selectable-mixin` redefines `:name-for-selection` to return `nil`. If you want more complex behavior from a team than simply having a single entry, you redefine `:name-for-selection` on the flavors of many different windows in the team.

The `:name-for-selection` method also affects autoselection, which is done by the screen manager. A window can be autoselected only if its `:name-for-selection` is not `nil`.

**Selection With
TERM and
SYSTEM Keys**

6.3.2 The TERM S keystroke sequence can be thought of as acting on a combined list that contains the selected window followed by the previously selected windows. The TERM *n* S keystroke sequence rotates the first *n* elements of this list, makes the selected window the first previously selected window, and makes the *n*th previously selected window the selected window. The SYSTEM key also uses this database to find a window of the appropriate flavor to select or to rotate through all the windows of that flavor.

Windows are put into the `w:previously-selected-windows` array and removed from it automatically when they are selected, deselected, activated, or deactivated. The applications programmer must identify only teams of windows that should be treated as a unit. The system uses the `:alias-for-selected-windows` method to determine whether a window can be selected. If two windows are in a hierarchy, one above the other, and both have the `w:alias-for-inferiors-mixin` flavor, then the window having the higher priority is selected.

NOTE: No record is kept of which window in a team was actually selected most recently. The `w:previously-selected-windows` variable records only the alias or team leader window, and this is the window that receives the `:select` message if the TERM key is pressed to switch back to that team. To ensure that the proper window within the team is selected, use selection substitutes as described in the following paragraphs.

w:not-externally-selectable-mixin

Flavor

Makes a window and its descendants have the window's superior as an alias and keeps the window out of the Select menu.

Using this flavor you can control which windows appear in the Select menu or which can be selected by the TERM key. Specifically selected descendants are given this mixin so that they do not appear in the Select menu; any other descendants do appear in the Select menu.

w:alias-for-inferiors-mixin

Flavor

Makes a window be an alias for all of its inferiors. Thus, the window and all of its inferiors form a team considered as a unit by the TERM and SYSTEM keys, and this window is the leader.

:alias-for-selected-windowsMethod of *windows*

Returns the alias that represents this window in the **w:previously-selected-windows** variable. When this window gets deselected, its alias is recorded in that variable. In the example of independent Lisp Listeners, the alias of each Lisp Listener is itself. For a window in a team, this method should return the team's leader window.

The default definition of this method either returns the superior's **:alias-for-inferiors** method if that is non-nil, or it returns this window.

:alias-for-inferiorsMethod of *windows*

Returns a window to act as the alias for all inferiors for all levels of this window, if needed. Otherwise, it returns nil.

The default definition returns the **:alias-for-inferiors** for this window's superior. Thus, if an ancestor of this window wants to be an alias for all of its descendants, its request is sent to whatever invokes this method; otherwise, the descendants decide for themselves.

w:previously-selected-windows

Variable

An array whose contents are all the active windows—not including the selected window—that the TERM and SYSTEM keys should know about for window selection. The windows of a team are generally all represented by a single member of the team, which is the leader. The leader is usually a frame that contains the remainder of the team as panes, but this is not required.

Typically, you want the process associated with the selected window to run with a higher priority than processes associated with deselected windows. For example, if you do a **make-system** in a Lisp Listener, you probably would like to go to an editor window and still get good response. Two variables enable this: **w:*selected-process-priority*** and **w:*deselected-process-priority***.

w:*selected-process-priority*

Variable

Default: 0

The priority of the process associated with the selected window is raised to **w:*selected-process-priority***. The process priority is reset to its previous value when a new window is selected. Setting this variable to nil causes process priorities to be unaffected by window selection.

You should not set this variable to a value greater than zero to avoid problems caused by locking out system processes running at the default priority of zero. To avoid this problem, you can set the following variable to a negative number, thus effectively *lowering* the priority of processes associated with deselected windows.

w:*deselected-process-priority*

Variable

Default: -1

When a window is deselected, the priority of the process associated with it is normally reset to what it was before the window was selected. When this previous priority is equal to **w:*selected-process-priority***, the priority of the deselected window is set to **w:*deselected-process-priority*** instead of its previous value.

The window system includes safeguards so:

- When a process priority is set above zero, the window system never lowers the priority.
- When a process priority is set below zero, the window system does not raise the priority when the window is deselected.

However, when a window is selected, the window system raises the window's priority to **w:*selected-process-priority***, which is typically the correct action.

Selection Substitutes

6.4 All windows have the ability to designate a *selection substitute*. If a window has a substitute, any requests to select or deselect the original window are passed along to the substitute. The substitute can have a substitute of its own, and so on. A window's selection substitute is remembered in the **w:selection-substitute** instance variable, whose value is either another window or **nil**.

The main use of selection substitutes is for controlling selection within a team of windows. The team has one window designated as the leader; all requests to select members of the team are accomplished with **:select** methods. Selection is made as described in the previous paragraphs. As a result, the team's program can choose a selected window within the team by making that window the leader's selection substitute.

To avoid paradoxical results when pressing **TERM S**, the **:alias-for-selected-windows** method should be the same for both the substitute window and for the window for which it substitutes. With a hierarchical team of windows, this substitution is usually arranged by using **w:alias-for-inferiors-mixin** in the top window of the team. The substitute window should not appear in the **Select** menu, because its entry and the entry of the window for which it substitutes would be duplicates. The **w:inferiors-not-in-select-menu-mixin** flavor in the top window of the team prevents the duplicate entry.

NOTE: When the team's program uses the **:set-selection-substitute** method on the team's leader window to change the selected pane within the team, it does not matter whether the team is currently selected. The correct results occur regardless of whether the team is deselected and reselected at any given time.

- :selection-substitute** Method of *windows*
Settable. Default: *nil*
 Returns this window's selection substitute or *nil* if the window does not currently have one. For **:set-selection-substitute**, if this window or its substitute was previously selected, then the window's new substitute (or the window itself) is selected afterward. Thus, the value of **:self-or-substitute-selected-p** on this window is not changed by this method.
- :ultimate-selection-substitute** Method of *windows*
 Returns this window's substitute, then the substitute's substitute, and so on until a window is reached that has no substitute. If this window has no substitute, the window itself is returned.
- :self-or-substitute-selected-p** Method of *windows*
 Returns *t* if this window, or its substitute, or its substitute's substitute, and so on, is selected; otherwise, it returns *nil*.
- w:with-selection-substitute** (*window for-window*) *body* Macro
 Executes a body of Lisp code with *window* defined as the selection substitute for *for-window*. On exit, **w:with-selection-substitute** sets the selection substitute back to whatever it was previously and *deexposes* or *deactivates* the substitute, if appropriate.
 You should use **w:with-selection-substitute** to switch the selected pane temporarily.
- w:preserve-substitute-status** *window body* Macro
 Executes *body*, then selects *window* if *window* or its substitute had been selected immediately before this macro was executed.

Typeout Windows and Selection Substitutes

6.4.1 Typeout windows use the selection substitute mechanism to select themselves to receive output. The typeout window makes itself the substitute of a suitable superior in the hierarchy. The typeout window does not necessarily substitute itself for its immediate superior. This allows output to go to the typeout window although the typeout window's immediate superior is not the selected window. For example, when you press the META-X HELP key-stroke sequence in the Zmacs editor, a typeout window substitutes itself for the selected window so you can use the Help facility.

When a typeout window substitutes itself for a superior, the typeout window may not have been the substitute for that superior. To resolve this problem, the typeout window records the selection substitute of the superior, makes itself the selection substitute, and *exposes* itself. Once the typeout window is *deexposed*, it restores the original selection substitute for the superior.

- :remove-selection-substitute** *window-to-remove suggested-substitute* Method of *windows*
 Ensures that *window-to-remove* is not this window's substitute, suggesting *suggested-substitute* (possibly *nil*) as a substitute instead. The standard implementation of this method sets the substitute of this window to *suggested-substitute* if the substitute of this window was *window-to-remove*. This method is used and documented so that specific windows can define their own ways of calculating the new value for the substitute, perhaps ignoring *suggested-substitute*.

When a typeout window is deactivated, `:remove-selection-substitute` ensures that the typeout window ceases to be another window's substitute.

Nonhierarchical Selection Substitutes

6.4.2 Some programs need to temporarily replace one window with another. For example, the `telnet` function can behave this way, giving the appearance of temporarily changing the Lisp Listener or other window in which it is called into a Telnet window. Substitution is accomplished by creating a suitable Telnet window and making it the substitute for the original window. In this case, the substitute window has the same edges and the same superior as the original window, but it is not an inferior of the original window. The substitute window need not be the same size as the original.

Noninferior selection substitutes are usually established and deestablished by using the `w:with-selection-substitute` macro in a straightforward manner. However, you must be sure that the original window is the `:alias-for-selected-windows` method of the substitute. In the case of the `telnet` function, this substitution creates the illusion that the Lisp Listener has changed temporarily into a Telnet window. Because the substitute window is not a descendant of the original one, it must have some other way to find the original window (such as an instance variable for this specific purpose) and a specially defined `:alias-for-selected-windows` method to return the original window.

The Status of a Window

6.5 A window's status is represented by a keyword that encodes whether the window is selected, exposed, or active.

`:status`

Method of *windows*

Settable.

Returns the status of the window, which is one of the following:

Keyword	Status of the Window
<code>:selected</code>	The selected window.
<code>:exposed</code>	Exposed but not selected. The window may not be visible if its <code>deexposed</code> typeout action is <code>:permit</code> .
<code>:exposed-in-superior</code>	Exposable but its superior has no screen array.
<code>:deexposed</code>	Active in its superior but not exposable.
<code>:deactivated</code>	Not active. This is the status of a window that has been created but never exposed or selected.

The `:status` and `:set-status` methods are useful for selecting a window temporarily and then restoring everything as it was. `:set-status` can `deexpose` the window or `deactivate` it in addition to `deselecting` it.

Windows and Processes

6.6 A self-contained interactive system that has its own window(s) usually has its own process to drive the window(s). Peek, Zmacs, and the Inspector all do this when called through the SYSTEM key. Normally each window you create has its own process. For example, there is a process for each Peek window, so different Peek windows run independently.

Whether a window is managed by a dedicated process or by various processes is not a crucial decision. The program that reads commands from the window and draws on the window can always be run in one dedicated process or in different processes at different times—though if you run it in two processes at one time, you should be careful to keep the processes from confusing each other. (See paragraph 6.3, Teams of Windows.)

The mechanisms of selection and exposure control whether input and output are possible on a window at a given time, and these mechanisms work automatically on any process(es) that tries to do input or output. When there is a dedicated process for a window, often the only connection between the process and the window is that the dedicated process is running a program that has a pointer to that window (typically the value of `*terminal-io*` in the process is that window). Thus, when in another process, setting `*terminal-io*` to the window where output is to go causes the output of functions such as `print` to send their output to the correct window by default.

The Inspector Example

6.6.1 For example, the Inspector window you invoke with the SYSTEM I keystroke sequence has a dedicated process, whereas the Inspector window you get by typing (`inspect`) while in a Lisp Listener runs in the process that calls the Inspector program. These two Inspector windows have the same flavor; the same function, `w:inspect-command-loop`, does the main work. The only differences between these two Inspector windows are in deciding when to deexpose the window, what to do when that happens, when it can be reused, what to do if the user presses the END key, and anything else related directly to the difference in the two user interfaces for entering and exiting.

The Inspector window makes an instructive example for comparing these two ways of managing a window. The `inspect` function allocates a window out of a resource of reusable windows of the correct flavor. The `inspect` function sends the window some messages to initialize the Inspector window for this particular session; this is how `inspect` tells the window about the object that is the argument to `inspect`. Then it selects the window using the `w:window-call` macro and calls the Inspector program. When the user presses the END key, the program returns, the `w:window-call` macro reselects the old window and deactivates the Inspector window, and `inspect` returns. The `inspect` function uses an `unwind-protect` macro so that aborting outside of `inspect` for any reason brings back the old window.

Pressing the SYSTEM I keystroke sequence finds or creates an Inspector window of the same flavor. When no initialization options are specified, this flavor's default initialization property list specifies the creation of a process, which is initialized to call the Inspector program. If the user presses the END key—or clicks on Exit—and the Inspector command loop returns, the top-level function in the dedicated process buries the Inspector window and loops back to the beginning. That is all that is required to make the SYSTEM I keystroke sequence work.

**Process-Related
Methods and Flavors**

6.6.2 The following process-related methods are defined on the `w:select-mixin` flavor so that they are always supported by the selected window. Because windows lacking `w:process-mixin` do not explicitly remember a process, a heuristic is used to produce a process: it is the last process to have read input from this window's input buffer.

When you use both `w:process-mixin` and `w:select-mixin`, you should always put the `w:process-mixin` flavor before the `w:select-mixin` flavor in the components of a window flavor, so the methods in `w:process-mixin` override methods in `w:select-mixin`.

The `w:select-mixin` and `w:process-mixin` flavors both have a `:process` method. When you use the `:process` method of the `w:process-mixin` flavor, it replaces the `:process` method of the `w:select-mixin` flavor.

w:process-mixin

Flavor

Provides the `w:process` instance variable that can remember a process associated with the window. Windows that are sometimes used with a dedicated process should use this mixin.

The most valuable service that `w:process-mixin` provides is an easy way to create and initialize a process for each window created and to inform the process which window it was created for. Once this is done, the desired results generally follow without special effort.

Selecting the window or making it visible gives the process a run reason. The window itself is used as the run reason. (See the description of processes in the *Explorer Lisp Reference* manual for details on run reasons.) Also, this resets the process if it is flushed, waiting with `false` as its wait function.

The `:kill` method on the window calls the `:kill` method on the process.

Using `w:process-mixin` guarantees that the `:process` method returns the explicitly specified process, regardless of which process has most recently read from the window.

:process *process*Initialization Option of `w:process-mixin`

Gettable, settable. Default: `nil`

Sets the process associated with the window either to *process* or to `nil`. *process* can be either a process or a list used as a description for creating a process. The following shows what the list looks like:

(initial-function make-process-options)

When the process starts up, it calls *initial-function* with the window as its sole argument. Normally *initial-function* should bind `*terminal-io*` to the argument.

:processMethod of `w:select-mixin`

Settable.

Returns a process with this window, heuristically if necessary. The `:process` method provided by `w:select-mixin` is invoked if the window was not created using the `w:process-mixin` flavor.

:processes Method of *windows*

Returns a list of processes dedicated to this window. The **:append** method combination is used so that all the processes used by any of the methods are put into the returned list. These are the processes that the **:kill** method kills.

The default is to return **nil** if no processes are found. The **w:process-mixin** flavor contains the **w:process** instance variable, which holds the name of the process associated with this window. Therefore, adding **w:process-mixin** to the window flavor provides the information needed by **:processes**.

:arrest Method of **w:select-mixin**
:un-arrest Method of **w:select-mixin**

Arrests or unarrests the process returned by the **:process** method.

:call Method of **w:select-mixin**

Selects an idle Lisp Listener window, which could be this window. If the window selected is not this window, **:call** arrests this window's process with arrest reason **:call**. This arrest reason is removed automatically by selecting this window again.

w:reset-on-output-hold-flag-mixin Flavor

Resets any process that tries to draw on this window when it has an output hold condition. (See the **:reset** method in the description of processes in the *Explorer Lisp Reference* manual.) Specifically, this flavor sets **deexposed-typeout-action** to **'#** for its window instances, which means that the **:reset-on-output-hold-flag** method is invoked, sending a **:reset** message to the current process.

The **w:truncating-pop-up-text-window-with-reset** flavor creates a temporary window that truncates lines and resets processes that try to output on it when its output hold is set.

Associating a Process With a Window

6.6.3 In most cases when you want to associate a process with a window you also want to make that window selectable—that is, you want the user to be able to select the window by pressing a **SYSTEM** keystroke. This procedure is described in paragraph 8.7.3, Global Asynchronous Characters.

If you want to create a window with an associated process that is *not* invoked by pressing the **SYSTEM** keystroke, you must do something similar to the following code. This code enables the Profile utility to be selected from a listing in the System menu.

```
(defmethod (profile-frame :after :init) (&rest ignore)
  "Sets up the various panes."
  (setq selection-menu-pane (send self :get-pane 'selection-menu-pane)
        action-menu-pane (send self :get-pane 'action-menu-pane)
        cvv-pane (send self :get-pane 'cvv-pane))
  (unless (boundp 'w:typeout-window)
    (setq w:typeout-window
          (make-instance 'w:typeout-window
                        :deexposed-typeout-action '(:expose-for-typeout)
                        :font-map '(fonts:tr12 fonts:tr12i fonts:tr12b fonts:tr12bi)
                        :io-buffer w:io-buffer
                        :superior self)))
  (setq tv:process (make-process w:name :special-pdl-size 4000
                                :regular-pdl-size 4000))
  (process-preset tv:process self :command-loop)
  (send tv:process :run-reason self)
  (send selection-menu-pane :add-highlighted-value '*important-variables*))
```

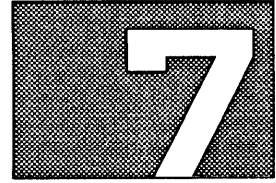
Handling a Long-Running Process

6.6.4 Sometimes you might want to execute a `make-system` or some other long-running process in a Lisp Listener, but you do not want to wait for the function to complete before you can perform some other task. Yet if you `deexpose` the Lisp Listener, the process executing in the Lisp Listener may stop if it must perform any `timeout` on the `deexposed` Lisp Listener.

The following example shows a solution to this problem: set the Lisp Listener's `deexposed-timeout` action to `:permit` and the process priority to a low value. Because the default `deexposed` `timeout` action for the Lisp Listener is `:permit`, this example is present only for illustrative purposes.

```
(defun window-permit (flavor &optional priority quantum)
  "Find a flavor window and set its deexposed-timeout-action to
  :permit.
  If the optional priority parameter is given, set the window's
  process priority.
  If the optional quantum parameter is given, set the window's process
  quantum."
  (check-arg priority (or (null priority) (numberp priority))
    "a number")
  (check-arg quantum (or (null quantum) (numberp quantum)
    (plus quantum)) "a positive number")
  (let ((this-window (or (and (eq (typep w:selected-window) flavor)
    w:selected-window)
    (w:find-window-of-flavor flavor))))
    (unless (boundp 'this-window)
      (setq this-window (make-instance flavor
        :superior w:default-screen))
      (send this-window :activate))
    (send this-window :set-deexposed-timeout-action :permit)
    (when priority
      (send (send this-window :process) :set-priority priority))
    (when quantum
      (send (send this-window :process) :set-quantum quantum))))

(window-permit 'w:lisp-listener -2 15.)
(window-permit 'w:telnet)
```

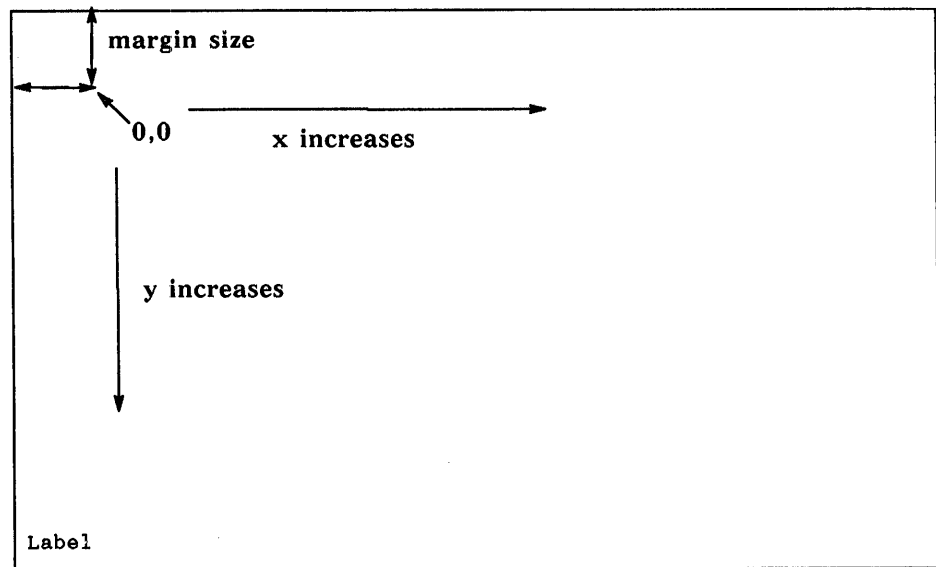


OUTPUT OF TEXT

Introduction

7.1 All windows can function as output streams, displaying output as if it were on the screen of an ordinary video display terminal. The `w:minimum-window` flavor implements the methods of the Explorer output stream protocol as well as many additional output methods such as `:insert-line`. (See the *Explorer Input/Output Reference* manual for a description of the standard output methods.) Every window has a current *cursor position*; its main use is to show where to put characters that are drawn. Windows handle typeout by drawing characters at the cursor position and moving the cursor position forward past the just-drawn character.

Cursor position arguments to stream methods are always expressed as inside window coordinates, that is, coordinates relative to the upper left corner of the inside part of the window; therefore the margins do not count in cursor positioning. The cursor position always stays in the inside portion of the window and never enters the margins. The point (0,0) is at the top left corner of the window; x coordinate values increase to the right, and y coordinate values increase towards the bottom.

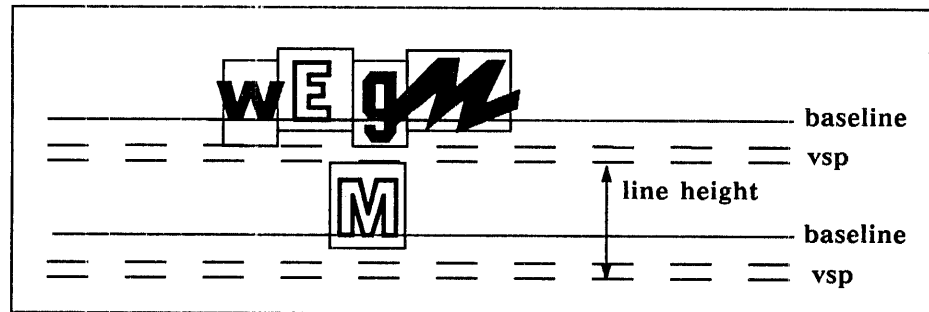


The x cursor position is the position of the left edge of the character box of the next character output. The leftmost nonzero pixels of the character can be either left or right of the edge of the character box, according to the left kern of the character. (See paragraph 9.8, *Format of Fonts*, for more details about left kern values.)

The y cursor position is the position of the top of the character on the line being output. If only a single font is in use, the top of the character box is at this vertical position.

In fact, characters are positioned so that their baselines come out on the baseline of the line. When characters of different fonts are used in the line,

they come out with baselines aligned rather than with their top edges aligned. The position of the character's baseline is a property of its font. The window's baseline is computed from the set of fonts in use to provide enough space above the baseline for any of the fonts.



Every window has a *font map*, an array of fonts in which characters on the window can be typed. At any given time, one of these is the window's *current font*. The methods that type out characters always use the window's current font. For now, fonts are described only enough to explain the *character width* and *line height* of the window; these two units are used by many of the methods documented in this section. The character width is the *char width* attribute—the amount of space that the cursor moves when it prints the character, which is typically the width of the character—of the first font in the font map.

The line height is the sum of the vertical spacing of the window and the maximum of the *char height* of the fonts. The vertical spacing is an attribute of the window that controls how much vertical spacing there is between successive lines of text. That is, each line is as tall as the tallest font is, and you can add vertical spacing between lines by controlling the vertical spacing of the window. Methods for controlling the vertical spacing are documented in paragraph 7.10, *Window Parameters Affecting Output*. No instance variable holds the vertical spacing, but the system can compute it from the line height and the font map.

Every window has a *current font*, which the window's methods use to determine the font to type in. If you are not interested in fonts, you can allow the system to supply a default font. In some fonts, all characters have the same width; these are called *fixed-width fonts*. The default font is an example of a fixed-width font. In other fonts, each character has its own width; these are called *variable-width fonts*. With variable-width fonts, it is not meaningful to express horizontal positions in numbers of characters, because different characters have different widths.

When a character is drawn, it is combined with the existing contents of the pixels of the window according to an ALU argument. The different ALU arguments for a monochrome environment are described in paragraph 12.2, *ALU Arguments*. When characters are drawn, the value of the window's `w:char-aluf` instance variable is the ALU argument used. Normally `w:char-aluf` says that the bits of the character should be bit-wise logically ORed with the existing contents of the window. This means that if you type a character, set the cursor position back to where it was, and type out a second character, then the two characters both appear, ORed together one on top of the other. This is called *overstriking*. Erasure is also done using an ALU argument, which the window can specify, called the `w:erase-aluf` instance variable.

Normally this is an ALU argument that ANDs the old pixel value with the complement of the area erased.

Windows display reverse video by interchanging the normal values of the `w:char-aluf` and `w:erase-aluf` so that erasing an area sets it to 1 and drawing a character clears the character's pixels to 0.

The same ALU operations that are available in the monochrome environment are also available in the color environment, as well as other operations. The color ALU arguments are described in paragraph 19.6, Color ALU Functions. Anything drawn on the window using the general ALU arguments writes in the foreground color and erases to the background color. If an existing application uses the general arguments, in most cases you need do nothing to update them for color. However, if an existing application uses `w:alu-xor` to draw output to the window, you should use a color ALU function while running in a color environment.

If you plan to use your application on both color and monochrome systems, you should use `w:char-aluf` or `w:combine (w:alu-transp)` and `w:erase-aluf` or `w:erase (w:alu-back)` for the two general ALUs. These two ALUs have the desired effect on a color system while behaving as if they were `w:alu-ior` and `w:alu-andca` on a monochrome system. However, `w:alu-transp` and `w:alu-back` are not merely different names for `w:alu-ior` and `w:alu-andca`.

`w:char-width` Instance Variable of *windows*
`w:sheet-char-width` *window* Macro

Contains the character width of the window. The character width is not used for ordinary output, because each font determines its own widths. `w:char-width`, which is expressed in pixels, is used for interpreting cursor positions expressed in characters. The macro can access the instance variable to get or change the value of the variable, depending on how the macro is used in code.

`w:line-height` Instance Variable of *windows*
`w:sheet-line-height` *window* Macro

Contains the line height of the window. The line height is actually used for outputting a `#\return` character. `w:line-height`, which is expressed in pixels, is used for interpreting cursor positions expressed in lines.

How a Character Is Displayed

7.2 Typeout does more than just draw a character on the screen; typeout also handles these situations:

- Moving the cursor to the proper place
- Processing nonprinting characters reasonably
- Attempting to display outside the edges of the window
- Enabling **MORE** processing, which causes **MORE** breaks to occur at the proper time

Figure 7-1 shows a pseudo-code example of the logic that types out characters to the window. For information about the Explorer character set, see the discussion of input functions in the *Explorer Input/Output Reference* manual.

The following paragraphs use the sharp-sign backslash construct for special character codes. This construct is discussed in the section on characters in the *Explorer Lisp Reference* manual.

Figure 7-1

Pseudo-Code for Character Displaying

```

    Handle output exceptions                                     ①
IF character is printable
THEN
    print the character                                       ②
    move the cursor to the right by the width of the
    character
ELSEIF character is a #\tab
THEN
    move the cursor to the next tab stop                       ③
ELSEIF character is a #\return
THEN
    IF cr-not-newline flag = 0                                ④
    THEN
        move the cursor to the far left and down one line
    ELSE
        display the character as a lozenge
    ENDIF
ELSEIF character is a #\backspace
THEN
    IF backspace-not-overprinting flag = 0
    THEN
        move the cursor left by one character width           ⑤
        but not past the inside-left edge
    ELSE
        display the character as a lozenge                     ⑥
    ENDIF
ELSEIF character is a defined character code
THEN
    display the character as a lozenge                         ⑦
ELSE
    display the character's octal code number as a lozenge    ⑧
ENDIF

```

In more detail, the system uses the following procedure to output characters:

- ① First, any output exceptions that are present are dealt with and made to disappear. (See paragraph 7.4, Output Exceptions.)
- ② If the character is a printable character, the character is typed in the current font at the current cursor position, and the cursor position moves to the right by the width of the character.

If the character is one of the format constructs `#\tab`, `#\return`, or `#\backspace`, the character is handled in a special way.

- ③ The `#\tab` construct causes the cursor to move one position to the right to the next tab stop, moving at least one character width. Tab stops are equally spaced across the window. The `:tab-nchars` initialization option sets the distance between tab stops using the `w:char-width` instance variable. The `:tab-nchars` initialization option defaults to eight but can be changed.

- ④ Normally, the `#\return` construct causes the cursor position to be moved to the inside left edge of the window one line down, and clears the line. The `#\return` construct also initiates **MORE** processing and the end-of-page condition processing. However, if the window's `:cr-not-newline-flag` initialization option is on, the `#\return` construct is not regarded as a format construct and is displayed as `<RETURN>`, like other special characters.

If the character being typed out is `#\backspace`, the result depends on the value set by the window's `:backspace-not-overprinting-flag` initialization option.

- ⑤ If `:backspace-not-overprinting-flag` is 0, as is the default, the cursor position is moved left by one character width, or to the inside left edge, whichever is less.
- ⑥ If `:backspace-not-overprinting-flag` is 1, `#\backspace` is treated like all other special characters.
- ⑦ All special characters other than `#\tab`, `#\return`, or `#\backspace` have their names typed out in tiny letters surrounded by a lozenge, and the cursor position is moved right by the width of the lozenge. For example, `\#escape` would be displayed as `<ESCAPE>`.
- ⑧ If an undefined character code is typed out, the character is treated like a special character; the octal code number of the special character is displayed in a lozenge.

Stream Output

7.3 The following paragraphs discuss methods affecting stream output.

<code>:tyo char &optional font color</code>	Method of <i>windows</i>
<code>:sheet-tyo window char &optional font color</code>	Function

Types a character specified by *char* on the window (either the current window or *window*) at the current cursor position and advances the cursor to the next position. *font* is the font object to use when typing out the character. If *font* is not specified, `:tyo` uses the current font.

The optional *color* argument specifies the value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is `w:green`. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of `w:sheet-foreground-color` (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

<code>:string-out string &optional (start 0) (end nil) color</code>	Method of <i>windows</i>
<code>:line-out string &optional (start 0) (end nil) color</code>	Method of <i>windows</i>
<code>:fat-string-out string &optional (start 0) (end nil) color</code>	Method of <i>windows</i>

Types a string on the window. These methods are more efficient than moving characters singly.

`:string-out` performs as if each character in the string, or the specified substring, were printed with the `:tyo` method, but `:string-out` is much faster. `:line-out` does the same thing as `:string-out` and then advances the cursor to

the next line, like typing a `#\return` construct. The execution of `:line-out` is not affected by the `:cr-not-newline-flag` initialization option.

`:fat-string-out` types a string on the window using the font attribute of each character. The window's current font is not used. This method is similar to the one used by the Zmacs editor to display lines that contain characters from different fonts.

- Arguments:*
- string* — The string to be typed on the window.
 - start* — The first character of the string to be printed. Recall that the Explorer system uses a zero-based index for strings.
 - end* — The last character to be printed. If *end* is `nil`, all of *string* from *start* to the last character of the string is typed on the window.

If neither *start* nor *end* is specified, the entire string is typed.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is `w:green`. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of `w:sheet-foreground-color` (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

For example, the following code types a string on the window in the color red:

```
(send my-window :line-out "This is a red string" 0 20 w:red)
```

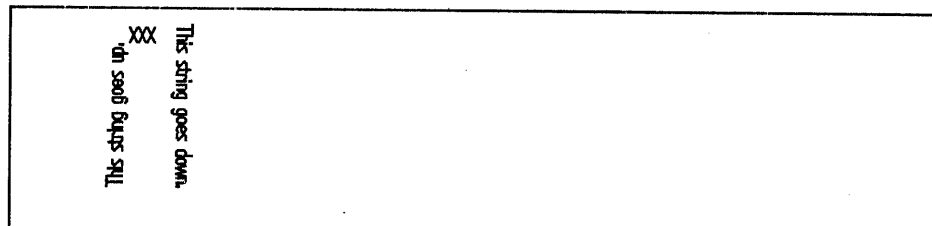
<code>:string-out-up</code>	<i>string</i> &optional (<i>start</i> 0) (<i>end</i> nil) <i>color</i>	Method of <i>windows</i>
<code>:string-out-down</code>	<i>string</i> &optional (<i>start</i> 0) (<i>end</i> nil) <i>color</i>	Method of <i>windows</i>

Prints a string that is rotated. `:string-out-up` prints *string* on the window by drawing the characters turned 90° counter-clockwise and moving up toward the top of the window. `:string-out-down` prints *string* on the window by drawing the characters turned 90° clockwise and moving down toward the bottom of the window. The arguments to these methods are the same as the arguments to the `:string-out` method.

These methods execute very slowly, but they are useful for labelling the y-axis of a graph.

For example, when the following code is evaluated in a Zmacs buffer, it produces an image similar to the following:

```
(send w:selected-window :set-cursorpos 200 500)
(send w:selected-window :string-out-up "This string goes up.")
(send w:selected-window :string-out " XXX ")
(send w:selected-window :string-out-down "This string goes down.")
```



w:draw-char <i>font char x y alu window-or-array</i> &optional <i>color</i>	Function
w:draw-char-up <i>font character</i> &optional (<i>x nil</i>) (<i>y nil</i>) (<i>alu nil</i>) (<i>window w:selected-window</i>) <i>color</i>	Function
w:draw-char-down <i>font character</i> &optional (<i>x nil</i>) (<i>y nil</i>) (<i>alu nil</i>) (<i>window w:selected-window</i>) <i>color</i>	Function

Draws *char* of *font* on *window-or-array* using *alu*. The upper left corner of *char* begins at (*x,y*). *x* and *y* are relative to *window*'s outside edges. *char* can have a width greater than 32.

The optional *color* argument specifies the value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is **w:green**. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

w:draw-char-up draws characters that advance up the video display; **w:draw-char-down** draws characters that advance down the video display. These functions are analogous to the **:string-out-up** and the **:string-out-down** methods.

:draw-char *font char x y* &optional *alu color* Method of **w:stream-mixin**

Draws *char* of *font* using *alu*. The upper left corner of *char* begins at (*x,y*). *x* and *y* are relative to *window*'s outside edges. *font* need not be a member of the window's font map. *alu* defaults to the value of the **w:char-aluf** instance variable for the window.

The optional *color* argument specifies the value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is **w:green**. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

:string-out-centered *string* &optional *left right top color* Method of *windows*

Outputs a string to the window and centers the string within specified parameters. The cursor is at the end of the string when this method completes execution. If the string contains multiple lines, the entire rectangular shape it occupies is centered as a unit. To center lines individually, you should output each line individually with this method.

Arguments: *string* — The string to be typed on the window.

left, right — The leftmost or rightmost position, in pixels, of the window that determines centering. *left* defaults to 0; *right* defaults to the inside width of the sheet.

top — The vertical position, in pixels, of the top of the output, relative to the window's top margin.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is **w:green**. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of **w:sheet-**

foreground-color (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

:fresh-line Method of *windows*

Moves the cursor position to the beginning of a blank line. **:fresh-line** does this in one of two ways:

- If the cursor is already at the beginning of a line (that is, at the inside left edge of the window), **:fresh-line** clears the line to make sure it is blank and leaves the cursor where it was.
- Otherwise, **:fresh-line** advances the cursor to the next line and clears the line just as if a **#\return** had been output.

The execution of this method is not affected by the **:cr-not-newline-flag** initialization option.

:display-lozenged-string *string* &optional *color* Method of *windows*

Outputs the value of *string* enclosed in a lozenge. This is how special characters are echoed, such as **<RETURN>** for the RETURN key.

Arguments: *string* — The string to be typed on the window.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is **w:green**. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

w:sheet-line-out *window string* &optional (*start* 0) (*end* nil) Function
set-xpos set-ypos dwidth color

Performs a line output function, handling font changes and line overflows. If any existing characters are present, **w:sheet-line-out** erases them from the line of the window. The font changes are stored in the font attribute of each character. Line overflows are handled if the right-margin-character flag is set for this window; otherwise, the line is truncated. **w:sheet-line-out** is used for line redisplay in the Zmacs editor.

Arguments: *window* — The window on which the text is displayed.

string — The string to be displayed. The *start* and *end* arguments can specify that only a part of *string* be displayed.

start — The first character of the string to be printed. Recall that the Explorer system uses a zero-based index for strings.

end — The last character to be printed. If *end* is nil, all of *string* from *start* to the last character of the string is typed on the window.

set-xpos, set-ypos — The x and y window coordinates, in pixels, where the first character of *string* is displayed. If either or both of these arguments are nil or are not specified, the appropriate coordinate of the current cursor position is used.

dwidth — Prevents inadvertently erasing the end of an existing line. For example, if the line being displayed is italic font, the last character may extend into the area where `w:sheet-line-out` displays *string*. Because `w:sheet-line-out` erases all existing characters between *start* and *end*, *dwidth* allows `w:sheet-line-out` to redisplay the last character you want of the existing line and then to display *string*. Specifying *dwidth* makes sense only if *start* is specified.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is `w:green`. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of `w:sheet-foreground-color` (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

Output Exceptions

7.4 Before sending output to a window, the window system checks for various exceptional conditions. If an exceptional condition is discovered, the window system invokes a standard method to handle it. Redefining or adding `:before` and `:after` methods to these methods can change the handling of exceptions. For example, output that puts the cursor too close to the right margin causes an end-of-line exception; the handling of this exception moves the cursor to the next line or truncates the line or whatever the window's flavor specifies. There are four exceptional conditions: end-of-line, end-of-page, more, and output-hold.

The exceptions are actually indicated by flag bits that are automatically set in the window. If the method handling the exception is invoked, it should do nothing unless the corresponding flag is set and should not return with the flag still set; otherwise, an error is signaled.

- The end-of-line, end-of-page, and more flags are set and cleared automatically by moving the cursor; the flags are set if and only if the cursor is in the right place for them. Thus, the exception handler need only make sure to move the cursor to a proper place.
- The output-hold flag is set when a window is deexposed. Usually, the output-hold exception handler simply waits for or brings about a situation in which the reason for the output-hold exception is no longer valid, typically because the window has been exposed.

Deexposed Timeout Actions

7.4.1 When a process attempts to type out on a window that is deexposed and that has its output hold flag set, what happens depends on the window's *deexposed timeout action*. The deexposed timeout action can be any of certain keyword symbols, or it can be a list. After the specified action is taken, if the output hold flag is still set, the process waits for it to clear. The action may affect the value of the output hold flag.

:deexposed-typeout-action *action* Initialization Option of *windows*
Gettable, settable. Default: :normal
w:sheet-deexposed-typeout-action *window* Macro

Initializes a window's deexposed typeout action to *action*. The macro returns the value. The deexposed typeout action can have any of the following values:

Value	Description
:normal	No action—the process waits for the output hold flag to clear.
:expose	Sends the window an :expose message. If the superior has a screen-array, this message may expose the window; if it does expose the window, the output hold flag is probably cleared, allowing typeout to proceed immediately. If the superior is the screen, :expose provides a very different user interface from :normal .
:permit	Permits typeout even though the window is not exposed, as long as the window has a screen array; that is, the window can type out on its own bit-save array even though it is not exposed. The next time the window is exposed, the updated contents are retrieved from the bit-save array. :permit turns off the output hold flag if the window has a screen array. This mode has the disadvantage that output can appear on the window without anything being visible to the user, who might never see what is going on and might miss something interesting. You can request that output in this mode to partially visible windows be transferred to the screen periodically. See the description of the w:screen-manage-update-permitted-windows variable in paragraph 5.9.3, Control of Partial Visibility.
:notify	Notifies the user when there is an attempt to do output on the window by sending a :notice message to the window with an :output argument. (The :notice message is described in paragraph 18.2, Notifications.) The default response to this message is to notify the user that the window wants to type out and to put the window on a list for TERM 0 S to select it. Telnet windows have :notify deexposed typeout action by default.
:error	Signals an error.
a list of the form (<i>operation arguments...</i>)	Sends the window a message with <i>operation</i> and <i>arguments</i> .

(Continued)
Value

Description

Functions such as `ed`, whose purpose is to select a window for the user, should not return immediately. If `ed` returns immediately, then when it is called in a Lisp Listener with its `deexposed` typeout action set to `:expose`, the system would immediately switch back to the Lisp Listener after printing the value returned by `ed`. Because this would not allow a user to see the printed value, this action defeats the purpose of `ed`. To avoid this behavior, `ed` calls `w:await-window-exposure`.

`w:await-window-exposure` Function

Waits to return until `*terminal-io*` is exposed (more precisely, until its `:await-exposure` operation returns).

`:await-exposure` Method of *windows*

Does not return until the window is exposed. (Some window flavors implement it differently.)

`w:sheet-force-access` (*window*) *body* Macro

Allows you to do typeout on a window that has a screen array even if its output hold flag is set. It works by turning off the output hold flag temporarily around the execution of *body*. This is useful for drawing on a window while it is not visible. For example, changing the menu items of a menu redraws the menu contents immediately even if the menu is not visible; this is because it looks better to the user for the menu to become visible in one instant with the correct contents.

If the window is exposed, `w:sheet-force-access` outputs to it. If the window is not exposed but has a bit-save array, the output goes into the bit-save array. If the window is not exposed and does not have a bit-save array, `w:sheet-force-access` does nothing; it returns *without* evaluating *body*.

For example, when a text scroll window is given a new item generator, the new generator completely changes the text that it should display. The window is redisplayed in its bit-save array if necessary.

```
(defmethod (w:text-scroll-window :set-item-generator)
  (new-item-generator)
  (setq item-generator new-item-generator)
  (w:sheet-force-access (self)
    (send self :clear-screen)
    (send self :redisplay 0
      (w:sheet-number-of-inside-lines))))
```

**Output-Hold and
End-of-Page
Exceptions**

7.4.2 Output-hold and end-of-page exceptions cause the window system to perform two actions. First, if the window's output-hold flag is set, an output-hold exception occurs. The window system invokes the `:output-hold-exception` method to handle it.

Next, if the end-of-page flag is set, the window system invokes the `:end-of-page-exception` method to handle an end-of-page exception. The end-of-page flag is usually set if the y position of the cursor is less than one line height above the inside bottom edge of the window.

- :handle-exceptions** Method of *windows*
 Performs all the exception processing described in the rest of this section. Exceptions are processed in the following order: `output-hold`, `end-of-page`, `more`, and `end-of-line`.
- :output-hold-exception** Method of *windows*
 Should not return until the `output-hold` is cleared. **:output-hold-exception** may wait for the `output-hold` flag to be cleared or try to cause it to be cleared. The window's `deexposed` timeout action determines the default action for this method. For example, if the window's `deexposed` timeout action is `:notify`, a window pops up to let the user know that the original window is trying to send output.
- w:sheet-output-hold-flag &optional *window*** Macro
 Returns the `output-hold` flag of *window*, which is 1 if there is a hold and 0 if there is no hold. This macro can be used with `setf`.
- :end-of-page-exception** Method of *windows*
 Handles the `end-of-page` exception when present. It does nothing if invoked when the flag is 0.

 The default definition is simply to move the cursor to the top line, clear that line, and set the vertical position for the next ****MORE**** if ****MORE**** processing is enabled.
- w:sheet-end-of-page-flag &optional *window*** Macro
 Returns the `end-of-page` flag of *window*, which is 1 if the next output method should wrap and 0 otherwise. This macro can be used with `setf`.

****MORE****
Exceptions

7.4.3 If the window's more flag is set, a ****MORE**** exception happens. The more flag is set when the cursor is moved to a new line; for example, when a `#\return` is typed, the cursor is positioned below the *more vpos* of the window. If `w:more-processing-global-enable` is `nil`, this exception is suppressed and the more flag is turned off. The `:more-exception` method is invoked to handle the exception.

The more flag is set only when the cursor moves to the next line because a `#\return` is typed out, after a `:line-out`, or by the `:end-of-line-exception` method. The more flag is not set when the cursor position of the window is explicitly set—for example, with `:set-cursorpos`. In fact, explicitly setting the cursor position clears the more flag. When `timeout` is being output sequentially to the window, ****MORE**** exceptions happen at the proper times to pause while the user reads the text that is being typed. When `cursor-positioning` is being used, however, the system cannot guess what order the user is reading text in and when (if ever) is the proper time to stop. In this case, it is up to the application program to provide any necessary pauses.

The algorithm that sets the vertical position in the next ****MORE**** exception never overwrites something before you have had a chance to read it, and the algorithm tries to execute a ****MORE**** only if a large amount of output occurs. However, if output starts near the bottom of the window, there is no way to tell how much output will occur. If little output is processed, you do not want to be bothered by a ****MORE****, so the algorithm does not execute one immediately. This postponement may make it necessary to cause a ****MORE**** break somewhere other than at the bottom of the window. As

additional output happens, the position of successive ****MORE**** exceptions migrates and eventually ends up at the bottom of the window.

- w:sheet-more-flag** &optional *window* Macro
Returns the more flag, which is 1 if the next output method should execute a ****MORE****, and 0 otherwise. This macro can be set using **setf**.
- :more-vpos** Method of *windows*
w:sheet-more-vpos *window* Macro
Returns the vertical position at which the next ****MORE**** should happen during output to the window. The macro accesses the instance variable.
- w:more-processing-global-enable** Variable
Default: **nil**
Determines whether ****MORE**** processing occurs. ****MORE**** processing does not happen if this variable is **nil** during the output method in which the ****MORE**** would have happened.
- :more-exception** Method of *windows*
Invokes a **:clear-eol** method, types out ****MORE****, reads a character using the **:tyi** method, restores the cursor position to where it was originally when the **:more-exception** was detected, executes another **:clear-eol** to wipe out the ****MORE****, and resets the **w:more-vpos** instance variable. The character read in is ignored.

:more-exception works by calling **w:sheet-more-handler** if the more flag is set. **:more-exception** should do nothing if the flag is set to 0. If you want to customize **:more-exception**, you should redefine **:more-exception** to call the **w:sheet-more-handler** function with different arguments and to do other things as well. You should not write a new definition from scratch, because of the complexity of the **w:sheet-more-handler** function.
- w:sheet-more-handler** &optional (*method read-char*) Function
(*more-string w:*unidirectional-more-standard-message**)
Implements the standard handling of ****MORE**** exceptions.

Arguments: *method* — The method to use when reading input.
more-string — The output to be printed and then erased.
- w:*unidirectional-more-standard-message*** Variable
Default: *****MORE*****
A default string displayed to indicate that more information can be displayed than is currently visible. The user can request the next window of text by pressing the space bar. Once text has scrolled off the display, however, the user cannot look at that text again (hence the name unidirectional). For example, typeout windows are unidirectional.
- w:*bidirectional-more-standard-message*** Variable
Default: **---MORE---**
A default string displayed to indicate that more information can be displayed than is currently visible. The user can move towards the beginning or end of the text by using the CTRL-V and META-V keystroke sequences, respectively. For example, the buffers invoked by the Zmacs View File command are bidirectional.

w:autoexposing-more-mixin

Flavor

If a window includes the **w:autoexposing-more-mixin** flavor, the window is exposed when a **:more-exception** method executes, meaning that an **:expose** message is sent to it. This flavor is intended to be used with a deexposed timeout action of **:permit** so that a process can type out on a deexposed window and then have the window expose itself when a ****MORE**** break happens.

End-of-Line Exceptions

7.4.4 If the cursor is at or near the end of the line so that there is no room to output the next character, an end-of-line exception happens. The **:end-of-line-exception** method is invoked to handle it. A flag is not used to trigger this exception because the condition depends on the width of the character output.

The way the cursor position advances to the next line when it reaches the right edge of the window is called *horizontal wraparound* or *continuation*. If you prefer, you can make windows that truncate lines instead of wrapping around by using **w:line-truncating-mixin**.

:end-of-line-exceptionMethod of *windows*

Defined by default to advance the cursor to the next line, just as typing a **#\return** character does normally. Using this method may, in turn, cause an end-of-page exception or a more exception to happen. Furthermore, if **w:right-margin-character-flag** is on, then before going to the next line, an exclamation point in the current font is typed at the cursor position. When this flag is on, end-of-line exceptions occur earlier to make room for the exclamation point.

:tyo-right-margin-character *xpos ypos ch*Method of *windows*

Prints a right margin character when the right margin character is to be printed. This is the facility that prints a **!** at the right margin in Zmacs when text extends over more than one line.

Arguments: *xpos* — Is ignored; the right margin is used instead.

ypos — The y pixel position where the margin is to be drawn.

ch — The character to be drawn as the right margin character.

w:line-truncating-mixin

Flavor

Required flavor: **w:stream-mixin**

Gives a window the ability to truncate lines at the right margin instead of continuing output onto the next line as usual. Truncation is performed if the window's truncate-line-out flag is set. When the cursor position is near the right-hand edge of the window and there is an attempt to type out a character, the character simply is not typed out.

:truncate-line-out-flag *flag*Initialization Option of **w:line-truncating-mixin****w:sheet-truncate-line-out-flag** &optional *window*

Macro

Initializes the truncate-line-out flag of the window to the value of *flag*: 1 causes truncation and 0 does not. The macro returns or sets this flag for *window*.

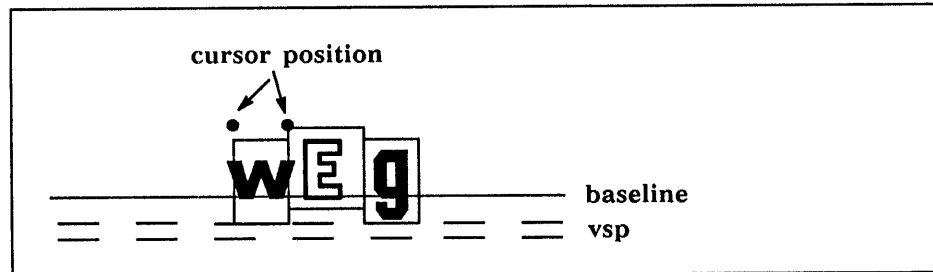
w:truncating-window

Flavor

Built on the **w:window** flavor with **w:line-truncating-mixin** mixed in. A window instantiated of this flavor is like regular windows of flavor **w>window** except that lines are truncated instead of wrapping around (that is, the **truncate-line-out** flag is set to 1).

Cursor Motion

7.5 The window's cursor position is where the upper left corner of the next output character appears, with a vertical offset if necessary to match up the baselines of various fonts.

**Cursor Position
for Stream
Operations**

7.5.1 Recall that cursor position arguments and values of stream methods are relative to the inside upper left corner of the window. Using the following methods, the end-of-page, more, and end-of-line exception flags are set if the cursor is moved to a position where the flags should be on, and the flags can be cleared if they were previously on and the cursor is moved to a place where the flags should be off.

:read-cursorpos &optional (*units* :pixel)

Method of *windows*

Returns two values: the x and y coordinates of the cursor position. These coordinates are expressed in pixels by default, but if *units* is **:character**, the coordinates are given in character widths and line heights. Character widths are of little significance when you are using variable-width fonts.

:increment-cursorpos *x y* &optional (*units* :pixel)

Method of *windows*

Advances the cursor position the specified amount for each coordinate. This method moves the cursor through a variable amount of space, rather than instantaneously jumping it to another position. This means that exceptions can happen, just as if output were being processed. Thus, the cursor wraps around at the margins or does whatever this window does when **:end-of-line-exception**, **:end-of-page-exception**, or ****MORE**** processing happens at the appropriate place.

Arguments: *x, y* — The number of units, specified by the *units* argument, that the cursor position is adjusted.

units — A unit of measure. The default is **:pixel**, but **:character** can also be used. **:character** implies the character height and width, in pixels, of the current font. Character widths are of little significance when you are using variable-width fonts.

:set-cursorpos *x y* &optional (*units* :pixel) Method of *windows*
w:sheet-set-cursorpos *window x y* Function

Moves the cursor position to the specified coordinates immediately, without moving through the intervening space. If the coordinates are outside the window, this method moves the cursor position to the place within the window nearest the specified coordinates. The arguments for this method are identical to the arguments for **:increment-cursorpos**.

:home-cursor Method of *windows*

Moves the cursor to the upper left corner of the window.

:home-down Method of *windows*

Moves the cursor to the lower left corner of the window.

:forward-char &optional *char* Method of *windows*

:backward-char &optional *char* Method of *windows*

Moves the cursor forward or backward, respectively, one character position, or, if *char* is specified, the width of *char* in the current font. Exceptions are processed. **:forward-char** is like outputting a space that has the appropriate width. With **:backward-char**, however, there is no reverse wraparound—if the cursor is at the left margin, the cursor does not move.

:size-in-characters Method of *windows*

Returns two values: the dimensions of the window in units of character widths and in line heights.

:set-size-in-characters *width-spec height-spec* &optional *option* Method of *windows*

Sets the inside size of the window to the specified width and height, without changing the position of the upper left corner.

Arguments: *width-spec* — Either a number of characters or a character string. The inside width of the window is made wide enough to display those characters in the current font.

height-spec — Either a number of lines or a character string containing a certain number of lines separated by carriage returns. The inside height of the window is made high enough to display those lines in the current font.

option — Passed along to the **:set-edges** method. (This method is described in paragraph 4.3.2.)

NOTE: Character widths are of little significance when you are using variable-width fonts.

Cursor Position Relative to Outside Coordinates 7.5.2

<code>w:cursor-x</code>	Instance Variable of <i>windows</i>
<code>w:sheet-cursor-x</code>	Defsubst
<code>w:cursor-y</code>	Instance Variable of <i>windows</i>
<code>w:sheet-cursor-y</code>	Defsubst

Contains the x or y coordinate of the window's current cursor position. Note that these are outside coordinates. Unlike the arguments to stream methods, the coordinates refer to the outside upper left corner of the superior at the highest level in the window hierarchy containing this window. The instance variables are the number of pixels from coordinate 0,0 of the screen; see paragraph 5.7.1, Concepts of Screen Arrays, for an explanation of offsets.

The defsubsts access the respective instance variables. You can set these by using `setf`.

Erasing

7.6 Erasing methods and macros operate on window pixels by drawing the area to be erased using the window's `w:erase-aluf` instance variable as the ALU argument. This is by default `w:alu-back`, which clears the screen bits of the screen area drawn. (See paragraph 12.2, ALU Arguments, for more details about `w:erase-aluf` and `w:alu-back`. Also see `w:erase` in Table 12-2, ALU Values for Graphics Methods.)

<code>:clear-char</code>	<i>&optional char</i>	Method of <i>windows</i>
<code>w:sheet-clear-char</code>	<i>window &optional char</i>	Macro

Erases the character at the current cursor position for *window*. *char* determines the width of the character being erased. When variable-width fonts are used, the character code of the character being erased specifies how wide the character is. The default is the current font. If you do not specify *char*, the `:clear-char` method simply erases a character width of the current font, which is fine for fixed-width fonts. You should specify *char* when you are working with variable-width fonts.

<code>:clear-string</code>	<i>string &optional start end</i>	Method of <i>windows</i>
----------------------------	---------------------------------------	--------------------------

Erases enough space, starting at the cursor, to contain a string or a portion of a string printed in the current font. Using a fixed-width font is equivalent to executing `:clear-char` once for each character in the string. The `:clear-string` method is necessary because of variable-width fonts.

If *string* contains return characters, then each part of *string* clears characters from successive lines of the screen. That is, the first part of *string*, from the beginning to the first return, clears space on the first line of the screen equal to the length of the first part of *string*. The next part of *string*, from the first return to the next return, clears space on the next line of the screen equal to the length of that part of *string*, and so on. For example, consider the following code:

```
(progn ()
  (send w:selected-window :clear-string
    "ABCDEFGH
12345
987654321")
  (send w:selected-window :set-cursorpos 500 500))
```

Suppose that the screen contents are as follows:

```
Text that originally covers the screen
so that you can see the effect of
the :clear-string method.
```

Then, when the code is executed, the screen appears as follows:

```
 t originally covers the screen
at you can see the effect of
r-string method.
```

- Arguments:**
- string* — How much space on the window is to be erased. The entire height of the line is erased, so it does not matter whether the text on the screen is *string* or something else; *string* determines how far horizontally to erase.
 - start* — The index position in the string where erasing is to begin. Recall that the Explorer system uses a zero-based index for strings.
 - end* — The index position of the string where erasing is to end.

:clear-eol Method of *windows*
w:sheet-clear-eol *window* Function

Erases from the current cursor position to the end of the current line. That is, **:clear-eol** erases a rectangle horizontally from the cursor position to the inside right edge of the window, and vertically from the cursor position to one line height below the cursor position.

:clear-eof Method of *windows*
w:sheet-clear-eof *window* Function

Erases from the current cursor position to the bottom of the window. In more detail, **:clear-eof** first executes a **:clear-eol**, and then clears all of the window beyond the current line.

:clear-screen Method of *windows*
w:sheet-clear *window* *&optional (margins-p nil)* Function

Erases the whole window and moves the cursor position to the upper left corner of the window. For the **w:sheet-clear** function, if *margins-p* is **t**, the function erases the margins of *window* as well as the contents of *window*.

:clear-between-cursorposes *start-x start-y end-x end-y* Method of *windows*

Erases an area starting at cursor position *start-x* and *start-y*, wrapping around, if necessary, at the end of the line or the page, and ending at *end-x* and *end-y*.

Though the arguments are expressed as cursor positions relative to *window*'s inside coordinates, the cursor position of the window is not changed.

Inserting and Deleting Characters and Lines

7.7 Inserting a character means printing it at the current cursor position and pushing the rest of the text on the line toward the right margin. Similarly, deleting a character means pulling the following text on the line back toward the left so that the position occupied by the character is closed up. Inserting or deleting lines works the same way vertically, moving the lines below the cursor down or up.

Some methods use a numeric argument to specify the amount of space to insert or delete. These methods also use an argument to specify the unit in which the space is measured—either `:pixel` or `:character`. The unit can be the same as in the `:read-cursorpos` method, but the default is `:character` rather than `:pixel`.

`:insert-char` *&optional* (*n* 1) (*unit* `:character`) Method of *windows*

Opens up a space in the line the cursor is currently on. This space is equal to the width of the specified number of characters or pixels and is created by shifting the characters to the right of the cursor farther to the right to make room. Characters pushed past the right edge of the window are lost.

Arguments: *n* — The number of character widths or pixels to open up in the line.

unit — Whether `:insert-char` opens up character widths or pixel widths in the line. If *unit* is `:pixel`, `:insert-char` opens up the specified number of pixels. If *unit* is `:character`, `:insert-char` opens up the specified number of character widths. In this case, `:insert-char` assumes that all characters are one character width wide and thus does nothing useful with variable-width fonts.

`:insert-string` *string* *&optional* (*start* 0) (*end* nil) Method of *windows*
(*type-too* t) *color*

Inserts a string at the current cursor position by moving the rest of the line to the right to make room for it. Characters pushed past the right edge of the window are lost.

Arguments: *string* — The string to insert. This argument can also be a number, in which case the character with that code is inserted.

start — The index position in the string of the first character to be inserted. Recall that the Explorer system uses a zero-based index for strings.

end — The index position in the string of the last character to be inserted.

type-too — Whether *string* is actually typed in the space opened up. If *type-too* is `nil`, *string* is not actually printed; the space opened up is big enough for the string but is left blank.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is `w:green`. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of `w:sheet-foreground-color` (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

:insert-line &optional (*n* 1) (*unit* :character) Method of *windows*

Moves the line the cursor is currently on, as well as all the lines below it, down far enough to insert the specified number of lines. The line containing the cursor is moved in its entirety, not broken, no matter where the cursor is on the line. A blank line is created at the cursor position. Lines pushed off the bottom of the window are lost.

Arguments: *n* -- The number of lines or pixels of space to open up.

unit -- Whether **:insert-line** opens up a specified number of lines (for **:character**) or pixels (for **:pixel**).

:delete-char &optional (*n* 1) (*unit* :character) Method of *windows*

Deletes the character at the current cursor position and moves any remaining text on the line to the left to close up the empty space.

Arguments: *n* -- The number of characters or pixels to delete, starting at the current cursor position and deleting the specified number to the right.

unit -- Whether to delete characters or pixels. If *unit* is **:character**, **:delete-char** deletes characters, and the method assumes that all characters are one character width wide. Thus, **:delete-char** does not do anything useful with variable-width fonts.

:delete-string *string* &optional (*start* 0) (*end* nil) Method of *windows*

Uses *string* to determine the region to be removed from its window. **:delete-string** should be used to delete strings containing variable-width fonts.

If *string* is a string (or a substring specified by *start* and *end*) **:delete-string** removes a region exactly as wide as that (sub)string beginning at the current cursor position, and closes the gap by moving to the left the part of the current line that is to the right of the removed region. If *string* is a number, it is considered to be a character code. The single character is treated like a string containing that character.

:delete-line &optional (*n* 1) (*unit* :character) Method of *windows*

Deletes the line at the current cursor position and closes the empty space by moving up any remaining text below the current line.

Arguments: *n* -- The number of lines or pixels to delete, starting at the current cursor position and deleting the specified number.

unit -- Whether to delete characters or pixels. If *unit* is **:pixel**, **:delete-line** deletes rows of pixels; if *unit* is **:character**, **:delete-line** deletes lines. If *unit* is **:character**, this argument assumes that all characters on the line are one character height high.

Anticipating the Effect of Output

7.8 The following methods do not produce output but provide information about what would happen to the cursor and the screen if output were produced.

:character-width *char* &optional (*font* **w:current-font**) Method of *windows*

Returns the width in pixels of the specified character.

Arguments: *char* — The character whose width is to be returned. If *char* is `#\tab`, the value returned depends on the current position of the cursor. If *char* is `#\return`, the value returned is 0. If *char* is `#\backspace`, the value returned is a negative number. For other cases, the actual width of the character is returned.

font — The font of the character. If *font* is not specified, the current font is used.

:compute-motion *string* &optional (*start* 0) (*end* nil) Method of *windows*
 (*x w:cursor-x*) (*y w:cursor-y*) (*cr-at-end-p* nil) (*stop-x* 0) *stop-y*
bottom-limit right-limit font line-height tab-width

w:sheet-compute-motion *window x y string* &optional (*start* 0) (*end* nil) Function
 (*cr-at-end-p* nil) (*stop-x* 0) (*stop-y* nil)
bottom-limit right-limit font line-height tab-width

Determines where the cursor would be if a string were displayed on the window using the `:string-out` method. `:compute-motion` returns the following four values:

1. *final-x* — The x position where output is computed to stop.
2. *final-y* — The y position where output is computed to stop.
3. *final-index* — One of three values:
 - The index in *string* where output would stop, which is useful for inserting strings at *start-x* and *end-x* positions
 - nil if the output would reach the end of the string
 - t if the string itself was processed but not the implicit return that was supposed to follow the string
4. *maximum-x* — The largest x position value reached during processing.

All coordinates for this method are cursor positions, relative to the window's inside edges. However, if you specify all the arguments, you can use the origin of a different coordinate system as long as you interpret the values in the same coordinate system.

Arguments: *string* — The string that would be typed on the window. This can be either a normal string or an array that contains elements of type `:character`.

start — The first character of the string. If *start* is nil, `:compute-motion` starts with the first character in the string. Recall that the Explorer system uses a zero-based index for strings.

end — The last character of the string. If *end* is nil, all of *string* from *start* to the last character of the string is used.

x, y — The starting x and y coordinates of the position where *string* would be typed. If either of these is not specified, the appropriate part of the current cursor position is used.

cr-at-end-p — Whether to perform the computations for a `:string-out` method (if *cr-at-end-p* is nil) or for a `:line-out` method (if *cr-at-end-p* is t). A `:string-out` method does not output a RETURN (a carriage return, or cr) at the end of the string; a `:line-out` method does.

stop-x, stop-y — Define the size of an imaginary window in which the string would be printed. `:compute-motion` stops if the cursor position becomes

simultaneously greater than or equal to both of these arguments. These arguments default to the lower left corner of the window. This corner is reached before the right one, because output goes from left to right on each line.

bottom-limit — The vertical position where wraparound begins. This argument defaults to the inside height of the window. *bottom-limit* differs from *stop-y* in that it causes **:compute-motion** to assume that text wraps around rather than stopping when the bottom of the window is reached.

right-limit — The farthest position to the right where text is typed. This argument defaults to the inside width of the window. *right-limit* differs from *stop-x* in that it causes **:compute-motion** to assume that text wraps around rather than stopping when the farthest position to the right in the window is reached.

font — The font to use in the computation. The computation normally uses *font*, or the current window's font if *font* is **nil**. If *string* is an array that contains elements of type **:character**, each character's font attribute is used as an index in the window's font map to find the font for that character, and *font* is ignored except possibly for defaulting the *tab-width*.

line-height — The vertical spacing to be used in the computation. The default for *line-height* is the line height of *font* if *font* is non-**nil**; otherwise, the default for *line-height* is the window's line height.

tab-width — The number of pixels between tab stops. If *tab-width* is omitted, **:compute-motion** defaults to one of the following:

- If no font is specified — (w:sheet-tab-width self)
- If a font is specified — (w:sheet-tab-nchars self) multiplied by (w:font-char-width font)

:string-length *string* &optional (*start* 0) (*end* **nil**) *stop-x* Method of *windows*
font (*start-x* 0) *tab-width*

Determines how far the cursor would move if the string were to be displayed. The **:string-length** method is very much like **:compute-motion**, but works in only one dimension.

:string-length returns three values:

1. *final-x* — Where the imaginary cursor ends up.
2. *final-index* — The index of the next character in the string—the length of the string if the whole string were processed—or the index of the character that would have moved the cursor past *stop-x*.
3. *maximum-x* — The maximum x coordinate reached by the cursor. This is the same as the first value unless there are **#\return** or **#\backspace** characters in the string.

See the **:compute-motion** method, described previously, for details about the arguments.

**Explicit
(Noncursor)
Output**

7.9 A window has certain information about its state that can change as output is produced. This information includes the cursor position, the current font, alu argument, and exception flags. The presence of this information makes the window behave coherently as a stream so that the output from one method follows that of the previous method. Sometimes this behavior is not desirable. The explicit output methods use a window only for its position and size, with all additional information passed by the caller explicitly. In this way, multiple streams of output to the same window can exist without interfering with each other by trying to use a single cursor.

The *x* and *y* position arguments used by the following methods are relative to the outside edges of the window. This is different from the stream and higher-level methods, because these explicit output methods are frequently used for drawing parts of the margins, such as labels and margin regions.

:string-out-explicit *string start-x start-y x-limit y-limit font* Method of *windows*
alu &optional (start 0) end multi-line-line-height color

Prints a string or a portion of a string on the window without using or moving the cursor position.

:string-out-explicit returns three values: the final *x* position, the final *y* position, and the final index in the string. You can use these values to execute multiple methods in consecutive places on the screen.

Arguments: *string* — The string to be typed on the window.

start-x, start-y — The position where the first character of *string* is to be typed.

x-limit, y-limit — The window coordinates where output of *string* stops. If *x-limit* or *y-limit* is non-nil, output stops if it reaches that position before typing out the entire string.

font — The font to use. There is no default; you must specify a font.

alu — Used when typing *string* on the window. The window's current ALU argument is not used or altered.

start — The index in *string* where output begins. The default displays the first character. Recall that the Explorer system uses a zero-based index for strings.

end — The index in *string* where output stops. If *end* is nil, all of *string* from *start* to the last character of the string is typed on the window.

multi-line-line-height — Line output height. If *multi-line-line-height* is a number, then the window's line height is ignored and the horizontal output position moves to *start-x* rather than the left margin for the next line of output. If *#\return* constructs are in the output and *multi-line-line-height* is nil, they are printed as **<RETURN>**.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is **w:green**. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

:string-out-centered-explicit *string* &optional *left y-pos right* Method of *windows*
y-limit font alu (start 0) end multi-line-line-height color

Outputs a string or a portion of a string, and centers the string horizontally.

Arguments: *string* — The *string* is the string to be output.

left — The left bounding window coordinate within which *string* is to be centered. This argument defaults to the inside left edge of the window.

y-pos — The vertical position where *string* is to be centered. This argument defaults to the inside top edge of the window.

right — The right bounding window coordinate within which *string* is to be centered. This argument defaults to the inside right edge of the window.

y-limit — The vertical limit where output is to stop if all of *string* has not been printed. The argument defaults to the inside bottom edge of the window.

font — The font to be used. This argument defaults to the window's current font.

alu — The ALU argument to be used. This argument defaults to the window's current ALU argument.

start — The index in *string* where output begins. The default displays the first character. Recall that the Explorer system uses a zero-based index for strings.

end — The index in *string* where output stops. If *end* is nil, all of *string* from *start* to the last character of the string is typed on the window.

multi-line-line-height — Line output height. If *multi-line-line-height* is a number, then the window's line height is ignored.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is *w:green*. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of *w:sheet-foreground-color* (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

:string-out-x-y-centered-explicit *string* &optional *left top right* Method of *windows*
bottom font alu start end multi-line-line-height color

Outputs a string centered both horizontally and vertically. It is horizontally centered between *left* and *right*, and vertically between *top* and *bottom*.

Arguments: *string* — The *string* is the string to be output.

left, top, bottom, right — The bounding window coordinates within which *string* is to be centered. These arguments default to the inside edges of the window.

font — The font to be used. This argument defaults to the window's current font.

alu — The ALU argument to be used. This argument defaults to the window's current ALU argument.

start — The index in *string* where output begins. Recall that the Explorer system uses a zero-based index for strings.

end — The index in *string* where output stops.

multi-line-line-height — Line output height. If *multi-line-line-height* is a number, then the window's line height is ignored.

color — The value or name of a color in the color map. The value can range from range 0 to 255; an example of a name is `w:green`. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table. The default color is the value of `w:sheet-foreground-color` (the foreground color of the window to which you are drawing). Note that in a monochrome environment the *color* argument is ignored; therefore, your application can be used on both color and monochrome systems.

NOTE: Many of the `:string-out` methods also have an associated function who name begins with `w:sheet`.

Window Parameters Affecting Text Output

7.10 The following methods and initialization options initialize, get, and set various window attributes that are relevant to typing out characters. See also the methods that manipulate the current font in paragraph 9.4, Flavors and Methods, and the functions that manipulate pixels in paragraph 5.5, Pixels.

`:more-p` *t-or-nil*

Initialization Option of *windows*

Gettable, settable. Default: `t`

Determines whether the window has ****MORE**** processing. `t` enables ****MORE**** processing; `nil` disables it. (See paragraph 7.4.3, ****MORE**** Exceptions.)

`:vsp` *n-pixels*

Initialization Option of *windows*

Gettable, settable. Default: `2`.

Sets the amount of vertical spacing, in pixels, between lines of text for this window. The argument is the number of pixels.

`:right-margin-character-flag` *x*

Initialization Option of *windows*

Default: `0`.

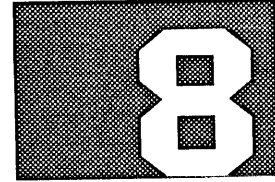
`w:sheet-right-margin-character-flag` &optional (*window self*)

Macro

Initializes the right-margin-character flag, which can be either 0 or 1. If *x* is 1, the window should print an exclamation point in the right margin when an end-of-line exception happens; if *x* is 0, it should not. The default is 0. (See paragraph 7.4.4, End-of-Line Exceptions for more details about the exception.)

This macro allows the value of the right-margin-character flag to be changed using the `self` function. When used without an argument, `w:sheet-right-margin-character-flag` refers directly to the associated instance variable and therefore must be called from methods or functions that use `(declare (:self-flavor ...))`.

- :backspace-not-overprinting-flag *x*** Initialization Option of *windows*
 Default: 0.
- w:sheet-backspace-not-overprinting-flag &optional (*window self*)** Macro
 Initializes the window's backspace-not-overprinting flag, which can be either 0 or 1. If *x* is 0, output of `#\backspace` moves the cursor position backward; if it is 1, it displays the overstrike character in a lozenge. `#\backspace`, which is the synonym for the `#\overstrike` character, is exactly like other special characters. The default is 0. See paragraph 7.2, How a Character Is Displayed, for a description of how special characters are handled.
- This macro allows the value of the right-margin-character flag to be changed using the `setf` function. When used without an argument, `w:sheet-backspace-not-overprinting-flag` refers directly to the associated instance variable and therefore must be called from methods or functions that use `(declare (:self-flavor ...))`.
- :cr-not-newline-flag *x*** Initialization Option of *windows*
 Default: 0.
- w:sheet-cr-not-newline-flag &optional (*window self*)** Macro
 Initializes the window's cr-not-newline flag, which can be either 0 or 1. If *x* is 0, output of `#\return` moves the cursor position to the beginning of the next line and clears that line; if *x* is 1, the output of a `#\return` character displays the word `return` in a lozenge. That is, `#\return` is treated exactly like other special characters. The default is 0. This flag affects neither the behavior of the `:line-out` nor the `:fresh-line` methods.
- This macro allows the value of the right-margin-character flag to be changed using the `setf` function. When used without an argument, `w:sheet-cr-not-newline-flag` refers directly to the associated instance variable and therefore must be called from methods or functions that use `(declare (:self-flavor ...))`.
- :tab-nchars *n*** Initialization Option of *windows*
 Default: 8.
- w:sheet-tab-nchars &optional (*window self*)** Macro
 Initializes the window's tab size. The *n* argument specifies the separation of tab stops on this window, expressed in units of the window's character width. This initialization option controls how the `#\tab` construct prints.
- This macro allows the value of the right-margin-character flag to be changed using the `setf` function. When used without an argument, `w:sheet-tab-nchars` refers directly to the associated instance variable and therefore must be called from methods or functions that use `(declare (:self-flavor ...))`.
- w:sheet-tab-width &optional (*window self*)** Macro
 Default: 64.
- Returns the distance between tab stops, measured in pixels. To alter the tab width, you should use `w:sheet-tab-nchars` instead of this macro.



Introduction

8.1 Windows can function as input streams. (See the discussion of streams in the *Explorer Input/Output Reference* manual.) The `w:stream-mixin` flavor, a component of `w>window`, provides this ability. Input characters normally come from the terminal keyboard but can also come from clicking the mouse buttons or from programs.

Window selection controls which process reads keyboard input because all keyboard input passes through the selected window. In fact, the concept of selection exists to control which process receives input by selecting the appropriate window.

Reading characters from a window normally returns a character object that represents a character in the Explorer character set. Character constants in code are written with the `#\` or `#/` construct and are described in the *Explorer Lisp Reference* manual.

Programs decode character objects with the Common Lisp character functions. For example, the `char-code` function can be used to return the character code of a character object. Other functions can be used to extract other components, such as:

```
(char-mouse-button #\mouse-m-2) => 1
(char-mouse-clicks #\mouse-m-2) => 1
(char-bit #\mouse-m-2 :mouse) => t
(char-font #3\A) => 3
(char-bits #\control-A) => 1
```

Characters behave just like fixnums in arithmetic operations and are similar (`=`) but not identical (`eq`) to fixnums. `:tyi` and related stream methods continue to return fixnums, while new methods and functions are defined that return character objects instead. When you extract components of a character that is actually a fixnum, you should convert the fixnum to a character object before invoking the extraction function. For example,

```
(char-code (int-char 65)) => 65
```

where `(int-char 65)` converts the fixnum 65 to a character object, which is then passed to the `char-code` function.

The keyboard hardware actually sends codes to the Explorer whenever a key is pressed or lifted; thus, the Explorer system knows at all times which keys are pressed and which are not. You can use the `w:key-state` function to check whether a key is down or up. You can also program a window to read the raw hardware codes, exactly as they are sent, by putting a non-nil value of the `:raw` property on the property list of the input buffer; however, the format of the raw codes is dependent on the hardware implementation, and it is not documented here.

Input Buffers

8.2 Input buffers are examples of input/output (I/O) buffers, a general facility provided by the window system. Every window generating input or from which input is read must have an input buffer that holds characters typed by the user before any program reads the characters. When you type a character, the character enters the selected window's input buffer. I/O buffers are explained in paragraph 8.6. Reading input from a window with the `read-char` function removes objects from the window's input buffer. The `w:stream-mixin` flavor gives the window an input buffer, but other flavors, such as command menus, provide an input buffer without `w:stream-mixin`. The window's `w:io-buffer` instance variable points to the input buffer.

You can explicitly manipulate input buffers in order to perform certain advanced functions by using the `:io-buffer` initialization option and the `:io-buffer` and `:set-io-buffer` methods. Also, you can put properties on the I/O buffer's property list; this capability lets you use various special features.

A window can be thought of as generating input when the keyboard is used while the window is selected. This is how ordinary characters enter the input buffer. The `:force-kbd-input` method can generate input at any place in the program. For example, mouse clicks are often handled by forcing input, which the window's command-interpreting process reads. Thus, the mouse click can also be said to generate input.

All the input, no matter how it is generated, goes to the same input buffer in chronological order. All the input methods remove input from the buffer in chronological order, that is, in a first-in, first-out (FIFO) order.

All windows can share a single input buffer; thus, all input generated by all windows goes into one buffer, and the input can be read through any of the windows. All keyboard input directed at a window and all mouse clicks on the windows are merged into a single chronological input stream. The program simply reads input from one of the windows (always the same one if the programmer prefers) and receives all the input intended for the program.

The input buffer does not record which window was responsible for generating input to be read from a shared input buffer. For mouse clicks, the program may need to know which window was selected with the mouse in order to respond to the command properly. The standard way to pass this information is to use a list as the input character and make the window selected by the mouse one of the elements of the list. Such a list is called a *blip*.

Windows that use input methods must use `w:stream-mixin`. The other windows need to use a function such as `w:io-buffer-put` to put their input into the proper input buffer. It is often easiest to use `w:stream-mixin` to create buffers and to generate the input with `:force-kbd-input`. However, for windows that support the `:io-buffer` method, returning the correct shared input buffer and putting the generated input into that buffer is sufficient. This procedure works for input generated by the `w:io-buffer-put` function and for input sent to other windows by the `:force-kbd-input` method, if these other windows use `w:stream-mixin`.

If a frame includes a pane that is handled by its own process (such as a Zmacs frame), this pane should not share the input buffer used by the rest of the panes. In general, there should be one input buffer for each process you are using, and this input buffer should be shared by the windows used by that process.

Normally, to make windows share an input buffer, you should create the buffer using `w:make-default-io-buffer` and specify the input buffer for the `:io-buffer` initialization option when each pane is created. Also, certain frame flavors automatically make the panes share an input buffer. See paragraph 15.2, Constraint Frames, for details.

w:stream-mixin

Flavor

Defines the standard input stream methods for entering input from the keyboard, as well as some nonstandard input methods. The nonstandard `:bitblt` methods are discussed in paragraph 12.4.2, Bit Block Transferring.

w:kbd-last-activity-time

Variable

The value returned by the `time` function when the last input character was typed, or a mouse button was pressed, whichever is later.

:io-buffer specInitialization Option of `w:stream-mixin`

Gettable, settable.

Initializes the `w:io-buffer` instance variable to be the input buffer of the window. The *spec* argument can be an I/O buffer, a number, or a list. If *spec* is a number, an I/O buffer is made with the size specified by *spec*, no input function, and the default output function. If *spec* is a list, *spec* is interpreted as *(size input-function output-function)*. If *output-function* is nil or omitted, `w:kbd-default-output-function` is used.

Blips

8.3 Input does not have to be in the form of characters; lists are often used as well for program-generated input, especially for representing mouse clicks in different kinds of mouse-sensitive areas. Such lists are called *blips*. The `car` of the list is, by convention, a symbol that identifies the kind of blip. Various methods create blips, including the `:command-menu` method of `w:menu` and the `:io-buffer` method of `w:basic-choose-variable-values`. Details about blips are described with the method that produces the blip.

CAUTION: When using blips, you should keep in mind that the blips may be discarded if the process has called any function that does not know what to do with them. Two such functions are `debugger` and `break`, so this situation can happen at any time.

Blips should either describe mouse actions, which can safely be ignored if they happen when they are not meaningful, or blips should notify the process to check other data structures. A blip should not be used to indicate a request or response from another process because this information must not be lost. Instead, put the data on a separate queue and have the process check the queue after every command. A blip that does nothing serves to wake up the process if it is waiting for input when the data goes on the queue.

Using the `:around` method for `:read-any` causes blips to be processed even in the middle of calls to `read`, to the debugger, and to other programs that do not look for blips. The `:around` method can examine the value being returned. If the value is one of certain types of blips, you can process it and then loop around, calling the original `:read-any` handler again without returning to the caller. If the value is anything else, simply return it.

Input Editor

8.4 The *input editor*, formerly known as the *rubout handler*, is a feature of all interactive streams, that is, streams that connect to terminals. Its purpose is to allow the user to edit minor mistakes in type-in. At the same time, it is not supposed to get in the way; input should be seen by Lisp as soon as a syntactically complete form has been typed. The definition of syntactically complete form depends on the function that is reading from the stream; for `read`, a syntactically complete form is a Lisp expression.

There are three types of common input editors, which offer varying capabilities:

- Some interactive streams, such as editing Lisp Listeners, have an input editor with the full power of the Zmacs editor.
- Most windows have an input editor that imitates Zmacs, implementing more than thirty common Zmacs commands.
- The cold load stream has a simple input editor that allows the user to rub out single characters and includes a few simple commands such as clearing the screen and erasing the entire input typed so far.

All three kinds of input editor use the same protocol. The three types of input editor and their common protocol are described in the following paragraphs.

How the Input Editor Works

8.4.1 The most difficult task for an input editor is to decide when the user is finished typing. The idea of an input editor is that the user can type in characters, and they are saved up in a buffer so that if the user changes his or her mind, the user can rub them out and type different characters. However, at some point, the input editor has to decide that the time has come to stop putting characters into the buffer and to let the function parsing the input, such as `read`, return. This is called *activating*. The right time to activate depends upon the function calling the input editor, and may be very complicated (if the function is `read`, figuring out when one Lisp expression has been typed requires knowledge of all the various printed representations, what all currently-defined reader macros do, and so on).

Input editors should not have to know how to parse the characters in the buffer to figure out what the caller is reading and when to activate; only the caller should have to know this. The input editor interface is organized so that the calling function can do all the parsing while the input editor does all the handling of rubouts, and the two are kept completely separate.

Basically, the input editor works as follows. When an input function that reads characters from a stream, such as `read` or `read-line`, is invoked with a stream that has `:rubout-handler` in its `:which-operations` list, that function *enters* the input editor. The input function then goes ahead reading characters from the stream. Because control is inside the input editor, the stream echoes these characters so that the user can see what is being typed. (Normally, echoing is considered to be a higher-level function outside the province of streams, but when the higher-level function tells the stream to enter the input editor, the higher-level function is also handing the responsibility for echoing to the input editor.) The input editor is also saving all these characters in a buffer (why it saves them is discussed shortly). When the function, `read` or whatever, decides it has enough input, it returns and control *leaves* the input editor. That is the easy case.

If the user types a rubout, a **throw** is done out of all recursive levels of **read**, reader macros, and so forth back to the point where the input editor was entered. Also, the rubout is echoed by erasing from the screen the character that was rubbed out. Now the **read** is tried over again, rereading all the characters that have not been rubbed out but not echoing them this time. When the saved characters have been exhausted, additional input is read from the user in the usual fashion.

The effect of this is a complete separation of the functions of the input editor and parsing, while at the same time the execution of these two functions mingles in such a way that input is always activated at the proper time. Because of this mingling, the parsing function (in the usual case, **read** and all macro-character definitions) must be prepared to be thrown through at any time; the function should have only trivial side effects because it may be called multiple times.

If an error occurs while inside the input editor, the error message is displayed and then additional characters are read. When the user types a rubout, it rubs out the error message as well as the character that caused the error. The user can then proceed to type the corrected expression; the input is reparsed from the beginning in the usual fashion.

Common Input Editors

8.4.2 The common input editor recognizes a subset of the Zmacs editor commands, including RUBOUT, CTRL-F, META-F and others. Pressing CTRL-HELP while in the input editor displays a list of the commands. The kill and yank commands in the input editor use the same kill history as Zmacs. so you can kill an expression in Zmacs and yank it back into the input editor with CTRL-Y. The input editor also keeps a command history of the most recent input strings (a separate command history for each stream), and the CTRL-C and META-C commands retrieve from this history just as CTRL-Y and META-Y do for the kill history.

When not inside the input editor, and when entering characters to a program that uses control characters for its own purposes, the control characters are treated the same as ordinary characters.

Some programs, such as the debugger, allow the user to type either a control character or an expression. In such programs, you are really not inside the input editor unless you have typed the beginning of an expression. When the input buffer is empty, a control character is treated as a command for the program (such as, CTRL-C to continue in the debugger); when there is text in the input editor buffer, the same character is treated as a input editor command. Another consequence of this is that the message you get by typing help varies, being either the input editor's documentation or the debugger's documentation.

The w:rubout-handler Variable

8.4.3 The way that the input editor is entered is complicated, because a **catch** must be established. The **w:rubout-handler** variable is non-nil if the current process is inside the input editor. This is used to handle recursive calls to **read** from inside reader macros and the like. If **w:rubout-handler** is nil and the stream being read from has a **:rubout-handler** in its **:which-operations**, functions such as **read** send the **:rubout-handler** message to the stream with the arguments of a list of options, the function, and its arguments. The input editor initializes itself and establishes its **catch**, then calls back to the specified function with **w:rubout-handler** bound to **t**.

User-written input reading functions should follow this same protocol to get the same input editing benefits as `read` and `read-line`.

`w:rubout-handler`

Variable

Specifies whether the current process is in the input editor.

Functional Interface
to an Input Editor

8.4.4 The easiest way to invoke an input editor for a particular piece of code is to use the `with-input-editing` special form. This special form provides the same functionality as the `:rubout-handler` method, but is typically more convenient to use.

`with-input-editing stream options {body-form}`*

Special Form

Executes *body-form* inside *stream*'s `:rubout-handler` method. If *body-form* does input from *stream*, such input is done with whatever rubout processing that *stream* implements.

options is an association list of keyword symbols and the arguments to them. The following options are acceptable to windows:

Keyword

Description

`(:activation function x-args)`

function is used to test characters for being activators. *function*'s arguments are the character read followed by the *x-args* from the option. If *function* returns non-`nil`, then the blip `(:activation numeric-argument)` is placed where it can be read with a `:read-any-tyi` method. *numeric-argument* is either 1 or -1: it is 1 if the current value of the numeric argument specified by the user of the input editor is either positive or unspecified, or it is -1 if the value is negative.

`(:command function x-args)`

Similar to `:activation` in terms of the test for characters, but command characters are handled differently. If *function* returns non-`nil`, the character is a command character and the `:rubout-handler` method immediately returns two values: `(:command character numeric-argument)` and `:command`. *numeric-argument* is either 1 or -1: it is 1 if the current value of the numeric argument specified by the user of the input editor is either positive or unspecified, or it is -1 if the value is negative. The input that was buffered remains in the buffer.

`(:do-not-echo char1 char2 ...)`

Similar to the `:activation` option except that:

1. Characters are listed explicitly.
2. The character itself is returned when it is read, rather than an `:activation` blip.

`(:editing-command (char doc)...)`

Specifies user-implemented editing commands. If any *char* in the alist is read by the input editor, it is returned to the caller (for example, to a `:read-any-tyi` method called by the function specified in the `:rubout-handler` method). The function specified in the `:rubout-handler` method should process these characters in appropriate ways and keep reading.

(Continued)

Keyword

Description

(:full-rubout <i>flag</i>)	If the user erases all of the characters and then presses the rubout character once more, control is returned from the input editor immediately. Two values are returned: <code>nil</code> and <i>flag</i> . In the absence of this option, the input editor simply waits for more characters.
(:initial-input <i>string</i>)	Treats the characters in <i>string</i> as type-ahead before reading anything from the keyboard.
(:initial-input-pointer <i>n</i>)	Specify this option to start out with editing pointer <i>n</i> characters from the start.
(:pass-through <i>char1 char2 ...</i>)	This option treats the characters <i>char1</i> , <i>char2</i> and so forth as ordinary characters even if they are normally special commands to the input editor. This can be useful for getting characters such as HELP into the buffer.
(:preemptable <i>token</i>)	This option makes all blips act like command characters. If the input editor encounters a blip while reading input, it immediately returns two values: the blip itself, and the specified token. Any buffered input remains buffered for the next request for input editing.
(:prompt <i>function-or-string</i>)	If <i>function-or-string</i> is a function, it is called before reading any characters; typically, the function displays a prompt. The arguments to the function are the window and a flag. When the input editor is first entered, the flag is <code>nil</code> , but if it is necessary to prompt again (for example, if the user clears the screen), the function is called again with the character the user typed. If <i>function-or-string</i> is a string, then that string is displayed before reading any characters, or when it necessary to prompt again.
(:reprompt <i>function-or-string</i>)	Acts the same as <code>:prompt</code> except that <i>function-or-string</i> is not used on initial entry. If both <code>:prompt</code> and <code>:reprompt</code> are specified, <code>:prompt</code> is used on initial entry and <code>:reprompt</code> is used for redisplay.

For example, the following code prompts the user from the stream and then reads one line of input from the user.

```
(with-input-editing
 (my-stream '((:prompt "Hello there"))) ; options
 (read-line my-stream)) ; body form
```

A Sample Input Editor Function

8.4.5 The following explanation describes how to write your own function that invokes the input editor. The `read` and `read-line` functions both work this way. You should use the `readline1` example below as a template for writing your own input editor function.

The following code is a simplified version of the `read-line` function. This function does not handle end-of-file conditions, use `:line-in` for efficiency, and so on.

```
(defun readline1 (stream)
  ;; If stream does input editing, get inside the input editor.
  (if (and (not w:rubout-handler)
          (member :rubout-handler (send stream :which-operations)
                  :test #'eq))
      (send stream :rubout-handler '() #'readline1 stream)
      ;; ELSE
      ;; Accumulate characters until return.
      (loop for ch = (read-char stream)
            with string = (make-array 0 :element-type :string-char
                                     :fill-pointer 0)
            until (or (null ch) (char= ch #\return))
            do (vector-push-extend ch string)
            finally (return string))))
```

The first argument to the `:rubout-handler` message is a list of options. The second argument is the function that the input editor should call to do the reading, and the rest of the arguments are passed to that function. Note that in the example above, `readline1` is passing itself to the `:rubout-handler` message as the function and is passing its own arguments as the arguments. This case occurs frequently. The returned values of the message are normally the returned values of the function (except sometimes when the `:full-rubout` option is used).

Stream Input Operations

8.5 You can use either stream operations (that is, methods) to obtain input from windows, or you can use Common Lisp-compatible functions to obtain input from windows.

Common Lisp-Compatible Read Functions and Methods

8.5.1 The functions discussed below follow the Common Lisp standard of returning a character object rather than a fixnum. There are analogous methods; however, in most cases you should use the function rather than the method because the function generally executes faster than the equivalent method. The methods are supplied for those cases where it is difficult to call the function.

The two most commonly used functions, `read` and `read-no-hang`, are described in the *Explorer Input/Output Reference* manual.

All of the functions and methods discussed in this paragraph include the same arguments:

stream — The stream from which the character or blip is read. By default, *stream* is `*standard-input*`.

eof-errorp — A flag that indicates whether to signal an error when an end of file is encountered, or whether to return *eof-value*. By default (`t`), the function signals an error when it encounters an end of file.

eof-value — The value returned when *eof-errorp* is not `t` and the function encounters an end of file.

recursive-p — Ignored. This argument is included for compatibility with Common Lisp.

w:read-any &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Function
:read-any &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Method of w:stream-mixin
w:read-any-no-hang &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Function
:read-any-no-hang &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Method of w:stream-mixin

Reads one character or blip from *stream*. If no input is available, **w:read-any** waits until a character of input becomes available. The character comes from the window's input buffer if the buffer contains any characters; otherwise, the character comes from the keyboard.

w:read-any-no-hang returns **nil** immediately if the buffer is empty rather than waiting. This function is used by programs that first execute continuously until a key is pressed, then examine the key and decide what to do next.

w:unread-any <i>char</i> &optional (<i>stream</i> * standard-input*)	Function
:unread-any <i>char</i> &optional (<i>stream</i> * standard-input*)	Method of w:stream-mixin

Puts the character just read back into the window's input buffer so that it is the next character returned by **w:read-any**. This character must be the very last character that was read. It is illegal to perform two **w:unread-any**s in a row. **w:unread-any** is used by parsers, such as **read**, that look ahead one character. *char* is the character just read from the window's input buffer.

w:read-list &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Function
:read-list &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Method of w:stream-mixin
Reads one blip from <i>stream</i> .	

w:read-mouse-or-kbd &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Function
:read-mouse-or-kbd &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Method of w:stream-mixin
w:read-mouse-or-kbd-no-hang &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Function
:read-mouse-or-kbd-no-hang &optional (<i>stream</i> * standard-input*) (<i>eof-errorp</i> t) <i>eof-value recursive-p</i>	Method of w:stream-mixin

Reads one character or **:mouse-button** blip from *stream*, except that certain kinds of blips are not discarded and do count as input. All other blips (that is, all blips whose **car** is *not* the symbol **:mouse-button**) are discarded. If these functions read a blip whose **car** is the symbol **:mouse-button**, they return three values:

1. The third element (**caddr**) of the blip, which is always a fixnum.
2. The whole blip.
3. A character whose mouse bit is 1. This bit identifies the mouse button that was clicked. (See paragraph 11.4.2, Encoding Mouse Clicks as Characters.)

If no input is available, `w:read-mouse-or-kbd` waits for input; `w:read-mouse-or-kbd-no-hang` returns `nil` rather than waiting.

Methods for Stream Operations

8.5.2 The methods of `w:stream-mixin` and `w:preemptable-read-any-tyi-mixin` flavors provide stream input operations. Some of the methods have an optional *eof-action* argument; this argument is included because it is part of the input stream protocol. It is ignored because end-of-file is not meaningful for windows.

`:force-kbd-input` *input* Method of `w:stream-mixin`

Inserts *input* into the window's input buffer to be read by `read-char` in its turn. This method is the standard way of putting blips into the input stream. If *input* is a character or a list, *input* is forced into the input buffer. If *input* is a string, each character in the string is forced into the input buffer, one by one.

`:listen` Method of `w:stream-mixin`

Returns `t` if there are any characters available to `:any-tyi`, or `nil` if there are not. For example, the editor uses `:listen` to defer redisplay until the editor has processed all of the characters that have been typed in.

`:string-in` *eof-action string* &optional (*start* 0) *end* Method of `w:stream-mixin`

Operates like `read-char`, except that it reads a string from the input buffer. `:string-in` reads characters into a string using `read-char` until no more room remains in the string.

Arguments: *eof-action* — Ignored because end-of-file is not meaningful for windows.

string — The string to be affected by the input.

start — The first character of *string* into which input is placed. If *start* is `nil`, `:string-in` starts reading with the first character in the string. Recall that the Explorer system uses a zero-based index for strings.

end — The last character *string* into which input is placed. If *end* is `nil`, all of *string* from *start* to the last character of the string is to be affected by the input.

Suppose that you evaluate the following code:

```
(setq string-var "A string")
(send w:selected-window :string-in t string-var 3 6)
```

If you type 123 in the input buffer, the new value of `string-var` is:

```
"A s123ing"
```

`:string-line-in` *eof-action string* &optional (*start* 0) *end* Method of `w:stream-mixin`

Operates like `read-char`, except that it reads a line from the input buffer. `:string-line-in` reads characters into a string using `read-char` until a `#\return` is encountered or no more room remains in the string. The arguments for `:string-line-in` are identical to the arguments used for `:string-in`.

`:string-line-in` returns three values:

1. An index, pointing to the position in *string* where the next character is placed.

2. Either **t** (if the end-of-file character was read) or **nil** (otherwise).
3. Either **nil** (if a **#\return** character was read) or **t** (otherwise).

The following are the possible permutations of these values.

Index	EOF	Value Returned	Where the Index Points
<i>nnn</i>	nil	t	The position in <i>string</i> where the next character read will be put—the eof was not reached, but the end of string was reached.
<i>nnn</i>	nil	nil	The position in <i>string</i> where the next character read will be put—the eof was not reached, and a #\return character was not the last character read.
<i>nnn</i>	t	t	One position beyond the end of <i>string</i> —the eof and end of <i>string</i> were reached simultaneously. This is an unlikely occurrence.
<i>nnn</i>	t	nil	The position in <i>string</i> where the next character read will be put—the eof was reached, and a #\return character was the last character read. This is an unlikely occurrence.

NOTE: *nnn* represents any number that can be returned by **:string-line-in** as a value.

-
- :wait-for-input-with-timeout** *timeout* Method of **w:stream-mixin**
 Waits until either input is available or a specific time period elapses. *timeout* is the time period, specified in 60ths of a second, that **:wait-for-input-with-timeout** waits.
- :clear-input** Method of **w:stream-mixin**
 Clears this window's input buffer. **:clear-input** flushes all the characters that have been typed in at this window but have not yet been read.
- :playback** Method of **w:stream-mixin**
 Returns an array of the last *n* characters read from this window, where *n* is the size of the array. The array elements are used in a circular fashion, the last element being followed by the first one, and array leader element 1 containing the index of the last array element into which a character was read. The editor command **HELP L** uses this method. *n* is the value of **w:io-buffer-record-length**, which has a default of 60.
- :rubout-handler** *options function &rest args* Method of **w:stream-mixin**
 Applies *function* to the arguments inside an environment where input operations from this window echo the characters typed and provide for simple input editing.
- options* is an association list of keyword symbols and the arguments to them. These options are listed in paragraph 8.4.4, Functional Interface to an Input Editor, followed by an example of using the **:rubout-handler** method. In most cases, it is more convenient to use the **with-input-editing** special form (also described in paragraph 8.4.4) than to use the **:rubout-handler** method.

:save-rubout-handler-buffer Method of **w:stream-mixin**

Returns the input editor buffer's contents and then clears the buffer. Two values are returned: a string and a fixnum. The fixnum is the current cursor index in the string. (Rubout handler is a historical term for input editor.)

:save-rubout-handler-buffer is used on entry to the **break** function so that pressing the **BREAK** key interfaces properly with rubout handling.

:restore-rubout-handler-buffer *string index* Method of **w:stream-mixin**

Loads the contents of the input editor buffer from *string* and sets the cursor position. The arguments are usually two values obtained from **:save-rubout-handler-buffer**.

Arguments: *string* — The string returned by the **:save-rubout-handler-buffer** method; it specifies the new buffer contents for the input editor after **:restore-rubout-handler-buffer** executes.

index — The fixnum returned by **:save-rubout-handler-buffer** method. The *index* argument sets the cursor position in *string*.

:refresh-rubout-handler &optional *discard-last-character* Method of **w:stream-mixin**

Requests the input editor to reprint its buffer and then reprompt. If *discard-last-character* is non-nil, the last character in the buffer is discarded. **:restore-rubout-handler-buffer** uses **:refresh-rubout-handler**.

If you are reading input using the input editor but want to process certain characters immediately (perhaps the character Help) and not leave them as part of the ordinary input, you should use this method and specify *discard-last-character* as *t*.

w:preemptable-read-any-tyi-mixin Flavor

Defines the **:preemptable-read** method.

:preemptable-read *options function* Method of **w:preemptable-read-any-tyi-mixin**
&*rest arguments*

Forces the window to read a blip immediately.

You may have noticed that when you use the mouse to select an object in the Inspector or the window error handler, the program sees the object you select. If, while typing a Lisp expression, you use the mouse to select an object, the blip sent by the mouse process is put at the end of the input buffer and is not read because of the characters that you have typed, unless you force the window to provide special handling for the mouse blip. The **:preemptable-read** method corrects this problem by putting the mouse blip at the front of the input buffer.

The **:preemptable-read** method uses the same arguments as the normal **:rubout-handler** method (discussed previously) and does the same thing if the mouse is not used. In fact, **:preemptable-read**, despite the name, has nothing to do with the read function. The **:preemptable-read** method returns the blip sent to the window as the first value and the **:mouse-char** symbol as the second value. (**:preemptable-read** does this even if the blip did not come from the mouse; however, most blips do.) The characters that were in the input editor buffer when the blip arrived come back the next time a **:preemptable-read** method is used, so the user can keep typing in his or her expression.

I/O Buffers

8.6 An I/O buffer is an array of fixed size used as a ring buffer. Typically, characters are put into the buffer by one process and removed by another in FIFO order. The process that is removing characters can wait if the buffer is empty, and a process that is putting in characters can wait if the buffer is full, or either process can throw away the characters. Each window using the `w:stream-mixin` flavor has an input buffer that is an I/O buffer, and there is also one global I/O buffer for the keyboard itself.

NOTE: I/O buffers are typically used for storing characters, which can also be lists. However, any Lisp objects can be stored in an I/O buffer; they do not have to be characters.

To provide easy access to slots in the I/O buffer leader, the following macros are defined for the slots. These macros work in a manner similar to standard macros. See the discussion of array leaders in the *Explorer Lisp Reference* manual for more information.

Slot Name	Description
<code>w:io-buffer-size</code> <i>io-buffer</i>	The number of elements in the input buffer.
<code>w:io-buffer-input-pointer</code> <i>io-buffer</i>	The index pointing* to where the next character inserted should be stored.
<code>w:io-buffer-output-pointer</code> <i>io-buffer</i>	The index pointing* to the next available character.
<code>w:io-buffer-input-function</code> <i>io-buffer</i>	Either a function called when characters are inserted, or <code>nil</code> . The window system does not actually use this feature. The calling conventions are the same as for the output function.
<code>w:io-buffer-output-function</code> <i>io-buffer</i>	Either a function called when characters are removed, or <code>nil</code> . <code>w:io-buffer-output-function</code> is called with the buffer and the character as arguments. The value of <code>w:io-buffer-output-function</code> should be a translated version of the character; this value is usually the same as the argument. <code>w:io-buffer-output-function</code> can also return a non- <code>nil</code> second character, indicating that the character should be discarded. In this case, <code>w:io-buffer-get</code> removes the following character or waits for one.

In window input buffers, this slot is usually a function that checks for and handles synchronous interception.

Note:
If the input and output pointers are equal, the buffer is empty. If `w:io-buffer-output-pointer` points at the slot after the input pointer, the buffer is considered full—in fact, one slot is still empty but cannot be used.

Continued

(Continued)

Slot Name Description

w:io-buffer-state *io-buffer*

One of the following, which controls the use of the buffer:

Value	Characters Can Be
t	Inserted or removed
nil or :input	Inserted
nil or :output	Removed

w:io-buffer-plist *io-buffer*

A property list that can contain one of the following:

- **:raw** is specified non-**nil** to inhibit translation of characters from hardware codes to the Explorer character set. The effect of **:raw** is hardware dependent.
- **:asynchronous-characters** is an association list controlling which characters are intercepted asynchronously when this window is selected. See paragraph 8.7.2, Asynchronously Intercepted Characters, for details.
- **:dont-upcase-control-characters** specifies whether letters entered in conjunction with the CTRL, META, HYPER, or SUPER keys receive special treatment or not. If **:dont-upcase-control-characters** is **nil**, lowercase letters are converted to uppercase and uppercase are converted to lowercase. If it is **t**, the case of the letters does not change.

w:io-buffer-last-input-process *io-buffer*

The last process that put a character in this I/O buffer.

w:io-buffer-last-output-process *io-buffer*

The last process that removed a character from this I/O buffer.

w:io-buffer-record *io-buffer*

An array that records the last *n* characters read from this I/O buffer, where *n* is the value of the **w:io-buffer-record-length** constant. This array is also a ring buffer, but nothing is ever removed from it; after it is full, it contains the last *n* characters stored in it. The **w:io-buffer-record-pointer** macro gets the index of the last slot stored into.

w:io-buffer-record-pointer *io-buffer*

Macro

Gets the index of the last slot of the **w:io-buffer-record** array.**w:io-buffer-empty-p** *io-buffer*

Macro

Returns **t** if *io-buffer* is empty, **nil** if it is not.**w:io-buffer-full-p** *io-buffer*

Macro

Returns **t** if *io-buffer* is full, **nil** if it is not.

- w:make-io-buffer** *size* &optional *input-function output-function* Function
plist state
- Creates and returns an I/O buffer, initializing some of the slots from its arguments and the other slots in a default or reasonable fashion. The buffer, initially empty, is of size *size* + 2, because 2 elements must always be empty.
- The arguments for **w:make-io-buffer** are the same as the respective slots described previously. Also see the **w:make-default-io-buffer** function described in paragraph 8.6.2, I/O Buffers as Input Buffers.
- w:io-buffer-put** *buffer character* &optional (*no-hang-p nil*) Function
w:io-buffer-push *buffer character* Function
- Inserts *character* into *buffer*. **w:io-buffer-put** puts the character at the end of the buffer; **w:io-buffer-push** puts the character at the beginning of the buffer, that is, as the next character to be removed. Thus, **w:io-buffer-put** implements a FIFO buffer, and **w:io-buffer-push** implements in a last-in, first-out (LIFO) buffer.
- no-hang-p* specifies whether **w:io-buffer-put** waits for a full I/O buffer to empty or not. If *no-hang-p* is **t** and the buffer is full, this function does not write to the buffer; instead, it exits and returns **nil**. If *no-hang-p* is **nil** and the buffer is full, this function waits for the buffer to empty, writes to the buffer, and then returns **t**.
- w:io-buffer-get** *buffer* &optional (*no-hang-p nil*) Function
- Removes the next character from *buffer*. The character removed is put in the buffer's I/O buffer record array. **w:buffer-get** also waits if the buffer's state does not permit output. However, if *no-hang-p* is non-**nil** and *buffer* is empty, **w:buffer-get** returns **nil** immediately. **w:io-buffer-get** returns two values—the character and either **t** (if the method actually got a character) or **nil** (if the method did not get a character).
- w:io-buffer-unget** *buffer character* Function
- Inserts *character* into *buffer* as the next character to be removed rather than as the last one to be removed. **w:io-buffer-unget** reverses the action of **w:io-buffer-get** and returns an error if the character does not match the last character removed by **w:io-buffer-get**. The character is removed from the I/O buffer record array by backing up the array's pointer to avoid duplication when the character is read a second time.
- This function should not be used more than once between input operations.
- w:io-buffer-clear** *buffer* Function
- Empties the I/O buffer identified by *buffer*.
- w:process-typeahead** *buffer function* Function
- Uses another function as a filter for the characters in the buffer.
- Arguments:** *buffer* — The buffer that **w:process-typeahead** accesses.
function — The function called once for each character in *buffer*; the character is its sole argument. If *function* returns non-**nil**, the returned value is stored back in *buffer* instead of the original character that was in the buffer. If *function* returns **nil**, the character is deleted from the *buffer*.

I/O Buffers and Type-Ahead 8.6.1 Generally, keyboard input goes into the selected window's input buffer. However, this is not always true. Program-generated input from methods using `:force-kbd-input` goes directly into the window's input buffer, but keyboard input actually goes into another I/O buffer called the *keyboard input buffer*. There is only one keyboard input buffer in the system. The characters move from the keyboard input buffer to the selected window's input buffer whenever a program tries to read input from the window's input buffer while the buffer is empty. The keyboard input is not assigned to a selected window until the instant the program is ready to read it.

Asynchronous window-switching commands, such as `TERM S` and mouse clicks that select a window, actually copy the contents of the keyboard input buffer into the buffer of the window that is being deselected. If you type commands to the editor and then type `SYSTEM L` before the editor has read its commands, those commands still go to the editor, not to the Lisp Listener you have selected.

By contrast, synchronous window-switching (such as is done by the functions `ed` and `inspect`, and by `Exit` commands in various programs) does not perform this copying operation, because any further typed-ahead input should go to the program being switched to.

I/O Buffers as Input Buffers 8.6.2 The following paragraphs discuss functions that affect the use of I/O buffers as input buffers.

w:make-default-io-buffer Function

Creates and returns an I/O buffer of the type used for window input buffers with all slots properly initialized. This function uses `w:kbd-default-output-function` as the output function.

w:kbd-default-output-function *buffer char* Function

Checks the character *char* from *buffer* against the value returned by `w:kbd-intercepted-characters` and also checks `w:kbd-tyi-hook`. This function must be called with the `inhibit-scheduling-flag` variable set to `t`; the function may set the flag to `nil`. This is the default function for a window input buffer's output function.

w:*default-read-whostate* Variable

Default: "Keyboard"

Contains the default whostate displayed for various conditions, as explained below. This variable is referenced by several functions and methods, including `:tyi`, `:any-tyi`, and so on.

w:kbd-io-buffer-get *buffer* &optional (*no-hang-p nil*) Function
(*whostate w:*default-read-whostate**)

Removes a character from *buffer*, or possibly from the keyboard input buffer. When a character is read from the keyboard input buffer, the buffer's output function is executed, as if the character had been put into *buffer* and then read from there.

Arguments: *buffer* — The buffer from which this function removes a character. `w:kbd-io-buffer-get` reads from the keyboard input buffer only if *buffer* is the input buffer of the selected window, and then only if *buffer* is empty.

no-hang-p — If this argument is non-*nil* and *buffer* is empty, this function returns *nil* immediately.

whostate — The string that appears in the status line while this function waits.

w:kbd-wait-for-input-with-timeout *buffer timeout* Function
&optional (*whostate w:*default-read-whostate**)

Waits either until **w:kbd-io-buffer-get** goes into a wait state because the buffer is empty or until *timeout* elapses. The string specified by *whostate* appears in the status line while you wait.

Arguments: *buffer* — The buffer from which this function removes a character. **w:kbd-wait-for-input-with-timeout** reads from the keyboard input buffer only if *buffer* is the input buffer of the selected window, and then only if *buffer* is empty.

timeout — The time period to wait before this function times out. *timeout* is specified in 60ths of a second.

whostate — The string that appears in the status line while this function waits.

w:kbd-wait-for-input-or-deexposure *buffer window* Function
&optional (*whostate w:*default-read-whostate**)

Waits until **w:kbd-io-buffer-get** goes into a wait state because the buffer or the window is not exposed.

Arguments: *buffer* — The buffer from which this function removes a character. **w:kbd-wait-for-input-or-deexposure** reads from the keyboard input buffer only if *buffer* is the input buffer of the selected window, and then only if *buffer* is empty.

window — The window to check for deexposure. If *window* becomes deexposed, this function returns immediately.

whostate — The string that appears in the status line while this function waits.

w:kbd-snarf-input *buffer* &optional *no-hardware-chars-p* Function

Moves all characters that can be obtained through the **w:kbd-io-buffer-get** function into *buffer*. Asynchronous selection commands use **w:kbd-snarf-input** to ensure that type-ahead for the window being deselected remains with that window.

no-hardware-chars-p is *nil* if **w:kbd-snarf-input** is to get the characters directly from the hardware.

w:kbd-char-typed-p Function

Returns non-*nil* if input is available in the selected window. This function, which can be used in programs that loop with interrupts disabled, indicates when the user presses a key, that is, when input is typed.

Intercepted Characters

8.7 The window system intercepts two categories of characters:

- *Synchronously* intercepted characters are intercepted when a process tries to read them.
- *Asynchronously* intercepted characters are intercepted as soon as they are typed. Asynchronous characters are either *global asynchronous characters*, such as `#\terminal` and `#\system`, which are always available, or characters defined by the selected window, such as CTRL-ABORT.

An application program can specify the function keys, F1 through F4, as either synchronously or asynchronously intercepted characters. If you use the function keys in an application, you should also document their use in the mouse documentation window.

Synchronously Intercepted Characters

8.7.1 The **io-buffer-output-function** of the window input buffer performs synchronous interception. By default, this function is **w:kbd-default-output-function**, which uses the **w:kbd-intercepted-characters** variable to decide which characters to intercept and how to handle them. A program can change the set of synchronously intercepted characters for the program's window simply by binding this variable before reading input. Its default value specifies the characters ABORT, META-ABORT, BREAK, and META-BREAK.

By convention, all programs are expected to use the ABORT key as a command to abort operations in some sense appropriate for that program. If you do not do anything special, ABORT is intercepted automatically. However, you may want some programs to do something other than the system default action when the user presses ABORT. The system default action can be replaced by binding the **w:kbd-intercepted-characters** variable so that ABORT goes to your own intercept routine instead of to **w:kbd-intercept-abort**, or so that ABORT is read as an input character from the stream and is then handled by your program.

w:kbd-intercept-abort <i>char &rest ignore</i>	Function
w:kbd-intercept-abort-all <i>char &rest ignore</i>	Function
w:kbd-intercept-break <i>char &rest ignore</i>	Function
w:kbd-intercept-error-break <i>char &rest ignore</i>	Function

Implements the standard meanings of various keystrokes and keystroke sequences, as follows:

Function	Implements Keystroke
w:kbd-intercept-abort	ABORT and CTRL-ABORT
w:kbd-intercept-abort-all	META-CTRL-ABORT and META-ABORT
w:kbd-intercept-break	BREAK
w:kbd-intercept-error-break	META-BREAK

In addition, `w:kbd-intercepted-characters` uses the following keystroke sequences for special purposes:

- ABORT signals the `sys:abort` condition.
- META-CTRL-ABORT and META-ABORT reset the current process.
- BREAK calls the `break` function.
- META-BREAK calls the debugger.

If you want to handle ABORT, META-CTRL-ABORT, or META-ABORT differently, redefine either `w:kbd-intercept-abort` or `w:kbd-intercept-abort-all`, as appropriate, to perform as desired.

If `*terminal-io*` handles the `:inhibit-output-for-abort-p` method and `:inhibit-output-for-abort-p` returns non-nil, the string [Abort] is not printed.

The *char* argument is not used.

`w:kbd-intercepted-characters`

Variable

An association list specifying the characters to be intercepted synchronously, that is, when the characters are read by the program. Because a subroutine of the `read-char` function examines `w:kbd-intercepted-characters`, the current binding at the time `read-char` executes is important.

Each element of `w:kbd-intercepted-characters` should have the format of (*character function*). With this format, a function specified by *function* is called if *character* is read, with *character* as the argument.

function should return two values: a character and either `t` or `nil`. The second value determines whether the first value is accepted as input:

- If the second value is `t`, *function* processes *character*, and *character* is ignored.
- If the second value is `nil`, *character* is to be inserted into the input buffer.

In practice, *function* usually returns its argument and `t`.

On entry, *function* needs to set the `inhibit-scheduling-flag` to `nil` to ensure proper changing of the `w:kbd-intercepted-characters` variable. (The `inhibit-scheduling-flag` is described in the *Explorer Lisp Reference* manual.)

New entries can be added to the top-level value of `w:kbd-intercepted-characters` and can also bind `w:kbd-intercepted-characters` for programs. It is unwise to remove the standard entries in the top-level value. The default value for `w:kbd-intercepted-characters` is the value of `w:kbd-standard-intercepted-characters`, which is the following:

```
^((#\abort kbd-intercept-abort)
  (#\m-abort kbd-intercept-abort-all)
  (#\break kbd-intercept-break)
  (#\m-break kbd-intercept-error-break))
```

See the previous discussion of `w:kbd-intercept-abort` and related functions for a description of specific keystrokes and their meanings.

w:kbd-standard-intercepted-characters Variable

The initial value of **w:kbd-intercepted-characters**. The **w:kbd-standard-intercepted-characters** variable can be used to reset **w:kbd-intercepted-characters** to its original value after a program finishes special handling of intercepted characters.

The default value for **w:kbd-standard-intercepted-characters** is the following:

```

`((#\abort kbd-intercept-abort)
  (#\m-abort kbd-intercept-abort-all)
  (#\break kbd-intercept-break)
  (#\m-break kbd-intercept-error-break))

```

w:kbd-tyi-hook Variable

When this variable is non-**nil**, a user can process synchronously intercepted characters.

w:kbd-default-output-function checks whether the value of **w:kbd-tyi-hook** is non-**nil** before **w:kbd-default-output-function** does anything else. If the value is non-**nil**, **w:kbd-default-output-function** assumes that the value is a function of one argument and applies the function to the character that was typed. If the function returns a non-**nil** value, then the character is not returned to callers of **read-char** or other input operations; otherwise, the character is processed normally.

This variable allows you to write a function that intercepts anything passing through an input buffer using **w:kbd-default-output-function**. Your function intercepts the character and returns either **nil** if it does not want to handle the character, or **t** if your function processes the character.

Asynchronously Intercepted Characters

8.7.2 Each window that uses **w:stream-mixin** can define a set of characters to be intercepted asynchronously when that window is selected. The interception is performed through a different mechanism than the one used for synchronous interception, but the same handling functions, such as **w:kbd-intercept-abort**, can ultimately be used. By default, a window performs asynchronous interception on the four characters generated by the **CTRL-ABORT**, **META-CTRL-ABORT**, **CTRL-BREAK**, and **META-CTRL-BREAK** keystrokes. You can change the set of such asynchronous keys on a window by window basis.

The keyboard process and its handler function perform asynchronous interception. Therefore, asynchronous interception must obey certain strict conventions: asynchronous interception must not perform any I/O, wait for anything, run very long, or get an error. It is usually easiest to create another process and perform the real work there, using **process-run-function**.

Because the interception is performed by the keyboard process, the characters cannot be directly specified by a variable for the program to bind. So each window has a list of asynchronously intercepted characters. The list is actually stored as the **:asynchronous-characters** property on the input buffer's property list.

w:kbd-standard-asynchronous-characters

Variable

Contains the default set of characters that are asynchronously intercepted. This set includes the following:

```

^((#\c-abort kbd-asynchronous-intercept-character
  (:name "abort" :priority 50) kbd-intercept-abort)
 (#\c-m-abort kbd-asynchronous-intercept-character
  (:name "Abort all" :priority 50) kbd-intercept-abort-all)
 (#\c-break kbd-asynchronous-intercept-character
  (:name "break" :priority 40) kbd-intercept-break)
 (#\c-m-break kbd-asynchronous-intercept-character
  (:name "Error break" :priority 40)
  kbd-intercept-error-break))

```

:asynchronous-characters *alist*Initialization Option of **w:stream-mixin**Default: **w:kbd-standard-asynchronous-characters**

Specifies the list of characters intercepted asynchronously while this window is selected.

alist specifies the characters to be intercepted asynchronously. Each element of *alist* consists of a character, a function to call, and optionally some extra arguments to be passed to the function. When the function is called, the function's arguments specify the character, the selected window, and any specified additional arguments from *alist*.

:asynchronous-character-p *character*Method of **w:stream-mixin**

Returns non-nil if this window defines a character for asynchronous interception. *character* is the character that can be intercepted.

:handle-asynchronous-character *character*Method of **w:stream-mixin**

Invokes the handler function in your current process as defined for asynchronous interception of a character. *character* is the character to be immediately intercepted when it is typed.

:add-asynchronous-character *character handler-function*
*&rest additional-args*Method of **w:stream-mixin**

Defines a character for asynchronous interception in this window, that is, adds the element (*character handler-function . additional-args*) to the association list on the input buffer's property list.

:remove-asynchronous-character *character*Method of **w:stream-mixin**

Removes a character's element from the association list so that the character is no longer intercepted asynchronously in this process. *character* is the character to be removed.

w:kbd-asynchronous-intercept-character *character window*
&optional process-run-options function

Function

Provides a convenient way to handle asynchronously intercepted character(s). This function enables you to use the same functions used for synchronous interception.

Arguments: *character* — The character to be intercepted.

window — The window that intercepts *character*.

process-run-options — The option list for **process-run-function**, which is described in the *Explorer Lisp Reference* manual.

function — The subhandler function that actually performs the asynchronous action. When the function is called, it is passed the character and window as arguments.

The following example creates a process named `Break` with priority 40 and calls `w:kbd-intercept-break` in that process:

```
(#\ctrl-break w:kbd-asynchronous-intercept-character
 (:name "Break" :priority 40.)
 w:kbd-intercept-break)
```

Global Asynchronous Characters

8.7.3 The `TERM` and `SYSTEM` keys are also intercepted asynchronously, but because their functions do not usually relate to the selected window, they are not controlled by the selected window's association list of asynchronous characters. These are called *global asynchronous characters*.

The `TERM` and `SYSTEM` keys are defined to call functions that read a second character and then execute an operation depending on what that second character is. The meaning of the second character is controlled by an association list, so you can define new terminal and system commands.

`w:kbd-global-asynchronous-characters`

Variable

An association list that controls the characters intercepted regardless of the selected window. Its elements look and work exactly like those of the association list specified in the `:asynchronous-characters` initialization option for a window.

The initial value of `w:kbd-global-asynchronous-characters` is the following:

```
((#\term w:kbd-terminal)
 (#\system w:kbd-sys)
 (#\c-clear-input kbd-terminal-clear))
```

`w:*terminal-keys*`

Variable

`w:add-terminal-key` *char function &optional documentation &rest options*

Function

`w:remove-terminal-key` *char &rest ignore*

Function

`w:*user-defined-terminal-keys*`

Variable

Default: nil

The variables are association lists; each entry describes a subcommand of the `TERM` key. `w:*terminal-keys*` is the default list defined for the Explorer system; `w:*user-defined-terminal-keys*` is the list of terminal keys added by the user. You should use `w:add-terminal-key` or `w:remove-terminal-key` to modify the lists; these functions modify the appropriate list, as needed. Entries on the list have the following form.

char function documentation option1 option2 ...

where:

char — The character that should be typed after pressing the `TERM` key to get the new command. The character is mapped to uppercase before it is searched for in `w:*terminal-keys*` list, so you should not use lowercase characters.

function — Either a list to be evaluated or a symbol that is the name of a local function to be applied to one argument. The argument passed to

function is either the numeric argument specified by the user (for example, the 0 in TERM 0 S) or nil if the user gave no argument.

documentation — A string giving brief documentation, or a form that is evaluated and that returns either a string or nil. The string is displayed by pressing the TERM HELP keystroke sequence. A value of nil causes this character to be omitted from the TERM HELP display.

option1 and *option2* — Keywords with no associated values. The keywords can be one of the following:

- **:keyboard-process** — *function* runs in the keyboard process. (*function* is normally evaluated or applied in a new process created for the purpose.) The **:keyboard-process** option exists because some built-in commands must work this way. If you add your own commands, you should not use this option, because you do not want to interfere with the operation of the keyboard process.
- **:typeahead** — Everything typed before pressing the TERM key is inserted into the selected I/O buffer; that is, this text is treated as type-ahead to the currently selected window. You should use **:typeahead** with commands that change the selected window, to ensure that the user's typed input goes where he or she expects it to go. These commands should set **w:kbd-terminal-time** to nil as soon as they change the selected window, unless they complete quickly. (You should never gather characters for a TERM command while **w:kbd-terminal-time** is non-nil.)
- **:system** — The code is actually overriding the system code rather than adding a different user-defined command.

The following are typical entries of **w:*terminal-keys***:

```
(#\CLEAR-INPUT tv:kbd-terminal-clear "Discard type-ahead" :keyboard-process)
(#\RESUME (tv:kbd-terminal-resume)
  "Allow deexposed timeout in window that TERM-0-S would select.")
(#\A tv:kbd-terminal-arrest "Arrest process in status line (minus means unarrest)"
  :keyboard-process)
(#\B tv:kbd-bury "Bury the selected window" :typeahead)
(#\C tv:kbd-complement `("Complement video black-on-white state"
  " With an argument, complement the who-line documentation window")
  :keyboard-process)
```

w:*system-keys*

w:add-system-key *char find documentation &optional (create t)*

w:remove-system-key *char*

Variable

Function

Function

The variable is an association list; each entry describes a subcommand of the SYSTEM key. You should use **w:add-system-key** or **w:remove-system-key** to modify the list. Entries on the list have the following form. The arguments for **w:add-system-key** are analogous to these entries.

char find documentation create

where:

char — The character that should be typed after pressing the SYSTEM key to get the new command. The character is mapped to uppercase before it is searched for in the **w:*system-keys*** list, so you should not use lowercase characters. You can also use modified characters such as META-A, SUPER-B, and HYPER-C. CTRL is reserved for creating a

new instance of the window and process. For example, SYSTEM CTRL-E creates a new Zmacs editor. If you assign SYSTEM SUPER-F as the keystroke associated with the frobbos process, SYSTEM SUPER-CTRL-F creates a new frobbos process.

find — One of the following:

Value of <i>find</i>	Description
An instance of a flavor	The instance should be a window, and pressing the SYSTEM key selects that particular window.
The name of a flavor	This is the typical case. Pressing the SYSTEM key initiates a search of the <code>w:previously-selected-windows</code> variable. If this search finds a window having the proper flavor, the window is selected. If the currently selected window is of that flavor, the system causes a beep. Otherwise, the system creates a window of the proper flavor, establishes the process, and selects the window, among other operations.
A list	<i>find</i> is evaluated, and the value should be a window or a flavor name to be used as described previously.
Otherwise	SYSTEM evaluates <i>create</i> .

documentation should be a string to be printed by SYSTEM HELP.

create — Can be one of the following:

- `nil` — The system beeps when you press the SYSTEM key.
- `t` — A new window of flavor *find* is created by calling `make-instance` with no options, and *create* is selected.
- A symbol — The name of the flavor of a window to be created. (*create* can be different from the flavor to look for, which might be a mixin that is a component of several different flavors, all of which are suitable to select when the SYSTEM key is pressed.)
- Other cases — A form to be evaluated to create a window. SYSTEM starts a newly created process, and *create* is evaluated in its own process, not in the keyboard process.

If—after the SYSTEM key is pressed—the next key pressed is pressed in conjunction with the CTRL key, existing windows are ignored, and a new window is created according to *create*.

For example, the list in `w:*system-keys*` that includes the Zmacs editor on the System menu is as follows:

```
(#\E zwei:zmacs-frame "Editor" t)
```

To add items to the System menu, use the `:w:add-to-system-menu-column` function described in paragraph 18.6, The System Menu.

If you do not bind a window with a process to a SYSTEM keystroke, you must explicitly select, expose, and bind the process to the window, and then

reset the process and establish its run reason. An example of this is in paragraph 6.6.3, Associating a Process With a Window.

w:find-window-of-flavor *flavor-name* Function

Returns a previously selected window. This function searches the **w:previously-selected-windows** variable for a window of the specified *flavor* and uses the **typep** method to check the window.

w:select-or-create-window-of-flavor *find-flavor* Function
&optional (*create-flavor find-flavor*)

Selects a previously selected window or, if none exists, creates a new one and selects it. *find-flavor* is the window flavor to select. *create-flavor* is the flavor to use if a window must be created.

Querying the Keyboard Explicitly

8.8 The keyboard is often read as a stream of input; however, the keyboard can be treated as a random-access device, and the instantaneous state of any key can be examined.

w:key-state *key-name* Function

Returns **t** if the keyboard key specified by *key-name* is currently pressed, or **nil** if it is not. This argument can be either a character code or the symbolic name of a shift key:

- The character code of a key is the character produced when you type that key without any shifts: a lowercase letter, a digit, or a special character. These names or keys are predefined. (See the *Explorer Lisp Reference* manual for a list of characters.)
- Shift keys that come in pairs (such as SHIFT, META, and so forth) have three symbolic names: one for the left key only, one for the right key only, and one for both. The following table lists the symbolic names for the shift keys.

Key Pressed	Symbolic Name for Either Key	Symbolic Name for the Left-Hand key	Symbolic Name for the Right-Hand Key
SHIFT	:shift	:left-shift	:right-shift
SYMBOL	:symbol	:left-symbol	:right-symbol
CTRL	:control	:left-control	:right-control
META	:meta	:left-meta	:right-meta
SUPER	:super	:left-super	:right-super
HYPHER	:hyper	:left-hyper	:right-hyper
CAPS LOCK	:caps-lock	—	
MODE LOCK	:mode-lock	—	

For example, the following code displays a message in the status line while it waits for you to press the a key. The system ignores any other alphanumeric

or symbol keys pressed until you press that key, then inserts all the keystrokes into the input buffer.

```
(defun key-test ()
  (let ((sel-wd w:selected-window))
    (cond ((neq (send sel-wd :read-any
                     (process-wait "Press the 'a' key."
                                   #'w:key-state #\a))
           (char-int #\a)))
          )))
```

Other keys require more complicated code to verify whether a pressed key is the same as a particular key you are checking for. The following table shows tests for some of the more common keys:

Key Being Tested for	Key Name Used With charint	Code Used to Compare
Any lowercase alphanumeric key or unshifted symbol key, such as a, 1, or .	(char-int #\a)	(#'w:key-state #\a)
Any uppercase alphanumeric or shifted symbol key, where you obtained the uppercase by pressing the SHIFT key, such as A, @, or >	(char-int #\A)	#'(lambda () (and (w:key-state #\a) (w:key-state :shift)))
Any named key, such as HELP or ESCAPE	(char-int #\help)	(#'w:key-state #\help)
Any key that includes a modifier, such as META-HELP or CTRL-SHIFT-E	(char-int #\meta-help)	#'(lambda () (and (w:key-state #\help) (w:key-state :meta)))

Keyboard Parameters

8.9 The following operations are used to tailor the keyboard to the user's needs.

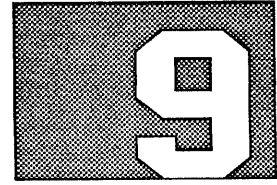
w:setup-keyboard-keyclick &optional (*state* nil) Function

Sets whether the keys, when pressed, make an audible click. *state* can be either *t* (when a key is pressed or released, it makes a clicking sound) or *nil* (the keys are silent).

:keypad-enable *t-or-nil* Initialization Option of *windows*

Default: *nil*, the keypad and the rest of the keyboard produce the same character codes

Determines whether pressing the keys on the keypad produces the same character codes as pressing the analogous keys on the keyboard or whether it produces unique character codes. When this initialization option is set and a keypad key is pressed, the code produced by that keypad key has its keypad character object bit set to *t*. The **char-bit** and **set-char-bit** functions can accept the **:keypad** keyword as their bit map arguments, which allows a program to distinguish keypad characters.



Introduction

9.1 If you have used the Explorer system for a while, you have probably noticed that characters can be typed out in any of a number of different typefaces. Some text is printed in characters that are small or large, boldface or italic, or in different styles altogether. Each such typeface is called a *font*. A font is an array, indexed by character code, of pictures showing how each character should be drawn on the screen.

A font is represented inside the Explorer system as a Lisp object. Each font has a name. The name of a font is a symbol, usually in the fonts package, and the symbol is bound to the font. A typical font name is `tr8`. In the initial Lisp environment, the symbol `fonts:tr8` is bound to a font object whose printed representation is something like the following:

```
#<font tr8 234712342>
```

The initial Lisp environment includes many fonts. Usually, there are more fonts stored in xld files. New fonts can be created, saved in xld files, and loaded into the Lisp environment; they can also simply be created inside the environment.

The drawing of characters in fonts is performed by microcode and is very fast. The internal format of fonts is arranged to make this drawing as fast as possible.

You can use the List Fonts command in Zmacs to display a list of all of the fonts that are currently loaded into the Lisp environment. Table 9-1 is a list of some of the useful fonts. For a complete list of all fonts that are loaded with the Explorer system, see Appendix A of the *Explorer Tools and Utilities* manual.

Specifying Fonts

9.2 You can control which font is used when output is sent to a window. Every window has a *font map* and a *current font*. The font map is an array of fonts; the font map associates a font with a small, nonnegative integer. The current font of a window is always one of the fonts in the window's font map. Whenever output is sent to a window, the characters are printed in the current font. You can change the font map and the current font of a window at any time with the appropriate methods.

Windows have a font map rather than merely a current font because it is necessary to know all the fonts in use before processing any output to ensure that output is positioned properly (so that output in different fonts on the same line is aligned correctly).

In addition, certain output methods can accept arrays that contain elements of type `:string-char`, and regard the font field as a font number to look up in the font map. These methods include `:compute-motion`, `:string-length`, and `:fat-string-out`.

Table 9-1


Some Commonly Used Fonts

Font	Description
cptfont	The default font for the U.S. market, used for almost everything.
medfnt	The default font in menus. It is a fixed-width font with characters somewhat larger than those of cptfont.
medfntb	A boldface version of medfnt; the default font for the European market. When you use the Split Screen, for example, the Do It and Abort items are in this font.
hl10	A very small font used for items in choose-variable-values windows that are currently not selected.
hl10b	A boldface version of hl10, used for selected items in choose-variable-values windows.
hl12i	A variable-width italic font. It is useful for italic items in menus.
tr10i	A very small italic font.
mouse	A font that contains the glyphs used as mouse cursors and icons.

```

;; -*- Mode:Common-Lisp; Fonts:(CPTFONT MEDFNT MEDFNTB HL12I TR10I HL10 HL10B MOUSE); Base:10 -*-
A sample of cptfont. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}
A sample of medfnt. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}
A sample of medfntb. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}
A sample of hl10. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}
A sample of hl10b. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}
A sample of hl12i. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}
A sample of tr10i. abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890-=" !@#%^&*()_+{}

```



Font Purposes

9.3 Because different users prefer different fonts, a facility called *font purposes* is provided. Wherever a font specifier is used, the program can specify a *purpose* keyword instead. This keyword means, *use whatever font the user prefers for this particular purpose*. The window remembers when a purpose keyword is specified instead of a particular font so that if the user associates a different font with that purpose, all the existing windows that use that purpose use the newly-associated font.

Each screen has its own association list that maps font purposes to font names, but normally they are all altered in parallel. Defined purpose keywords include the following:

Table 9-2

Purpose Keywords for Fonts	
Keyword	Description
:default	The font name for ordinary output. It is also called the <i>default font</i> .
:menu	The font name for use in most menu items.
:menu-standout	The font name for menu items that are supposed to stand out. It is normally a bold font.
:label	The font name used by default for labels.
:margin-choice	The default font name for margin choice boxes.

See paragraph 9.5, Font Specifiers, for a description of functions that can be used to manipulate font purposes.

Flavors and Methods

9.4

:font-map *new-map* Initialization Option of *windows*
Gettable, settable. Default: **w:*window-standard-font-map***
w:sheet-font-map *window* Macro

Initializes the **w:font-map** instance variable to contain the fonts given in *new-map*. *new-map* can be one of the following:

- An array of font specifiers — This array is installed as the new internal array of the window, and the font specifiers are replaced by fonts. A font or the name of a font can be used in the array. (Font specifiers are described in paragraph 9.5.)
- A list of font specifiers.
- **nil** — The font map is set to **w:*window-standard-font-map***.

new-map has a maximum length of 26.

The current font is set to the first font in the list or array. The line height and baseline of the window are adjusted appropriately.

The chosen font specifiers are remembered so that the **:change-of-default-font** method can cause the map to be recomputed from them. This feature is provided in case one of the specifiers is a font purpose keyword.

The **:font-map** method returns the array that is actually being used to represent the font map inside the window. The elements are actual font objects. You should not alter anything about this array because the window depends on it to function correctly. To change the font map, use the **:set-font-map** method.

The macro accesses the instance variable.

The following example of the `:font-map` initialization option sets up the font map for the `hl12b` font.

```
(:font-map (list fonts:hl12b))
```

`:current-font`

Method of *windows*

Settable. Default: The first font in the font map

`w:sheet-current-font` *window*

Macro

Returns the window's current font. `:set-current-font` can take either of the following as arguments:

- A number — That element of the font map becomes the current font. The number 0 corresponds to the first font in the font map.
- A font specifier — The font that the specifier describes is used. If that font is not in the font map, an error is signaled. Only fonts already in the font map can be selected.

The macro accesses the instance variable.

Font Specifiers

9.5 Different kinds of screens require different kinds of fonts. However, it is preferable to be able to write programs that work no matter what screen their window is created on. The problem is that if your program specifies which fonts to use by actually naming specific fonts, then the program works only if the window that you are using is on the same kind of screen as that for which the fonts you are using were designed.

To solve this problem, a program does not have to specify the actual font to be used. Instead, it specifies a symbol that stands for a collection of fonts. All of these fonts are the same except that they work on different kinds of screens. The symbol that you use is the name of the member of the collection that works on a black-and-white screen. In other words, when you want to specify a font, always use the name of a black-and-white font rather than a font itself. Every screen knows how to understand these symbols and how to find an appropriate font to use. This symbol is called a *font specifier* because it describes a font rather than actually naming a font.

For example, if you want to use `cptfont`, you would specify it as the symbol `fonts:cptfont`, which for a black-and-white screen is also the name of the font. For use with the IMAGEN printer, the symbol `fonts:cptfont` is interpreted as `imagen-fonts:cmasc10` (a font specific to the IMAGEN printer). If you added a frobboz brand color monitor to the system, you could create a font designed to work with that particular monitor. Assume you named this font `color-cptfont`. Then you could add `color-cptfont` to the property list of the `fonts:cptfont` symbol by executing the following form:

```
(setf (get 'fonts:cptfont) fonts:color-cptfont)
```

Thus, whenever you output text to the frobboz-brand monitor, text that would be in `cptfont` on the default monitor would be displayed in the `color-cptfont` instead.

A font object can be supplied as a font specifier. This does not mean to use the font as specified; it means to use the font's name as a font specifier.

The functions that understand font specifiers have some intelligence to make life easier for you. If you enter the name of a font that is not loaded into the Lisp environment, an attempt is made to load it from the file server, using the name of the font as the name of the file, leaving the version and type unspecified, and using the load function. The pathname used is `SYS:FONTS;fontname.XLD`.

:parse-font-specifier *font-specifier* Method of `w:screen`

Parses a font specifier in the proper way for this window, according to the screen the window is on. The value returned is a font object. *font-specifier* can be any of the things previously discussed in Table 9-1, Table 9-2, and the beginning of this paragraph.

:parse-font-name *font-specifier* Method of `w:screen`

Parses a font specifier in the proper way for this window, according to the screen the window is on. The value returned is a font name: a symbol that, when evaluated, produces a font.

w:font-evaluate *font-name* &optional (*screen w:main-screen*) Function

Returns the font specified by *font-name*. *font-name* can be a font specifier or a font purpose keyword. The optional *screen* argument is needed only where *font-name* is a font purpose keyword. In this case, *screen* is either a sheet or screen object used to determine the screen for which the font purpose is evaluated. *font-name* is evaluated repeatedly until the result is not a symbol, is an unbound symbol, or is nil.

:change-of-default-font *old-font new-font* Method of `windows`

Informs the window that the meaning of some standard font-name symbols has changed. If the window uses any of them, it may need to recompute the parameter that must change when a font changes. For example, if the changed font is used in the label, the window's inside size may be changed; if it is used in the window's font map, the line height may be changed. Either situation may cause the number of lines to change, and this may require adjustment of other data. This adjustment can be performed by an `:after` method on this method.

In addition, the method must be passed along to all inferiors and potential inferiors.

w:set-standard-font *font-purpose font-specifier* Function

w:set-screen-standard-font *screen font-purpose font-specifier* Function

Sets the standard font. **w:set-standard-font** sets the standard font on all screens. All windows on the screen that were set up to use the standard font for this purpose switch to using the newly specified font. **w:set-screen-standard-font** sets the standard font only on the screen specified by *screen*, rather than on all screens.

Arguments *font-purpose* — The purpose of the font. The font-purpose keywords are `:default`, `:menu`, `:menu-standout`, `:label`, and `:margin-choice`, which are described in Table 9-2.

font-specifier — The font that is to be the standard font.

w:get-standard-font *font-purpose* &optional (*the-screen* w:default-screen) Function

Returns the font object assigned for this *font-purpose*. *font-purpose* is a keyword, one of those described in Table 9-2, that indicates the kind of use. *the-screen* is the screen associated with this font purpose.

w:make-font-purpose *font-specifier font-purpose* Function

Adds or alters the *font-purpose* to be associated with a particular font (*font-specifier*). This function affects all the screens.

For example, consider the following code, which defines the font purpose **:flashy** to be the 43vxms font, a gigantic gothic font.

```
(w:make-font-purpose fonts:43vxms :flashy)
```

Any reference to the **:flashy** font purpose refers to the font 43vxms. For example, the following form

```
(defvar my-window)
(progn
  (setq my-window
        (make-instance 'w:window :font-map '(:flashy) :expose-p t))
  (send my-window :string-out "A flashy window."))
```

creates an instance of a window with the font map set to the font purpose **:flashy**. Any output displayed on this window instance is done using the font 43vxms.

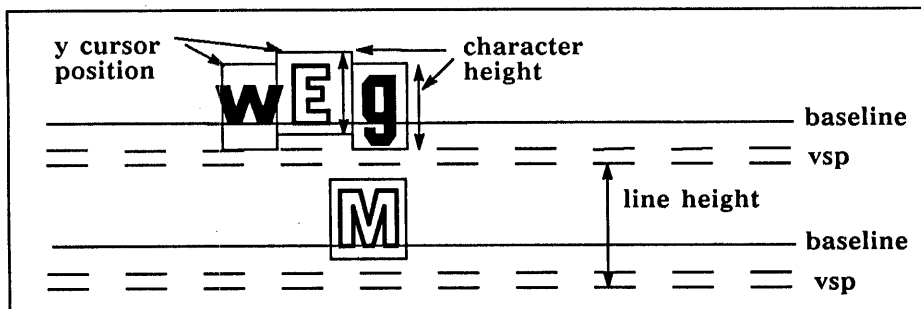
Suppose you want to change the **:flashy** font purpose to be a different font, say hl12bi. The following form accomplishes this change.

```
(w:make-font-purpose fonts:hl12bi :flashy)
```

This function first verifies that the **:flashy** font purpose already exists, and then changes it to be a different font, hl12bi. Subsequent output to *any* window referring to the **:flashy** font purpose in its font map uses hl12bi instead of 43vxms.

Attributes of Fonts 9.6 Fonts, and characters in fonts, have several attributes. One attribute of each font is its *character height*. This is a nonnegative fixnum used to determine how tall to make the lines in a window. Recall that each window has a certain *line height*. The line height is computed by examining each font in the font map and finding the one with the largest character height. This largest character height is added to the *vsp* specified for the window, and the sum is the line height of the window. The line height, therefore, is recomputed every time the font map is changed or the *vsp* is set. This procedure ensures that there is always enough room on any line for the largest character of the largest font to be displayed, while still leaving the specified vertical spacing between lines. One effect of this feature is that if you have a window that has two fonts, one large and one small, and you produce output in only the small font, the lines are still spaced far enough apart that characters from the large font fit. This spacing is used because the window system cannot predict when you might, in the middle of a line, suddenly switch to the large font.

Another attribute of a font is its *baseline*. The baseline is a nonnegative fixnum that is the number of raster lines between the top and base of each character. The base is usually the lowest point in the character, except for letters that descend below the baseline such as lowercase p and g. The baseline is stored so that when you are using several different fonts side by side, they are aligned at their bases rather than at their tops or bottoms. When you output a character at a certain cursor position, the window system first examines the baseline of the current font, then draws the character in a position adjusted vertically to make the bases of all the characters line up.



Note that the boxes that surround the characters are analogous to the boxes used in the font editor to create the characters.

:baseline

w:sheet-baseline *window*

Method of *windows*

Macro

Returns the position of the baseline of a text line—in pixels—measured from the top of the line's vertical extent (its cursor position).

The bases of all characters are aligned a specific number of pixels below the y cursor position, which is the top of the line on which the characters are printed. When a character is drawn, it is drawn below the cursor position by an amount equal to the difference between the number returned by the **:baseline** method and the baseline of the font of the character.

Another attribute of a font is the *character width*. The character width is the amount by which the cursor position should be moved to the right when a character is output to the window. Note that the character width does not necessarily correspond to the actual width of the bits of the character (although it usually does); it is simply defined to be the amount by which the cursor should be moved. Width can be an attribute either of the font as a whole or of each character separately.

Another attribute of each separate character is the *left kern*. Usually, its value is 0, but it can also be a positive or negative fixnum. When the window system draws a character at a given cursor position and the left kern is nonzero, then the character is drawn to the left of the cursor position by the amount of the left kern, instead of being drawn exactly at the cursor position. In other words, the cursor position is adjusted to the left by the amount of the left kern of a character when that character is drawn, but only temporarily; the left kern affects only where the single character is drawn and does not have any cumulative effect on the cursor position.

A font that does not have separate character widths for each character and does not have any nonzero left kerns is called a *fixed-width* font. The characters are all the same width, so they line up in columns, as in typewritten text. Other fonts are called *variable-width* because different characters have different widths, so characters do not line up in columns. Fixed-width fonts are typically used for programs where columnar indentation is used, while variable-width fonts are typically used for English text, because they tend to be easier to read and to take less space on the screen.

Each font also has attributes called the *blinker width* and *blinker height*. These are two nonnegative fixnums that tell the window system an appropriate width and height to make a rectangular blinker for characters in this font. These attributes are independent of other attributes and are used only for making blinkers. Typically, you should readjust the blinker width for each character in a variable-width font, making a wide blinker for wide characters and a narrow blinker for narrow characters. If you do not want to go to this trouble or do not necessarily know which character the blinker is on top of, you can use the font's blinker width as the width of your blinker. For a fixed-width font, the font's blinker width is always appropriate.

Each font has a *char-exists* table, which is an *art-1b* array with a value of 1 for each character that actually exists in the font and a value of 0 for other characters. This table is not used by the character-drawing software; it is used for information purposes. Characters that do not exist have pictures with no bits in them, exactly like the space character. Most fonts implement most of the printing characters in the character set, but some are missing some characters.

Displaying Fonts

9.7 You can use one of the following to display fonts:

- The font editor commands Select and Display. See the *Explorer Tools and Utilities* manual for more information about the font editor.
- The Zmacs command META-X List Fonts. When you execute this command, the Zmacs editor lists all the loaded fonts. You can then select a font and Zmacs displays a table of the selected font and a sample font. See the *Explorer Zmacs Editor Reference* manual for more information about this command.
- The `w:display-font` function. This function is used by the preceding commands to display the font.

`w:display-font font &key (:window w:selected-window) Function`
`:columns (:label-base 8) (:label-font fonts:cptfont)`
`(:sample-font fonts:cptfont) (:header-font fonts:h112b)`
`:mouse-sensitive-item-type :reset-cursor-p`

Displays a table showing *font* compared to *:sample-font* on *:window*.

Arguments *font* — The font to be displayed.

:window — The window in which the table is displayed.

:columns — The number of columns to use for the display. This value defaults to largest power of 2 that fits within the width of the window.

:label-base — The base to use for printing the label numbers.

:label-font — The font to use for printing the label numbers.

:sample-font — The font to print as a comparison to *font*.

:header-font — The font to use for printing the heading.

:mouse-sensitive-item-type — Whether the characters in the display are mouse-sensitive. If the value for this keyword is non-nil, the user can select a character with the mouse. This is typically useful only for the font editor.

:reset-cursor-p — Where to place the cursor on the display. If the value for this keyword is non-nil, the cursor is placed at 0,0 for **:window**. Otherwise, the cursor is placed at the end of the table.

For example, if you execute the following code in a Lisp Listener, it clears the display and produces a figure similar to the one shown:

```
(w:display-font fonts:courier)
```

		COURIER																																			
		0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37				
0	*	↓	α	β	^	~	ε	π	λ	ϑ	δ	†	±	•	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞			
40	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?						
100	⊖	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_					
140	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~	ƒ					
200																																					
240	ı	ç	€	£	¤	¥	¦	§	¨	©	ª	«	¬	®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿						
300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß					
340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ					

Format of Fonts

9.8 This paragraph explains the internal format in which fonts are represented. Most users do not need to know anything about this format. You can skip the discussion of format of fonts without loss of continuity.

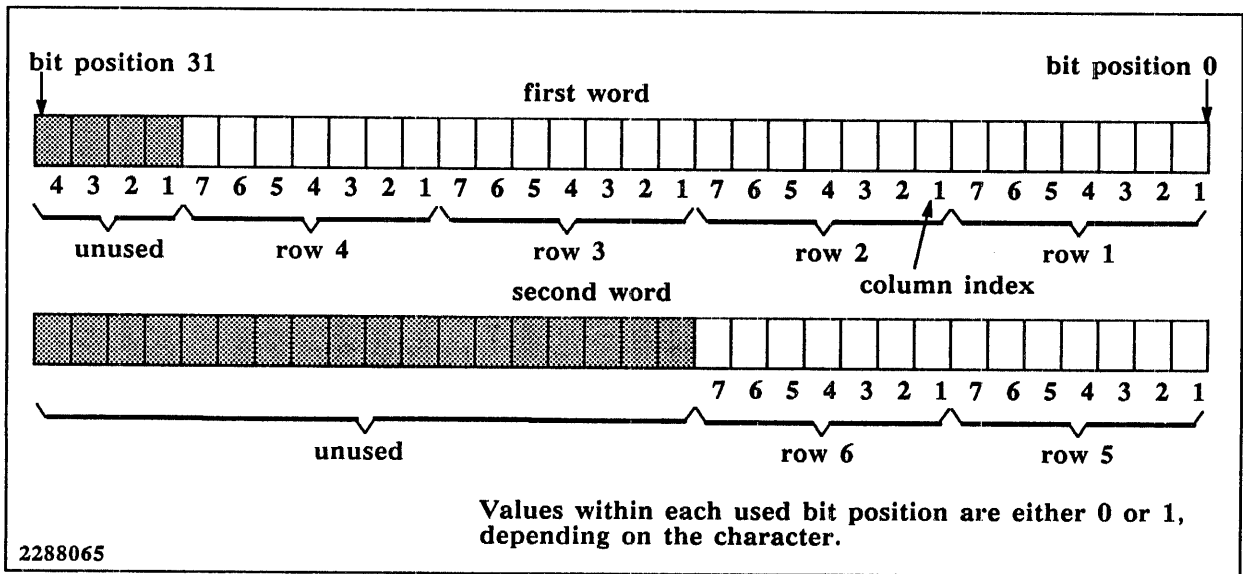
Fonts are represented as arrays. The body of the array holds the bits of the characters, and the array leader holds the attributes of the font and characters as well as information about the format of the body of the array. Note that one large array holds all the characters rather than a separate array for each character. The format in which the bits are stored is specially designed to maximize the speed of character drawing and to minimize the size of the data structure. Drawing speed is maximized by minimizing the number of words that need to be read from memory.

The font editor operates on a font by converting it into a different type of object containing the same data. This new object is called a *font descriptor*. See the files SYS:FONT-EDITOR;FNTDEF.LISP for the format of font descriptors, and SYS:FONT-EDITOR;FNTCNV.LISP for functions to operate on them and to convert between font descriptors and fonts.

If the font contains any characters that are wider than 32 bits, the font format works in a slightly different way. If such characters exist in a font, then the font is considered to be wide, and a single character can be made up of several subcharacters to be drawn side by side. A wide font stores subcharacters (instead of characters as such) and has a table indicating which subcharacters belong to each character of the character set. The following paragraphs discuss only narrow fonts in which there is no need to distinguish characters from subcharacters because each character is made of a single subcharacter.

Each character in a font can be thought of as having an array of bits stored for it. The dimensions of this array are called the *raster width* and *raster height*. The raster width and raster height are the same for every character of a font; they are properties of the font as a whole, not of each character separately. Consecutive rows are stored in the array; the number of rows per character is the raster height, and the number of bits per row is the raster width. An integral number of rows is stored in each word of the array. If there are any bits left over, those bits are unused. Thus, no row is ever split over a word boundary. When there are more rows than will fit into a word, the next word is used. Remaining bits to the left of the last word are ignored, and the next row is stored right-adjusted in the next word, and so on. An integral number of words is used for each character.

For example, consider a font in which the widest character is seven bits wide and the tallest character is six bits tall. The raster width of the font is seven and the raster height is six. Each row of a character is seven bits, so four of them fit into a 32-bit word, with four bits wasted, as shown in the following figure.



The remaining two rows require a second word, the rest of which are unused because the number of words per character must be an integer. Thus, this font has four rows per word and two words per character.

To find the bits for character 3 of the font, you multiply the character number—3—by the number of words per character—2—and find that the bits for character 3 start in word 6. The rightmost seven bits of word 6 are the first row of the character, the next seven bits are the second row, and so on. The rightmost seven bits of the seventh word are the fifth row, and the next seven bits of the seventh word are the sixth and last row.

Note the focus of this discussion is on *words* of the array. The character-drawing microcode does not actually care what type the array is; it only looks at machine words as a whole, unlike most of the array referencing in the Explorer. In an array that holds Lisp objects, such as an *art-q* array, the leftmost seven bits are not under control of the user, so these kinds of arrays are not suitable for fonts. In general, you need to be able to control the contents of every bit in the array, so fonts are usually *art-1b* arrays.

If any characters in the font are wider than 32 bits, then even a single row of the font does not fit into a word. Such characters are divided into subcharacters no more than 32 bits wide, and the character is drawn by drawing all of its subcharacters, one by one, side by side. The character-drawing microcode, which can handle only ordinary narrow characters, is invoked once for each subcharacter to draw a wide character. To make this process work, the wide font stores subcharacters in the same way a narrow font stores its characters.

In addition, the wide font has a *font indexing table*, which gives the first subcharacter number for each character code. In a narrow font, the font indexing table is *nil*. The character W is drawn by finding the value at index 87 (the character code for W) in the font indexing table, and the value at index 88. Suppose that these are 171 (for W) and 173 (for X). Then W is made up of two subcharacters: 171 and 172. Either of these subcharacters' bits can be found in the same way that the bits for character code 171 or 172 are found in a narrow font.

The array leader of a font is a structure defined by *defstruct*. The following functions access the elements of a font leader array.

w:font-name <i>font</i>	Function
Returns the name of <i>font</i> .	
w:font-baseline <i>font</i>	Function
Returns the baseline of <i>font</i> . The value returned is a nonnegative fixnum.	
w:font-blinker-width <i>font</i>	Function
w:font-blinker-height <i>font</i>	Function
Returns the blinker width or height, respectively, of <i>font</i> .	
w:font-char-height <i>font</i>	Function
Returns the character height of <i>font</i> . The value returned is a nonnegative fixnum.	
w:font-char-width <i>font</i>	Function
Returns the character width of <i>font</i> , which is typically the width of a lowercase m. The value returned is a nonnegative fixnum. If w:font-char-	

width-table of this font is non-nil, then this element is ignored except to compute the distance between horizontal tab stops.

w:font-char-width-table *font*

Function

Returns an array that contains the character width of each character of *font*. If this value is nil, then all the characters of the font have the same width (the width returned by **w:font-char-width**). Otherwise, this function returns an art-q array of nonnegative fixnums, one for each logical character of the font, giving the character width for that character. Note that the width of a character is actually the number of pixels that the cursor moves when the character is drawn, rather than the width of the actual character glyph.

w:font-left-kern-table *font*

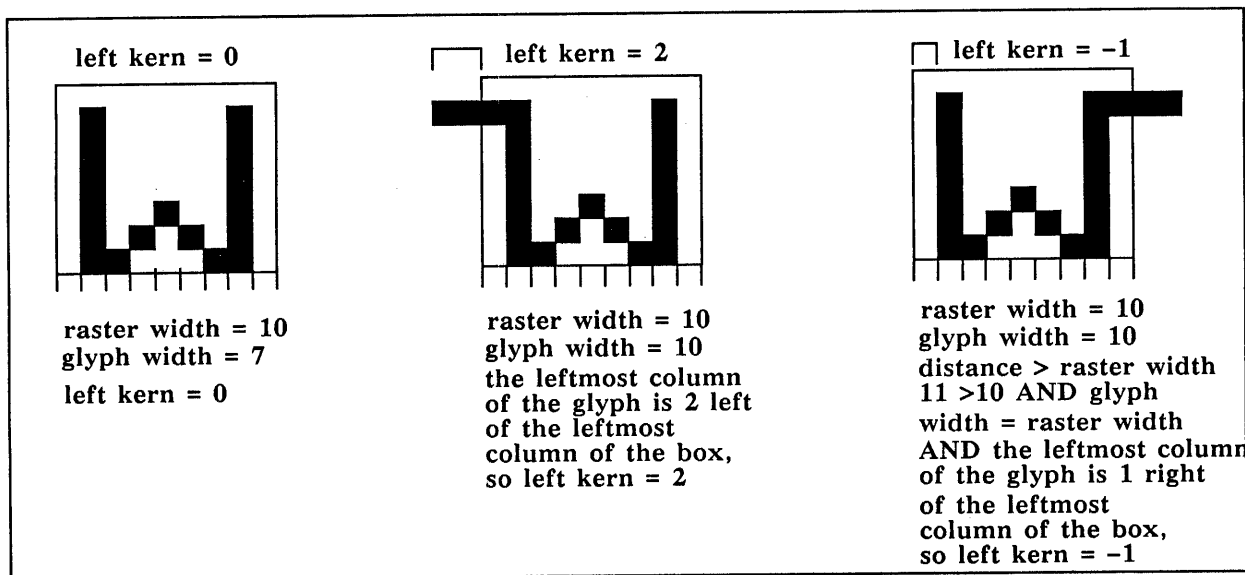
Function

Returns an array containing the kern of each character in *font*. If this value is nil, then all characters of the font have 0 left kern. Otherwise, this function returns an array of fixnums, one for each logical character of the font, giving the left kern for that character. A character has a positive left kern when any pixel of the character glyph extends outside the left edge of the character box. (The visual representation of width limits used in the font editor.) A character has a left negative kern when *both* of the following conditions are met:

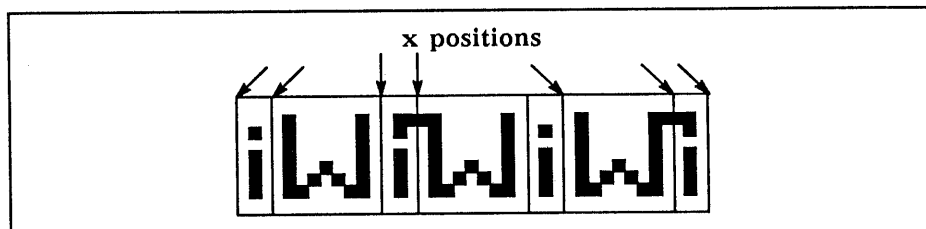
- The leftmost pixel of the glyph starts to the right of the leftmost column of the character box. That is, the leftmost column(s) of the character box are empty.
- The columnar distance from the rightmost pixel to the left edge of the character box is greater than the raster width *but* the glyph width is less than or equal to the raster width. In such a case, the character can still be stored in the allocated raster width. A negative left kern value tells the routines drawing the character to displace the output to the right, rather than to the left.

The left kern value is the number of pixels that the leftmost pixel of the glyph is displaced from the leftmost column of the character box. If the glyph width is ever wider than the raster width (which can occur when you are creating a font), the raster width is increased.

For example, consider the following characters:



A character that has left kerning is drawn, not at the current x position, but at the x position minus the kern value. After the character is drawn, the cursor moves to a new x position equal to the original x position plus the character width. The character width is taken from either the font char-width table, or, if the table is nil, from w:font-char-width. For example, consider the following characters shown drawn on the video display, along with their character boxes for reference:



Consider the characters as they would appear on the screen, with no extra reference marks:

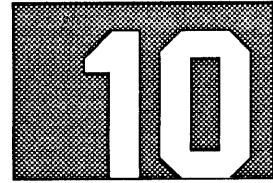


w:font-chars-exist-table *font*

Function

Returns an art-1b array with one element for each logical character of *font*. The value of the element is 1 if the character exists and 0 if the character does not exist.

- w:font-raster-height** *font* Function
w:font-raster-width *font* Function
Returns the raster height or width, respectively, of *font*. The returned value is a positive fixnum.
- w:font-rasters-per-word** *font* Function
Returns the number of rows of a character stored in each word of *font*. The returned value is a positive fixnum.
- w:font-words-per-char** *font* Function
Returns the number of words stored for each character or subcharacter in *font*. The returned value is a positive fixnum.
- w:font-indexing-table** *font* Function
Returns an array that is the indexing table of *font*. If this value is *nil*, then no characters of this font are wider than 32 bits. Otherwise, this is the font indexing table of *font*, an array indexed by character code that contains the number of the first subcharacter for that character code. An extra array element, referenced by an index number that is one greater than the largest character code, indicates where the subcharacters of the largest character code stop.



BLINKERS

Types of Blinkers

10.1 Like windows, blinkers are instances of flavors, but they are a different kind of flavor and support a different set of standard operations. The window system provides several kinds of blinkers, which differ in the way they appear on the screen.

Blinkers add visual cues to a window, but while programs are examining and altering the contents of a window, the blinkers all disappear. Before characters are output or graphics are drawn, the blinker is turned off; it comes back on later. Turning a blinker off is called *opening* the blinker. The `w:prepare-sheet` macro does this. You can see this happening with the *mouse blinker* when you move the mouse on the Explorer display.

To make this work, blinkers are drawn with an ALU operation that combines the blinker image's pixels with the screen's pixels in a reversible manner. That is, drawing the blinker image again restores the screen to its original appearance. On a monochrome system, this effect has been traditionally achieved by using the `w:alu-xor` ALU argument to draw the blinker the first time and `w:alu-xor` again the second time to restore the screen. On a color system, an equivalent effect can be achieved by first using `w:alu-add` for the first draw operation and `w:alu-sub` for the second draw operation. If `w:alu-add` and `w:alu-sub` are used on a monochrome system, the effect is the same as if `w:alu-xor` had been used both times. Thus, the most general way to draw blinkers is with `w:alu-add` and `w:alu-sub` because they exhibit the desired effect on both monochrome and color systems. (Table 12-2, ALU Values for Graphic Methods, lists several ALU values. For information on color ALUs, refer to paragraph 19.6, Color ALU Functions.)

Blinkers can be of many types and sizes. Some blinkers have geometric shapes, such as an arrow (mouse blinker), rectangle (cursor-following blinker), hollow rectangle (mouse blinker when positioned over a mouse-sensitive item), or any character of any font. The kind of blinker that you see most often is a blinking rectangle the same size as the characters you are typing. This blinker shows you the cursor position on the window. Another blinker often seen is the corresponding opening parenthesis when you are entering Lisp code while in the Lisp mode in the Zmacs editor. Blinkers do not always blink. For example, the *mouse arrow* does not blink at all. Mouse blinkers are discussed in detail in paragraph 11.7.

A window can have any number of blinkers on it at one time. They need not follow the cursor (some do and some do not). The ones that do are called *following* blinkers; the others have their position set by explicit operations.

In a color environment, you can set the color of the blinker as explained in paragraph 19.4, Initialization Options and Methods Used With Color Windows.

`w:blinker`

Flavor

Required methods: `:blink`, `:size`

The basic flavor for all blinkers.

w:make-blinker *window* &optional (*flavor* **w:rectangular-blinker**) Function
 &rest *options*

Creates and returns a new blinker.

Arguments: *window* — The window that the new blinker is to be associated with.
flavor — The flavor the blinker will be given. The default is **w:rectangular-blinker**. Other useful flavors of blinker are documented later in this section.
options — Initialization options to the blinker flavor. **w:make-blinker** accepts any initialization keywords and values that are accepted by **make-instance**. All blinkers include the **w:blinker** flavor, so initialization options taken by **w:blinker** work for any flavor of blinker. Other initialization options work only for particular flavors.

w:with-blinker-ready *do-not-open-p* *body* Macro

Used in writing methods of blinkers that change the size, position, shape, or anything else that affects how the blinker appears. This macro disables interrupts so that if the blinker is opened (that is, temporarily turned off), it remains open for the duration of the execution of *body*. Once the blinker is opened, its instance variables can be set.

Arguments: *do-not-open-p* — Determines whether the blinker should be opened first or not. If *do-not-open-p* is **nil**, the blinker is actually opened before *body* is executed. Specifying a non-**nil** value for *do-not-open-p* causes the blinker to remain on the screen in case the caller wishes to avoid updating the blinker if no change is needed. If a change is needed, *body* can call the **w:open-blinker** function to open the blinker.
body — Lisp code to be executed. **w:with-blinker-ready** executes *body* after preparing to remove the blinker **self** from the screen. This macro executes within a blinker method; therefore, **self** is the blinker instance.

Visibility and Deselected Visibility of Blinkers

10.2 Blinkers can have two modes: *visible* and *deselected visible*. The **:visibility** and **:deselected-visibility** initialization options describe the blinker's current visible attributes.

The blinker's visibility is controlled by its visibility and deselected visibility attributes combined with whether or not the window is selected. Usually, only the blinkers of the selected window actually blink, which shows where your keyboard input goes on the window.

While a blinker's current visibility is frequently changed by the program using the blinker, the deselected visibility is usually fixed and indicates something about how the blinker is generally used. When the window is deexposed, each blinker's visibility is initialized from its deselected visibility. When the window is selected, visibilities of **:on** or **:off** are changed to **:blink**. Blinkers whose visibility is **t**, **nil**, or **:blink** are not affected. These attributes are discussed with the **:visibility** and **:deselected-visibility** initialization options.

:visibility *visibility* Initialization Option of **w:blinker**
Gettable, settable. Default: **:blink**

Initializes the **w:visibility** instance variable to the current visibility of the blinker, which can be one of the following:

- **:blink** — The blinker blinks on and off periodically. The rate at which it blinks is called the *half-period* and is a fixnum giving the number of 60ths of a second between when the blinker turns on and when it turns off.
- **:on** or **t** — The blinker is visible but not blinking; it simply stays on.
- **:off** or **nil** — The blinker is invisible. When a blinker is invisible, it cannot be seen but is still in the window.

:deselected-visibility *visibility* Initialization Option of **w:blinker**
Gettable, settable. Default: **:on**

Initializes the **w:deselected-visibility** instance variable to the blinker's deselected visibility, which can be one of the following:

- **:on** — The blinker should be solid when deselected, blinking when selected. This is the most commonly used value and is the default for the blinkers that show the cursor position in a window.
- **:off** — The blinker should be off (that is, invisible) when deselected, and blinking when selected. When a blinker is invisible, it cannot be seen, but is still in the window.
- **:blink** — The blinker should be blinking whether selected or not.
- **t** — The blinker should be solid whether selected or not.
- **nil** — The blinker should be off whether selected or not.

:blinker-deselected-visibility *visibility* Initialization Option of *windows*
 Default: **:on**

Initializes the visibility of the window's cursor-following blinker. *visibility* specifies the blinker's visibility.

:half-period *half-period* Initialization Option of **w:blinker**
Gettable, settable. Default: 15.

Initializes the interval, in 60ths of a second between successive blinks of a blinker. **w:half-period** is specified in 60ths of a second. This instance variable is relevant only if the visibility is **:blink** or if the deselected visibility is **:on** or **:off**.

w:time-until-blink Instance Variable of **w:blinker**
 Default: 0.

The time interval in 60ths of a second until the next time this blinker should blink. For a blinking blinker, this instance variable controls the next turning on or off.

w:time-until-blink is set to the value of the **w:half-period** instance variable and is decremented by 1 every 60th of a second. When **w:time-until-blink** reaches 0, the blinker blinks and **w:time-until-blink** is reset to the value of **w:half-period**.

A nonblinking blinker does not necessarily change its state at the specified time; the blinker is checked at that time and displayed if the blinker is supposed to be visible but is not. This procedure is how blinkers reappear after being opened so that output can be done.

:defer-reappearance

Method of **w:blinker**

Is invoked whenever a blinker is opened to prepare a sheet if the visibility is not set to **:blink** and if the blinker is scheduled to reappear in less than 25/60ths of a second. By default, **:defer-reappearance** is defined to delay the blinker's reappearance until one-half second after the present.

w:open-blinker *blinker*

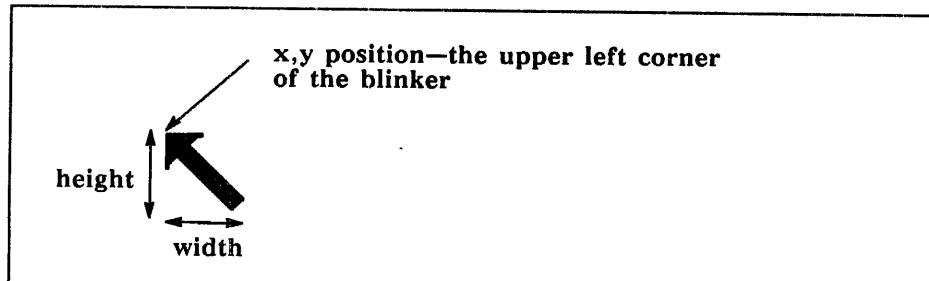
Function

Clears *blinker* from the screen if it is currently drawn, and then redraws it if it is supposed to be visible. Blinkers that are supposed to be visible but are not on the screen are occasionally put back on the screen by the scheduler. A blinker, then, can be relied on to stay open only as long as interrupts are disabled. Thus, the caller must turn off interrupts of the blinker should stay open during the execution of any specific piece of code.

Blinker Position

10.3 Every blinker is associated with a particular window. The blinker is displayed on this window so that its image can appear only within the window. When characters are output or graphics are drawn on a window, only the blinkers of that window and its ancestors are opened (because blinkers of other windows cannot occupy screen space that might overlap this output or graphics). The mouse blinker is free to move all over the screen it is on; it is therefore associated with the screen itself rather than a window and must be opened whenever anything is drawn on any window on the screen.

A blinker has a *position*, which gives the location of the blinker's upper left corner relative to the blinker's window. The blinker's lower right corner is controlled by the blinker's size together with its position. The blinker position must remain within the window's area. This restriction does not force the blinker's lower right corner to be within the window's area, but if it is not, the blinker's image is clipped, and the part outside the window does not appear.



w:sheet-following-blinker *window*

Function

Returns either a blinker that follows *window*'s cursor, or *nil* if *window* has no such blinker. If there is more than one blinker, this function returns the first one it finds. (Although a window can have more than one cursor-following blinker, having more than one would be confusing.)

- w:turn-off-sheet-blinkers** *window* Function
w:turn-on-sheet-blinkers *window* Function
- Sets the visibility of all blinkers on *window* to **:off** or **:on**, respectively. Blinkers in the window's inferior windows are not affected.
- w:turn-off-all-sheets-blinkers** *window* Function
- Turns off all the blinkers of *window*, including inferiors.
- w:get-visibility-of-all-sheets-blinkers** *window* Function
w:set-visibility-of-all-sheets-blinkers *window blinker-list-values* Function
- Returns or sets the visibility of all *window*'s blinkers, including inferiors. *blinker-list-values* is a list of values to be set. Each value in *blinker-list-values* corresponds to a blinker in *window*.
- w:open-all-sheets-blinkers** *window* Function
- Executes the **w:open-blinkers** function (described earlier in this section) for all the blinkers of *window*, including inferiors. **w:open-all-sheets-blinkers** should be done called within a **without-interrupts** macro to ensure that all blinkers are turned off when an operation completes execution.
- :sheet** *new-window* Initialization Option of **w:blinker**
Gettable, settable. Default: none
- Sets the window or screen on which this blinker appears.
- The **:set-sheet** method moves the blinker to another window specified by *new-window*. If the old window is an ancestor or descendant of *new-window*, **:set-sheet** adjusts the (relative) position of the blinker so that it does not appear to move. Otherwise, **:set-sheet** moves the blinker to the point (0,0) in the new window.
- :blinker-p** *t-or-nil* Initialization Option of *windows*
 Default: t
- Initializes the **w:blinker-list** instance variable to contain a cursor-following blinker for this window. *t-or-nil* specifies the blinker's visibility.
- In effect, **:blinker-p** creates one cursor-following blinker for a window; any other cursor-following blinkers you want for a window can be created manually in an **:init** method or elsewhere.
- :blinker-flavor** *flavor-name* Initialization Option of *windows*
 Default: **w:rectangular-blinker**
- Specifies a flavor for a cursor-following blinker.
- :x-pos** *x* Initialization Option of **w:blinker**
Gettable. Default: 0.
- :y-pos** *y* Initialization Option of **w:blinker**
Gettable. Default: 0.
- Sets the initial x or y position, respectively, of the blinker within the window. These options are irrelevant for blinkers that follow the cursor (the values would be nil). The initial position for nonfollowing blinkers defaults to the current cursor position, which defaults to (0,0).

- :read-cursorpos** Method of **w:blinker**
Returns two values—the *x* and *y* components of the position of the blinker that is inside the window.
- :set-cursorpos** *x y* Method of **w:blinker**
Sets the position of the blinker, relative to the inside of the window. If the blinker has been a cursor-following blinker, then it ceases to be one, and from this point on moves only when **:set-cursorpos** is invoked. *x* and *y* specify the *x* and *y* position within the window where the blinker position is to be set.
- After using the **:set-cursorpos** method, you should call the **:set-follow-p** method and specify a non-*nil* value for its argument to reset the blinker to follow the cursor. The preferred way to change the position of the following blinker of a window is to call the **:set-cursorpos** method for the window, not for the blinker.
- :follow-p** *t-or-nil* Initialization Option of **w:blinker**
Gettable, settable. Default: *nil*
Sets whether the blinker follows the cursor (*t*) or not (*nil*). The default is *nil*, so the blinker's position is set explicitly. If you specify *nil* for the argument to the **:set-follow-p** method, the blinker stops following the cursor and stays where it is until explicitly moved. Otherwise, the blinker starts following the cursor.
- :phase** Method of **w:blinker**
Returns the current phase of the blinker: *t* when the blinker is present on the screen, or *nil* when the blinker is not on the screen. The **w:phase** and **w:time-until-blink** instance variables determine if a blinker blinks.
- :blink** Method of **w:blinker**
Draws or erases the blinker. Because the blinker is always drawn using the **w:alu-xor**, drawing it and erasing it are usually exactly the same. The **:blink** method can examine the **w:phase** instance variable to determine the phase of a blinker, but usually there is no need to know. The **:blink** method can assume that the blinker's sheet is prepared for output. **:blink**, which is a required method of **w:blinker**, should always be called with interrupts disabled.

List of Blinkers

10.4 The **:blinker-list** method and **w:sheet-blinker-list** macro act on the blinkers listed in **w:blinker-list**.

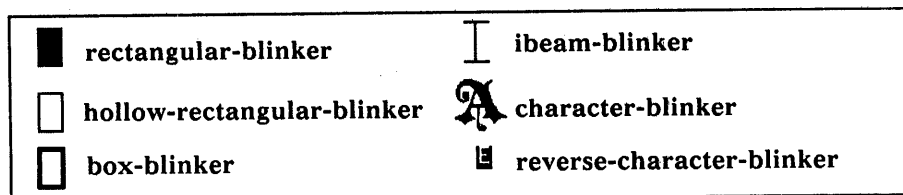
- :blinker-list** Method of *windows and screens*
w:sheet-blinker-list *window* Macro
The list of all blinkers associated with this window or screen. The macro accesses the instance variable for *window*.

Blinker Size 10.5

- :size** Method of **w:blinker**
 Returns two values indicating the width and height of the blinker area measured in pixels. Each flavor of blinker implements this method differently. **:size** is a required method for **w:blinker**.
- :set-size *new-width new-height*** Method of **w:blinker**
 Sets the size of the blinker's displayed pattern. Not all blinker flavors actually do anything, but they all allow this method. For example, character blinker size cannot change because there is no mechanism for automatically scaling fonts. The arguments specify the new width and height of the blinker, measured in pixels.
- :set-size-and-cursorpos *new-width new-height x y*** Method of **w:blinker**
 Sets the width and height of the blinker, in pixels as well as its position, all in one operation. This method prevents the blinker from appearing on the screen with its old size and new position, or vice versa.
- Arguments:* ***new-width, new-height*** — The blinker width and height, measured in pixels.
x, y — The window coordinates, relative to the inside of the window, where the blinker is to appear.

Rectangular and Character Blinkers

10.6 In addition to blinkers that follow the cursor, you can use rectangular, hollow rectangular, and character blinkers. The window system has flavors that use **w:blinker** to draw special blinkers on the window. You can specify the size of some of these blinkers. Some, especially those drawn using character fonts, have a fixed size.



The window system uses other flavors for the mouse blinker; these are discussed in paragraph 11.7, Mouse Blinkers.

- w:rectangular-blinker** Flavor
 Displays a solid rectangular blinker; this is the kind of blinker you see in Lisp Listeners and editor windows. You can use the **:set-size** method to control the width and height of the rectangle.
- :width *n-pixels*** Initialization Option of **w:rectangular-blinker**
 Default: the **font-blinker-width** of the first font in the font map of the associated window
- :height *n-pixels*** Initialization Option of **w:rectangular-blinker**
 Default: the **font-blinker-height** of the first font in the font map of the associated window
- Sets the initial width or height, respectively, of the blinker, measured in pixels.

:set-size *new-width new-height* Method of **w:rectangular-blinker**
 Sets the width and height of the blinker, measured in pixels.

:set-size-and-cursorpos *new-width new-height x y* Method of
w:rectangular-blinker
 Sets the width and height of the blinker, measured in pixels, and its x,y position.

w:hollow-rectangular-blinker Flavor
w:box-blinker Flavor

Displays a hollow rectangle. **w:hollow-rectangular-blinker** draws a box one pixel thick; **w:box-blinker** draws a box two pixels thick. These flavors include **w:rectangular-blinker**. The Zmacs editor uses such blinkers of flavor **w:hollow-rectangular-blinker** to show you which character the mouse is pointing at.

w:stay-inside-blinker-mixin Flavor
 Required flavor: **w:rectangular-blinker**

Keeps the corners of a rectangular blinker, or any modified version thereof, inside the blinker's window. Normally, a blinker flavor only makes sure that a blinker's position (its upper left corner) is within the window. **w:stay-inside-blinker-mixin** positions a blinker as close to the requested place as possible while keeping the entire blinker within the window.

w:ibeam-blinker Flavor

Displays a blinker that looks like an I beam (an uppercase I). Its height can be controlled using the **:height** initialization option. The vertical line is two pixels wide, and the two horizontal lines are nine pixels wide. An example of this kind of blinker is shown on the previous page.

:height *n-pixels* Initialization Option of **w:ibeam-blinker**
 Default: The line height of the window

Sets the initial height of the blinker. *n-pixels* specifies the blinker height, measured in pixels.

w:character-blinker Flavor

Draws a blinker that is a character from a font. You can control which font and which character within the font that **w:character-blinker** uses by supplying initialization options.

For example, the following code creates a blinker that is an A in the 43vxms font at the current cursor position. The blinker remains at that position until explicitly moved.

```
(w:make-blinker w:selected-window
  w:character-blinker
  :font fonts:43vxms
  :character #\A)
```

- :font *font*** Initialization Option of **w:character-blinker**
 Sets the font that **w:character-blinker** uses to draw the blinker. This font can be anything acceptable to the **:parse-font-specifier** method of the window's screen. You must provide the value for **w:character-blinker**. *font* specifies the font for **w:character-blinker** to use.
- :character *new-character*** Initialization Option of **w:character-blinker**
Gettable. Default: None; you must specify a character
- :set-character *new-character* &optional *new-font*** Method of **w:character-blinker**
 Sets the value for **w:character-blinker**, which you must provide (no default is provided). *new-character* is the character used to draw the blinker; *new-font* is the font used. *new-font* can be anything acceptable to the **:parse-font-specifier** method of the window's screen. If *new-font* is not specified, the font remains unchanged.

NOTE: For mouse blinkers, use the **w:mouse-set-blinker-definition** function instead of **:set-character**.

w:reverse-character-blinker Flavor

Creates a blinker as a solid rectangle with a character removed from it. That is, a solid rectangle and the character are both drawn and XORed with each other.

All the methods and initialization options of **w:character-blinker** are provided, though this flavor does not depend on **w:character-blinker**. This flavor includes **w:bitblt-blinker**.

The position of the blinker is at the upper left corner of the rectangle. The initialization options **:character-x-offset** and **:character-y-offset** specify the position of the upper left corner of the character with respect to the rectangle.

- :character-x-offset *n-pixels*** Initialization Option of **w:reverse-character-blinker**
Gettable.
- :character-y-offset *n-pixels*** Initialization Option of **w:reverse-character-blinker**
Gettable.

Specifies the x-offset or y-offset, respectively, of the character's upper left corner from the blinker's upper left corner. *n-pixels* specifies the amount of offset, measured in pixels.

For example, the following code creates a blinker that is an E in the **bigfnt** font at the current cursor position. The blinker remains at that position until explicitly moved.

```
(w:make-blinker w:selected-window 'w:reverse-character-blinker
      :font fonts:bigfnt
      :character #\E)
```

w:bitblt-blinker

Flavor

Displays a blinker by copying a two-dimensional array of pixels onto the screen. The array's size must be at least the size of the blinker. This flavor also has the ability to be the mouse blinker. You must specify an array, either with the **:array** initialization option or with both the **:height** and **:width** initialization options.

NOTE: Do not use **w:bitblt-blinker** to define the mouse blinker. The mouse blinker has a flavor with its own associated initialization options, methods, and instance variables.

:array arrayInitialization Option of **w:bitblt-blinker**

Gettable, settable. Default: none

Specifies the array of pixels to be used to display the blinker. You should use **w:make-sheet-bit-array** to create the array. If you do not specify this initialization option, you must specify both the **:height** and **:width** initialization options, which are used to create an empty array of that size. *array* is the array of pixels used to display this blinker.

:height n-pixels
:width n-pixelsInitialization Option of **w:bitblt-blinker**Initialization Option of **w:bitblt-blinker**

Sets the initial height or width of the blinker, respectively, in pixels. *n-pixels* is the number of pixels to which the dimension is set. If you do not specify the **:array** initialization option, you must specify both the **:height** and **:width** initialization options.

:sizeMethod of **w:bitblt-blinker**

Returns the width and height of the blinker. If these values are less than the size of the blinker's array, then only part of the array is used, starting at the upper left corner.

:set-size width heightMethod of **w:bitblt-blinker**

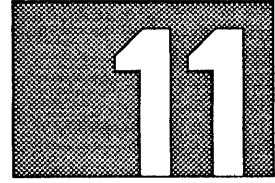
Sets the size of the blinker, making a new array if the old one is not as big as the new size. The arguments specify the new dimensions of the blinker, measured in pixels.

w:magnifying-blinker

Flavor

Provides a kind of bitblt blinker that automatically displays a magnified version of some of the dots underneath it. A small square of screen pixels is magnified by replacing each pixel with an *n* by *n* square of identical pixels, where *n* is the blinker's magnification factor. **w:magnifying-blinker** includes the **w:bitblt-blinker** flavor, so all initialization options and methods of **w:magnifying-blinker** can use **w:bitblt-blinker**.

The x-offset and y-offset, which the blinker has by virtue of **w:mouse-blinker-mixin**, help determine the center of magnification. The position of the magnifying blinker is, as always, the position of its upper left corner.



THE MOUSE

Using the Mouse

11.1 The mouse is an input device used by programs and windows. With the mouse, a user can move the cursor position and indicate choices from a menu or actions to be performed.

At any given time, the mouse is considered to be indicating a certain position on the screen, called the *mouse cursor position*. The mouse cursor is a conceptual entity that is regarded as what moves inside the machine when the user moves the mouse.

The mouse cursor position is indicated on the screen by a blinker called the *mouse blinker*, an actual Lisp object of the sort described in paragraph 10.1, Types of Blinkers. Different blinkers can be the mouse blinker at different times because each window can decide what to use as the mouse blinker when that window has control of the mouse.

There can be more than one screen, but the mouse cursor position is limited to one screen, called the *mouse sheet*. (It does not have to be a screen, but it normally is.) Mouse cursor positions are usually represented relative to the outside of the mouse sheet, though in operations on windows they are sometimes represented relative to the particular window.

The mouse cursor usually moves only if the user moves the mouse. The program, however, can move the mouse cursor and change the logical position of the mouse at any time. This is called *warping the mouse*. For example, double click left in the editor warps the mouse to where the editor cursor is currently located. Because there is no fixed association between positions of the physical mouse on the pad and screen positions, warping the mouse does not result in any inconsistency.

Tracking the mouse means examining the hardware mouse interface, noting how the mouse is moving, and adjusting the mouse cursor position and the mouse blinker accordingly. Mouse tracking is performed by either microcode (when the mouse is within a window) or by a process called the *mouse process* (when the mouse moves between windows). The mouse process also keeps track of which window has control of the mouse at any time. For example, when the mouse enters an editor window, the editor window becomes the owner, and to indicate this, the blinker becomes a northeast arrow. These changes are performed by the mouse process.

In general, the mouse process decides how to handle the mouse based on the flavor of the window that owns the mouse. Some flavors handle the mouse themselves, running in the mouse process, which enables them to put boxes (and the like) around things. Boxing an item usually indicates that clicking a button would affect the boxed item or perform some function determined by the boxed item. The effect of clicking the mouse buttons is also determined by the flavor of the window owning the mouse. The editor, the Inspector, menus, and other system facilities do this.

The functions, variables, and flavors described in this section allow you to use the mouse to do some simple things. This section also explains how to use advanced mouse behavior in your programs. The Zmacs editor uses the various functions, flavors, initialization options, and methods described in this section. Alternatively, you can invoke the built-in choice facilities, such as menus and multiple-choice windows; these high-level facilities are described in Section 14, Choice Facilities.

This section does not discuss how to create mouse-sensitive items. This is described in paragraph 14.5, Mouse-Sensitive Items.

Mouse Variables and Functions

11.2 The following variables and functions affect the position of the mouse on a window.

<code>sys:mouse-x</code>	Variable
<code>sys:mouse-y</code>	Variable
	The x or y position of the mouse measured, in pixels, from the outside upper left corner of the mouse sheet. The process handling the mouse—normally the mouse process—should maintain these variables. Note that the <code>w:mouse-input</code> function does not automatically maintain the <code>sys:mouse-x</code> and <code>sys:mouse-y</code> variables. You must add the code to maintain these variables if you want to track the cumulative mouse position.
<code>w:mouse-sheet</code>	Variable
	The mouse sheet, the one on which the mouse cursor moves.
<code>w:mouse-set-sheet window</code>	Function
	Sets <code>w:mouse-sheet</code> to <i>window</i> . Only inferiors of the mouse sheet (to any number of levels) can own the mouse.
<code>w:mouse-set-sheet-then-call window function &rest args</code>	Function
	Changes the value of the <code>w:mouse-sheet</code> variable to <i>window</i> , applies <i>function</i> to the variable, then resets the value of the <code>w:mouse-sheet</code> variable to the value it had before the <code>w:mouse-set-sheet-then-call</code> function was called. <i>args</i> are the argument(s) to be used by <i>function</i> .
<code>w:mouse-speed</code>	Variable
	Default: 0. The speed the mouse has been moving recently, in units approximating inches per second.
<code>w:mouse-warp x y &optional (relative nil)</code>	Function
	Warps the mouse to the positions specified by the <i>x</i> and <i>y</i> arguments. The <i>x</i> and <i>y</i> arguments are the respective outside <i>x</i> and <i>y</i> coordinates within the mouse sheet. If <i>relative</i> is <i>t</i> , <code>w:mouse-warp</code> moves the mouse relative to its current position. If <i>relative</i> is <i>nil</i> , <code>w:mouse-warp</code> moves the mouse to the position <i>x,y</i> .

w:mouse-fast-motion-speed	Variable
Default: 30. units approximating inches per second	
w:mouse-fast-motion-cross-size	Variable
Default: 40. pixels for each arm of the cross	
w:mouse-fast-motion-cross-time	Variable
Default: 2000. iterations of a do loop	
w:mouse-fast-motion-bitmap-time	Variable
Default: 16000. iterations of a do loop	
Variables that determine the behavior of the mouse when the user moves the mouse very quickly. Moving faster than w:mouse-fast-motion-speed causes the cursor-following blinker to change to either a cross that is w:mouse-fast-motion-cross-size in width, or a bitmap image, as determined by the value of w:mouse-fast-track-bitmap-mouse-p .	
Either the large cross or bitmap image appears while a do loop iterates the number of times specified by their respective variables (w:mouse-fast-motion-cross-time or w:mouse-fast-motion-bitmap-time).	
w:mouse-fast-track-bitmap-mouse-p	Variable
Default: nil, which means to use the default cross	
w:bitmap-mouse-pathname	Variable
Default: "SYS:WINDOW;GODZILLA-MOUSE.BITMAP"	
Specifies whether to use a bitmap image or a default large cross when the mouse is moved faster than w:mouse-fast-motion-speed . A value of nil means use the default cross; a non-nil value means use the bitmap image found in w:bitmap-mouse-pathname .	

Mouse Parameters 11.3 The following are used to tailor the mouse to the user's needs.

w:mouse-bounce-time	Variable
Default: 2000. microseconds	
The delay in microseconds that the system waits before reexamining the status of a mouse button. That is, when a mouse button is clicked, the system records the click and then waits the duration specified by w:mouse-bounce-time before examining the same mouse button again.	
w:mouse-double-click-time	Variable
Default: 200,000. microseconds (0.2 seconds)	
The delay in microseconds after which the system gives up checking for an additional mouse-click.	
w:mouse-discard-clickahead	Function
Clears out the microcode buffer in which the mouse-tracking microcode records mouse-clicks.	
w:use-kbd-buttons	Variable
Default: t	
Determines whether to interpret the LEFT, MIDDLE, and RIGHT keyboard keys as mouse clicks. A value of t specifies to treat input from those keys exactly as if the input was from mouse clicks.	

w:*mouse-incrementing-keystates*	Variable
Default: (:control :shift :hyper)	
A list of keys (valid arguments for w:key-state). When the mouse is clicked, each of these keys that is held down adds 1 to the number of clicks. Thus, if you perform a single click with the CTRL key pressed down, this single click is treated as a double click.	
w:mouse-handedness	Variable
Default: :right	
Determines whether the mouse is configured to be used by a left-handed or right-handed user. Possible values are :left and :right.	
Switching from one value to another automatically changes how the mouse reacts to clicks. For example, with the default value of :right, a user working in the Zmacs editor would double click right to invoke the System menu, click middle to mark a region, and click left to move the mouse cursor to point. With a value of :left, the clicks are mirrored—that is, the user would double click left to invoke the System menu, click middle to mark a region, and click right to move the mouse cursor to point.	
A user can set this variable in the Profile utility, or execute one of the following functions.	
w:setup-mouse-left-handed	Function
w:setup-mouse-right-handed	Function
Changes the mouse handling to be normal for a left-handed or right-handed person, respectively.	

Mouse Clicks

11.4 The system can interpret the signal produced when a user presses a mouse button as one of a button mask, a character, or a blip.

- Button masks can indicate any of the actual hardware conditions. For example, a button mask can indicate when a user has pressed two buttons simultaneously. Buttons masks are only used by low-level mouse handlers because they are not as obvious as the equivalent encoded character when placed in code.
- Encoded characters symbolize certain actual hardware conditions, and so are more obvious when placed in code than the equivalent button mask. For example, the mouse character #\mouse-L-1 is more obvious than its equivalent button mask (a value of 1).
- Blips are similar to encoded characters but include more information, such as the position of the mouse when the click occurred. Blips are discussed in general in paragraph 8.3, Blips. Specific blips are described with the methods that generate them.

Button Masks

11.4.1 A *button mask* is the internal representation of which mouse buttons are depressed. The numbers 1, 2, and 4 represent the left, middle, and right buttons, respectively, and the value of the mask is the sum of the numbers representing the buttons that were being held down. For example, a value of 5 indicates that both the left and right buttons were depressed.

w:mouse-last-buttons

Variable

Contains a mask describing the mouse buttons as they were the last time the process handling the mouse looked at them.

w:mouse-buttons &optional *peek*

Function

Returns four values:

1. The current state of the mouse buttons, in the format used by **w:mouse-last-buttons**, by examining the hardware mouse registers.
2. The time when the values item 1 were the true state of the buttons.
3. The x coordinate of the mouse at the time reported in item 2.
4. The y coordinate of the mouse at the time reported in item 2.

If *peek* is non-**nil**, the function examines and returns the state without removing the state from the input buffer. Processes other than the mouse process use this option. The value for *peek* defaults to **t** if it is called from the mouse process, and to **nil** otherwise. Typically, you should allow the function to default this argument.

NOTE: If you incorrectly specify *peek* as **nil**, subsequent mouse operations—even after you exit the process that calls **w:mouse-buttons**—may be interpreted incorrectly.

w:mouse-character-button-encode *buttons-down*

Function

Interprets pressing a mouse button as a click. This function monitors the mouse button and determines whether a single click or double click occurs; it returns either **nil** (if no button is pushed), or an encoded character describing the click (as explained in paragraph 11.4.2, Encoding Mouse Clicks as Characters). This function is used in the **:mouse-click** method of **w:essential-mouse-mixin**.

w:mouse-character-button-encode is called only when a button has just been pressed; that is, a button is pressed down that was not down before. You must pass *buttons-down*, which is a bit mask specifying which buttons were pressed, that is, which are down now that were not down before. The following form computes this mask:

```
(logand (logxor old-buttons -1) new-buttons)
```

where:

old-buttons is a mask of the buttons that were down before.

new-buttons is a mask of the buttons that are down now.

w:merge-shift-keys *char*

Function

Modifies *char* by setting the bits corresponding to all the shift keys currently pressed down on the keyboard and then returns *char*. The shift keys include **SHIFT**, **CTRL**, **META**, **HYPER**, **SUPER**, and **SYMBOL**.

This function is useful on the result returned by `w:mouse-character-button-encode` if you wish to record the state of the shift keys in the description of a mouse click so that the shift keys can alter the meaning of the click. For example, if a Zmacs buffer contains the form:

```
(w:merge-shift-keys #\A)
```

and you evaluate that form using the Evaluate Region command (that is, you mark the form and press CTRL-SHIFT-E), the system returns `#\c-A` because the CTRL key was depressed when the form was evaluated.

Encoding Mouse Clicks as Characters

11.4.2 Clicks on the mouse are sometimes *encoded* into characters. Such characters are normally forced into input buffers of windows, so they are distinguished from regular keyboard characters by having the mouse bit turned on. Encoding of clicks is done with the `w:mouse-character-button-encode` function (described previously).

Mouse clicks can also be done on the keyboard. See the variables `w:use-kbd-buttons` and `w:*mouse-incrementing-keystates*` in paragraph 11.3, Mouse Parameters.

The following subtle point explains some difficulties you may have with the `w:kbd-mouse-buttons-mixin` and `w:list-mouse-buttons-mixin` flavors. The characters (or blips) created by these two mixins go straight into the window's input buffer. Under some circumstances they may bypass pending characters that have been typed ahead at the keyboard. Thus, if you type some text and then select an item with the mouse in rapid succession while your program is busy, the program may see the mouse click before it sees the character from the keyboard. See paragraph 8.6.1, I/O Buffers and Type-Ahead, for further discussion of these issues.

The `w:kbd-mouse-buttons-mixin` and `w:list-mouse-buttons-mixin` flavors handle mouse clicks by forcing keyboard input describing the click.

`w:kbd-mouse-buttons-mixin`

Flavor

Required flavor: `w:essential-mouse`

Handles mouse clicks by encoding them as characters that are forced into the window's input buffer. Single clicking the left button on an unselected window selects that window; double clicking the right button calls the System menu. Otherwise, the encoded character representation of the click is forced into the input buffer of the window. (Specifically, the input is sent as a fixnum via `:force-kbd-input`; which button was clicked is stored by using the `w:mouse-buttons` function; the number of clicks is stored by using the `char-mouse-clicks` function.)

The state of the CTRL, META, SUPER, and HYPER keys is included in the bits attribute of the encoded character field. This state can be tested using the `w:key-state` function, described in paragraph 8.8, Querying the Keyboard Explicitly.

`w:list-mouse-buttons-mixin`

Flavor

Similar to `w:kbd-mouse-buttons-mixin`, except that a blip is placed in the input buffer rather than simply an encoded character. The blip has the following form:

```
(:mouse-button encoded-char window x y)
```

This blip is more useful than the encoded character: the blip tells you where the mouse was (relative to the outside part of the window) and which window the mouse was over (this is useful primarily if several windows are sharing the same input buffer).

The encoded character returned in the blip contains information about which mouse button was pressed, the number of clicks, and the state of the CTRL, META, SUPER, and HYPER keys when the mouse was pressed. You can retrieve this information by using the **char-mouse-buttons**, **char-mouse-clicks**, and **char-bits** functions. See the *Explorer Lisp Reference* manual for more information about these functions.

The state of the CTRL, META, SUPER, and HYPER keys is included in the bits attribute of the encoded character field. This state can be tested using the **w:key-state** function, described in paragraph 8.8, Querying the Keyboard Explicitly.

Ownership of the Mouse

11.5 The window that the mouse is positioned over usually handles the mouse; this window *owns the mouse*. The window that owns the mouse is the one that receives the **:handle-mouse**, **:mouse-moves**, and **:mouse-click** messages.

Because windows are arranged hierarchically, a window, its superior, its superior's superior, and so on, are all positioned under the mouse at the same time. So the window that owns the mouse is really the lowest window in the hierarchy that is visible (that is, the window farthest in the hierarchy from the screen that, along with all its ancestors, is exposed). If you move the window to a part of the screen occupied by a partially visible window, then one of its ancestors (often the screen itself) becomes the owner. The screen handles single clicking on the left button by selecting the window under it. This is why you can select partially visible windows with the mouse.

A window can be greedy and keep ownership of the mouse, even if the mouse moves outside of it, if the **w>window-owning-mouse** variable is set to that window. This variable should be used only when that window has gained ownership of the mouse by legitimate means, inside a **:handle-mouse** method on that window or one of the other methods invoked by the window. Inferiors of the greedy window can still own the mouse when it is over them. Greediness ends when **w>window-owning-mouse** is set back to nil (its normal state). Then mouse ownership reverts to whichever window is under the mouse. While a window is being greedy, mouse tracking continues to use the methods of the owning window, but the way of determining the owning window is changed.

The mouse can also be *grabbed*, which means that a process has taken it away from all windows. This state is represented by **w>window-owning-mouse** set to **t**.

Usurping the mouse is an even more drastic method of taking over control. It turns the mouse process off, so you have to do the tracking yourself. See paragraph 11.5.2, Usurping the Mouse.

w:window-owning-mouse Variable

Indicates whether the mouse is owned or grabbed, as follows:

Value	Explanation
nil	The mouse is owned by the window it is on.
t	The mouse is grabbed but the window is not able to identify itself.
A window	The mouse is owned by the window no matter where the mouse is.
:stop	Forces the mouse process to do nothing.

w:window-owning-mouse &optional x y Function

Returns the window that now owns the mouse, either because the window is being greedy or because the mouse is positioned over it. If *x* and *y* are specified, **w:window-owning-mouse** returns the window that would handle the mouse if the mouse was at *x,y*.

w:mouse-window Variable

Either the window that is currently handling the mouse, or **nil** if there is none. This value is returned by the **w:window-owning-mouse** function.

w:mouse-wakeup Function

Informs the mouse process that the screen layout has changed. Anything that can change the window that is under the mouse at any time should call this function.

w:hysteretic-window-mixin Flavor

Allows a window to continue owning the mouse (by being greedy) for a small distance beyond the edges of the window. This distance is called the *hysteresis*, and you can specify it using the **:hysteresis** initialization option. Pop-up menus use the **w:hysteretic-window-mixin** flavor so that if you accidentally move the mouse cursor a bit outside the menu, the menu remains exposed; you have to move the mouse cursor well away from the menu before the menu deexposes.

:hysteresis *hysteresis* Initialization Option of **w:hysteretic-window-mixin**
Gettable, settable. Default: 25.

Sets the value, in pixels, of the hysteresis.

Grabbing the Mouse

11.5.1 The window that owns the mouse usually interprets the mouse clicks and motion. Some applications, such as Edit Screen, use the mouse for choosing a window to operate on. It is then necessary to make sure that control of the mouse remains with the program that is interpreting the mouse clicks (for example, Edit Screen) rather than whatever window the user chooses. Control is accomplished by grabbing the mouse.

When the mouse is grabbed, the mouse process is told that no window owns the mouse, and the mouse process changes the mouse blinker back to the default (a northeast arrow). The mouse process continues to track the mouse, and your process can now watch the position and the buttons by using **sys:mouse-x** and **sys:mouse-y**, as well as the following variables and functions.

w:with-mouse-grabbed *body*

Macro

Contains a body of Lisp code; *body* is evaluated with the mouse grabbed. *body* should first set the mouse blinker with a function such as **w:mouse-standard-blinker**. *body* can wait for changes in the tracked position of the mouse by using **w:mouse-wait**.

The **w:with-mouse-grabbed** macro binds the **w:who-line-mouse-grabbed-documentation** variable to **nil**, which makes the mouse documentation window blank. You can change this variable to a string or a list using **setq** inside *body* (see **:who-line-documentation-string** in paragraph 11.6, How Windows Handle the Mouse, for a description of the list). The result is then displayed in the mouse documentation window. If your program has modes that affect how the mouse behaves, each part of the program should use this variable to display this documentation.

w:mouse-wait &optional (*old-mouse-x* **sys:mouse-x**) Function
 (*old-mouse-y* **sys:mouse-y**) (*old-mouse-buttons* **w:mouse-last-buttons**)
 (*whostate* "mouse")

Causes the process to wait until any of the variables **sys:mouse-x**, **sys:mouse-y**, or **w:mouse-last-buttons** change from the values passed of *old-mouse-x*, *old-mouse-y*, and *old-mouse-buttons*. *whostate* is displayed as the who-line status while the process is waiting. This function is used in processes other than the mouse process.

To avoid timing errors, your program should examine the values of these variables, use the values, and then pass them as arguments to **w:mouse-wait** before allowing the mouse to move again. It is important to do things in this order, or else you might fail to update one of the variables changed while you were using the old values and before you called **w:mouse-wait**. If you fail to update the variables, then the **w:mouse-wakeup** function cannot inform the mouse process that the mouse has been used.

w:mouse-wait binds the **w:who-line-mouse-grabbed-documentation** variable to **nil**, which makes the mouse documentation window blank.

w:who-line-mouse-grabbed-documentation

Variable

The text that appears in the mouse documentation window. When this variable is non-**nil**, it overrides all other sources of mouse documentation for the mouse documentation window.

You can change this variable to a string or a list using **setq** inside *body* (see **:who-line-documentation-string** in paragraph 11.6, How Windows Handle the Mouse, for a description of the list). The result is then displayed in the mouse documentation window. If your program has modes that affect how the mouse behaves, each part of the program should use this variable to display this documentation.

When grabbing or usurping the mouse, you should explain what is going on in the mouse documentation window at the bottom of the screen. The **with-mouse-grabbed** and **with-mouse-usurped** macros bind this variable to **nil**, which makes the mouse documentation window blank. Inside the body of one of these macros, you can change this variable to a string using **setq**; the result is then displayed in the mouse documentation window. If your program has modes that affect how the mouse behaves, each part of the program should use this variable in its own documentation.

w:window-under-mouse &optional *method* Function
 (*active-condition* :active) *x y*

Returns the visible window the mouse is positioned over.

If you specify values for *x* and *y* and if *x* and *y* are non-nil when **w:window-under-mouse** executes, the function returns the window at the screen coordinates specified by the *x* and *y* arguments.

If you do not specify values for *x* or if *y* and *x* and *y* are nil when **w:window-under-mouse** executes, the function returns the window under the mouse blinker.

The mouse process uses **w:window-under-mouse** to decide which window owns the mouse. You can also use this function when you grab the mouse.

Arguments: *method* — The method controlling the mouse. If *method* is non-nil, only windows that handle that method are considered.

active-condition — Whether to select active or exposed windows. If *active-condition* is :active, then **w:window-under-mouse** selects only active windows. If *active-condition* is :exposed, then **w:window-under-mouse** selects only exposed windows.

x, y — The *x* and *y* coordinates of the location of the mouse blinker.

The following functions are used in the screen editor to enable a user to position a window.

w:mouse-specify-rectangle &optional *left top right bottom* Function
 (*window w:mouse-sheet*) (*minimum-width* 0) (*minimum-height* 0)
 (*abortable* nil)

Grabs the mouse and allows the user to specify the upper left and lower right corners of a rectangle by using the mouse. **w:mouse-specify-rectangle** returns four values: the values of the left, top, right, and bottom of the rectangle, all relative to *window*.

The user can specify a corner of a rectangle by doing any of the following:

- Explicitly moving the mouse blinker to a location and clicking left.
- Accepting the default location by clicking left without moving the mouse. The default locations are either specified by the optional *left*, *top*, *right*, and *bottom* arguments. If these arguments are not specified, the upper left corner is specified by the current mouse position; the function assumes a nearby position for the lower right corner.
- Clicking right to allow the function to decide a location for the corner. Thus, for panes in a frame or other complex layouts, the system tries to fill the available space. Allowing the system to place the corner produces a layout that is precisely filled, that is, one whose panes are contiguous and not separated by a few pixels of space (which may happen if a user tries to explicitly place the panes with the mouse).

The user can also click middle to abort the operation if *abortable* is *t*. If the user specifies a rectangle that is zero or negative in size (for example, if the user places the lower right corner to the left or above the the location selected for the upper left corner), the function returns the entire main screen. This procedure is how the System menu Create option works.

Typically, you should use the `w:mouse-set-sheet-then-call` function to expose the window, and then call `w:mouse-specify-rectangle`.

Arguments: *left, top, right, bottom* — The initial x,y coordinates of the upper left and lower right corners of the rectangle. *left* and *top*, when non-`nil`, specify the upper left corner of the rectangle. *right* and *bottom*, when non-`nil`, specify the lower right corner of the rectangle. The updated values are returned by the `w:mouse-specify-rectangle` function.

window — The window to which *left, right, top, and bottom* are relative.

minimum-width, minimum-height — The constraints for the values that can be returned. The difference between the returned values of *left* and *right* cannot be less than the value of *minimum-width*. The same relationship holds for *minimum-height* and the difference between *top* and *bottom*.

abortable — Whether the execution can be aborted. Only if *abortable* is non-`nil` can the function be aborted. If this argument is non-`nil`, pressing the middle button on the mouse aborts the execution of this function.

`w:mouse-set-window-size` *window* &optional (*move-p t*) Function

Grabs the mouse and asks the user for new edges for *window*, returns the new edges, and (unless inhibited) sets the edges of *window* to the new edges as well. The edges of *window* are set unless *move-p* is `nil` or unless the user aborted.

The values returned are either the new left, top, right, and bottom edges, suitable for the `:set-edges` method, or `nil` if the user aborted.

`w:mouse-set-window-position` *window* &optional (*move-p t*) Function

Grabs the mouse and asks the user for a new position for *window*. If *move-p* is `t`, the window is moved to the position specified by the user. If *move-p* is `nil`, the window position is not changed. In both cases, `w:mouse-set-window-position` either returns the new x and y position of the upper left corner, suitable for the `:set-position` method, or returns `nil` if the user aborted.

Usurping the Mouse 11.5.2 For optimal real-time performance, you can *usurp* the mouse. Then the mouse process steps aside and lets you do everything related to tracking the mouse until you return control to it. The `sys:mouse-x` and `sys:mouse-y` variables are not updated while the mouse is usurped. The mouse blinker disappears, and if you want any visual indication of the mouse to appear, you must add the necessary code to display the mouse blinker.

`w:with-mouse-usurped` *body* Macro

Contains a body of Lisp code; *body* is evaluated while the mouse is usurped. You can track the mouse within *body* by using the `w:mouse-input` function.

`w:with-mouse-usurped` binds the variable `w:who-line-mouse-grabbed-documentation` to `nil`, which makes the mouse documentation window blank. Inside *body*, you can change this variable to a string or a list using `setq`

inside *body* (see `:who-line-documentation-string` in paragraph 11.6, How Windows Handle the Mouse, for a description of the list). The result is then displayed in the mouse documentation window. If your program has modes that affect how the mouse behaves, each part of the program should use this variable to display this documentation.

`w:mouse-input` &optional (*wait-flag* *t*) Function

Waits until something happens with the mouse; then it returns values that tell what happened and passes back the position of the mouse when the button was pressed. `w:mouse-input` returns six values:

- The first two values are delta-x and delta-y, which are the distance that the mouse has moved since the last time `w:mouse-input` was called.
- The second two values are `buttons-newly-pushed` and `buttons-newly-raised`, which are bit masks (using the bit assignment of the `w:mouse-last-buttons` function) that identify which buttons have been clicked since the last time `w:mouse-input` was called.
- The last two values are the x and y positions of the mouse.

You should call this function only when the mouse is usurped. Otherwise, you interfere with the mouse process, which also calls this function; then mouse tracking does not work correctly.

`w:mouse-input` does not maintain the `sys:mouse-x` and `sys:mouse-y` variables. You must add the code to maintain these variables if you want to track the cumulative mouse position. `w:mouse-input` function does maintain the `w:mouse-last-buttons` variable.

The `buttons-newly-pushed` value is suitable for being passed as an argument to the `w:mouse-character-buttons-encode` function, which can be used with the mouse usurped as well as with the mouse grabbed.

If *wait-flag* is `nil`, then the function does not wait; it may return with all zeroes, indicating that nothing has changed.

How Windows Handle the Mouse

11.6 The mouse is rarely grabbed or usurped. A window (or a screen) usually owns the mouse, in which case mouse handling works through various flavor methods of the owning window. There are several methods, used at various points in mouse handling, to give you convenient hooks for modifying a window's behavior.

The outermost loop of mouse handling determines the owning window and then invokes its `:handle-mouse` method. When this method returns, the owning window is recalculated.

`:handle-mouse`

Method of *windows*

Is invoked by the mouse process to handle the mouse while it is positioned on this window. `:handle-mouse` should return only when the mouse moves out of the window, or if the mouse is grabbed.

The default definition of this method calls the `w:mouse-standard-blinker` function followed by the `w:mouse-default-handler` function.

w:mouse-default-handler *window* &optional *scroll-bar-flag* Function

Is the core of the **:handle-mouse** method. **:handle-mouse** typically sets up the desired type of mouse blinker and then calls the **w:mouse-default-handler** function.

w:mouse-default-handler invokes the **:mouse-moves** method to inform the window about mouse motion and invokes the **:mouse-buttons** method to inform it about buttons that have been pressed. The **:mouse-moves** and **:mouse-buttons** methods are the most convenient hooks to use for implementing simple new mouse behaviors.

Arguments: *window* — The window for which the mouse is being handled.

scroll-bar-flag — A value that describes the position of the mouse in relation to the scrolling region. *scroll-bar-flag* can be one of the following:

Value	Mouse Position
:in	Already in the scroll bar. This value is used by the :handle-mouse-scroll method.
t	Entering the scroll bar.
nil	Not in the scroll bar.

:set-mouse-cursorpos *x y* Method of *windows*
:set-mouse-position *x y* Method of *windows*

Moves the mouse instantaneously to the specified position. The effect is as if the user had moved the mouse over to that spot without actually touching the mouse.

x and *y* specify the mouse's new position. For **:set-mouse-cursorpos**, these values are relative to the inside edges of the window (as in the **:set-cursorpos** method). For **:set-mouse-position**, the values are relative to the outside edges of the window.

:mouse-moves *x y* Method of *windows*

Is invoked in the mouse process every time the mouse moves either into, within, or out of this window. *x* and *y* are the current position of the mouse, relative to the outside edges of this window.

:mouse-moves should always call the **w:mouse-set-blinker-cursorpos** function to make the mouse blinker move. In addition, the **:mouse-moves** method frequently moves other blinkers or turns them on or off. This is how menus outline the item the mouse is over.

The **w:mouse-default-handler** function invokes the **:mouse-moves** method.

When this window ceases to own the mouse, for whatever reason, the **:mouse-moves** method is always called one final time, so that it can turn off extra blinkers, and so forth.

w:mouse-set-blinker-cursorpos &rest *ignore* Function

Moves the current mouse blinker to the current mouse position. The **:mouse-moves** method typically calls this function. The value specified for *ignore* is ignored.

:mouse-buttons *mask x y* Method of *windows*

Is invoked in the mouse process when a button is pressed. By default, **:mouse-buttons** calls **w:mouse-character-button-encode** to check for double clicks, then brings up the System menu for double click right; otherwise, **:mouse-buttons** invokes the **:mouse-click** method.

For examples of special uses for this method, see the **:mouse-buttons** methods defined for **w:scroll-bar-mixin** and for **w:menu**.

The **w:mouse-default-handler** function invokes **:mouse-buttons**.

Arguments: *mask* — A mask of the buttons pressed of the form acceptable to the **w:mouse-last-buttons** variable.

x, y — The mouse position (in the mouse sheet).

:mouse-click *mouse-char x y* Method of *windows*

Does most of the handling of mouse-clicks. **:mouse-click** is invoked in the mouse process.

Any window selection desired should be done in another process, using **process-run-function** or **w:mouse-select**.

The **:or** method combination is used so that all the methods are run until one of them returns non-*nil*. Thus, each mixin can define a way of handling the mouse under certain circumstances, and it can decline to handle the click by returning *nil*. For example, if the mouse is positioned inside a margin choice box, **w:margin-choice-mixin** defines a **:mouse-click** method that handles the click; otherwise, **w:margin-choice-mixin** returns *nil* so that the window's primary way of handling clicks can be run.

The **w:kbd-mouse-buttons-mixin** and **w:list-mouse-buttons-mixin** flavors work by defining **:mouse-click** methods.

Arguments: *mouse-char* — A character code describing the button pressed and how many times, such as **#\mouse-L-2**.

x,y — The position of the mouse at the beginning of the click. It is preferable to use this position rather than the current one because the user positions the mouse accurately before clicking, and mouse motion during the click is probably accidental.

:who-line-documentation-string Method of *windows*

Returns a string or list of items to be displayed in the mouse documentation window describing what happens if a mouse button is clicked while positioned over a window item. For example, menus are often defined to return a string that describes the menu item over which the mouse is positioned.

If the **:who-line-documentation-string** method returns a string, the string should describe what happens if a mouse button is clicked. If different buttons—or multiple clicks of a button—produce different results, the method should return a list rather than a string. Using a list enables the system to automatically switch the documentation for mouse clicks when the user selects a left-handed mouse.

If the **:who-line-documentation-string** method returns a list, the window system formats that list so that the items fit according to the amount of space currently available in the mouse documentation window. The returned list is

organized in keyword-value pairs. That is, the first, third, fifth, and so on, list items are keywords. The list item immediately following a keyword is the value for that keyword. The following keywords can be used:

- **:documentation** is a general documentation string displayed in the mouse documentation window.
- **:font** causes the following items in the list to be displayed in the specified font. The font is not required to be in the font map for the mouse documentation window.
- **:keystroke** is a keystroke associated with the command being documented. The value part of this keyword can be either a string or a character.
- Keywords for documentation used for clicking a specific button:

Keyword	Button and Number of Clicks
:mouse-L-1	Left mouse button once
:mouse-L-2	Left mouse button twice
:mouse-L-hold	Left mouse button held
:mouse-M-1	Middle mouse button once
:mouse-M-2	Middle mouse button twice
:mouse-M-hold	Middle mouse button held
:mouse-R-1	Right mouse button once
:mouse-R-2	Right mouse button twice
:mouse-R-hold	Right mouse button held

The number of lines available in the mouse documentation window controls how the documentation format is displayed.

- A keyword, **:mouse-any**, for documentation used for clicking any mouse button. Specifying this keyword causes the system to display L,M,R: followed by the documentation supplied.
- **:no-comma**, when specified, causes items in the mouse documentation window *not* to be delimited with commas. The argument for this keyword is a string that is used as the new delimiter. If you use the **:no-comma** keyword, you should do one of the following:
 - Specify **nil** for no delimiter.
 - Specify a string to provide a different delimiter (for example, a semicolon or period).
 - Explicitly format the strings. For example, the following code provides the mouse documentation for the scroll bar:

```
(defparameter *scroll-bar-who-line-documentation*
  `(:mouse-L-1 "This line to top"
    :mouse-L-hold "Continuous next line"
    :mouse-L-2 "Next page"
    :mouse-M-1 "To fraction of buffer"
    :mouse-M-hold "Drag lines"
    :mouse-R-1 "This line to bottom"
    :mouse-R-hold "Continuous previous line"
    :mouse-R-2 "Previous page.")
```

```

:no-comma      "  "
:documentation "  Bump top or bottom for single line scrolling.")
"The who-line-documentation-string when in the scroll-bar.")

```

```

L: This line to top      LH: Continuous next line      L2: Next page      M: To fraction of buffer  MH: Drag lines
H: Top line to here     HL: Continuous previous line  H2: Previous page.  Dump top or bottom for single line scrolling.
03/19/87 10:34:33AM WEBB      USER:      Keyboard      + Line: WEBB; IMAGE.HLDW3 0

```

w:mouse-select *window* Function

Selects *window*. This function is safe to use in the mouse process because **w:mouse-select** creates a temporary process to do the work. The **:mouse-click** method uses **w:mouse-select**.

w:mouse-call-system-menu &optional (*superior w:mouse-sheet*) Function

Brings up the System menu. This function is designed to be safe to use in the mouse process. The **:mouse-click** method calls **w:mouse-call-system-menu**.

Mouse Blinkers

11.7 At any given time, one blinker is the mouse blinker, which follows the motion of the mouse. It is not always the same blinker. Each window can create the kind of mouse blinker it wants or change the blinker, as long as that window owns the mouse.

The mouse blinker's sheet is the mouse sheet, not the window that owns the mouse and wants this blinker to be used. This arrangement avoids problems with displaying the blinker at points near the edge of the owning window such that parts of the blinker must be outside that window.

NOTE: Mouse blinkers track the position of the mouse; they do not follow cursor position (that is, their **:follow-p** attributes are **nil**). For example, the mouse blinker in the Zmacs editor is typically a northwest arrow. Its position is independent of the position of the cursor-following blinker, which are typically a solid rectangle. The Zmacs editor does offer functions that warp the mouse position to the cursor position and vice versa, so many users assume that cursor position and mouse position are related.

To make a window flavor use a particular mouse blinker, use the **:mouse-standard-blinker** method that alters the mouse blinker using **w:mouse-set-blinker** or **w:mouse-set-blinker-definition**.

Usually, only one form of mouse blinker is used for any given window. If you want the mouse blinker's appearance to vary while the mouse remains in the same window, a good technique is to have the **:mouse-standard-blinker** method know how to set up whichever blinker appearance is proper at the moment it is called, and then call **w:mouse-standard-blinker** after every event that might necessitate changing the blinker.

The **w:bitblt-blinker** and **w:magnifying-blinker** flavors are already suited to be mouse blinkers.

Variables and Functions **11.7.1**

w:mouse-blinker Variable

The blinker now following the mouse. This variable should not be changed by the user directly. A typical value for **w:mouse-blinker** is as follows:

```
#<MOUSE-CHARACTER-BLINKER 15600226>
```

w:mouse-set-blinker *type-or-blinker* &optional *x-offset* *y-offset* Function

Specifies a new mouse blinker. If *x-offset* and *y-offset* are non-nil, the pixel offsets of the blinker specified by the *type-or-blinker* argument are also set.

Typically, **:mouse-standard-blinker** methods call the **w:mouse-set-blinker** function.

Arguments: *type-or-blinker* — The new mouse blinker, which can be a defined blinker type instead of a blinker. In this case, the **w:mouse-set-blinker** is equivalent to the **w:mouse-set-blinker-definition** function with only three arguments specified.

x-offset, *y-offset* — The position of the mouse cursor relative to the mouse blinker. If *x-offset* and *y-offset* are non-nil, pixel offsets of the *type-or-blinker* are also set.

w:mouse-standard-blinker &optional (*window* (**w:window-owning-mouse**)) Function

Sets the mouse blinker to the standard blinker for *window* by invoking the **:mouse-standard-blinker** method on it. The function is called by the window system at appropriate times.

:mouse-standard-blinker Method of *windows*

Should use the **w:mouse-set-blinker** function or the **w:mouse-set-blinker-definition** function to set up the right kind of mouse blinker to use when the mouse is on this window. By default, the **:mouse-standard-blinker** method is defined to pass on the message to the superior window; finally, the screen handles the method by making the blinker a northwest arrow.

Flavors for Mouse Blinkers **11.7.2**

w:mouse-blinker-mixin Flavor
Required flavor: **w:blinker**

Makes a blinker suitable for use as the mouse blinker. Not all blinkers can serve as mouse blinkers.

:offsets Method of **w:mouse-blinker-mixin**

Returns two values: the x and y offsets of the blinker. The values give the position of the mouse cursor relative to the blinker; that is, to locate the cursor within the area of the blinker's display, the offsets must be positive.

:set-offsets *x* *y* Method of **w:mouse-blinker-mixin**

Sets the offsets of the blinker to the values specified by *x* and *y*.

A mouse blinker has two offsets that relate the blinker position to the mouse position. Remember that the blinker position is where the upper left corner of the blinker is displayed. The upper left corner is not always the part of the blinker you want to place at the precise position the mouse is pointing to. For example, if you are using a character blinker with the character x, probably the center of the x, rather than its upper left corner, should point to the specified position.

The following flavors produce blinkers of the specified type. These flavors have the same methods and initialization options as the analogous non-mouse blinkers. See paragraph 10.6, Rectangular and Character Blinkers, for a description of character blinkers.

- | | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| w:mouse-character-blinker | Flavor |
| Produces a blinker that is an actual character. | |
| w:mouse-rectangular-blinker | Flavor |
| Produces a blinker that is a solid rectangular blinker. | |
| w:mouse-hollow-rectangular-blinker | Flavor |
| Produces a hollow rectangular blinker having a border one pixel thick. | |
| w:mouse-box-blinker | Flavor |
| Produces a blinker that is a hollow box having a border two pixels thick. | |
| w:mouse-box-stay-inside-blinker | Flavor |
| Produces a rectangular blinker—or any version of a rectangular blinker. The w:mouse-box-stay-inside-blinker flavor ensures that the entire blinker remains within the window. Normally, a blinker flavor only ensures that its upper left corner remains within the window by not drawing the portion of the blinker that falls outside the window. | |

**Reusable Mouse
Blinker Types**

11.7.3 You do not usually create mouse blinkers yourself. Instead, each screen keeps a list of mouse blinkers of various sorts, and you reuse one of them. This is done by means of *mouse blinker type keywords*. A mouse blinker type keyword is given a meaning, which is a function for creating a blinker. The first time a user wants a blinker of that type on a given screen, one is created and remembered, and it is reused every time a blinker of that type is wanted. A blinker type keyword serves a purpose similar to that of a resource.

Predefined type keywords include **:character-blinker**, **:rectangle-blinker**, **:box-blinker** and **:box-stay-inside-blinker**. These keywords refer to their respective flavors. For example, the **:character-blinker** keyword refers to the **w:character-blinker** flavor. You do not have to use this mechanism, but using it saves creation of blinkers.

- | | | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|----------|
| w:mouse-define-blinker-type | <i>type creation-function</i> | Function |
| Defines as a mouse blinker type the blinkers specified by the <i>type</i> argument. <i>creation-function</i> specifies the function to use to create the blinker. This <i>creation-function</i> must be defined to receive a screen as an argument and to call w:make-blinker to actually create the blinker. | | |

w:mouse-get-blinker *type* &optional (*window* **w:mouse-sheet**) Function

Returns a blinker of *type* whose window is *window*. The same blinker is automatically reused for different windows on the same screen. In fact, the blinker's window is the screen, not the *window* argument. *type* should be a keyword of **w:mouse-blinker-type**.

w:mouse-set-blinker-definition *type* *x-offset* *y-offset* *visibility* Function
method &rest *args*

Sets the mouse blinker to the specified type, offsets, and visibility, then sends the blinker a message. For example, the **:set-character** method is useful with character blinkers.

The **:mouse-standard-blinker** method of a window can use the **w:mouse-set-blinker-definition** to specify a different appearance of the mouse blinker while the mouse is in that window.

Arguments: *type* — The type of blinker to use for the mouse blinker, listed in the previous discussion of reusable mouse blinker types.

x-offset, y-offset — The x and y offsets from the blinker's upper left corner, the part of the blinker that points to an item.

visibility — The blinker's visibility, which can be **t**, **:on**, **:off**, or **:blink**.

method — Either **nil** or a method name. The *method* argument is typically used to initialize aspects of the blinker other than *type*, *x-offset*, *y-offset*, and *visibility*.

args — Arguments used by *method* if *method* is non-**nil**.

w:mouse-blinkers Instance Variable of **w:screen**

A list of mouse blinkers (which are examples of various reusable mouse blinker types) that have been created for this screen. **w:prepare-sheet** turns off the blinkers in this list during drawing.

Mapping Mouse Characters

11.7.4 If desired, you can change the glyph associated with a standard mouse character. (The standard mouse characters, or *glyphs*, are described in paragraph 11.7.5, Standard Values for Mouse Characters.) For example, instead of using a double-headed thick arrow to indicate scrolling, you could use a mouse-shaped glyph or a diamond. To restore the mouse characters to their original forms requires a single function call.

To change the glyphs used for standard purposes, you must do two things. First, you must have an alternate mapping for the glyphs. Next, you must put this mapping in effect.

w:define-mouse-char-mapping *map-keyword* *new-mapping* Function

Defines a new mapping of mouse characters so that the user can change the appearance of the mouse. The mapping can use either the mouse font or a different font. This function performs error checking to ensure that the new font is compatible with the mouse. The mapping is stored as the **:mouse-char-mapping** property on *map-keyword*. The mappings are processed so that the characters are fixnums and the font is a symbol. Note that this mapping assumes that the offsets of the mouse cursor from the blinker glyph remain the same.

Arguments: *map-keyword* — A keyword used later to refer to this mapping of characters.
new-mapping — The list of the mapping. A mapping is of the form:

(mouse-character-descriptor new-glyph-descriptor optional-font)

where:

mouse-character-descriptor is either a keyword or a character object that indicates the mouse character being mapped, such as `:north-west-arrow` or `#\A`.

new-glyph-descriptor is either a keyword or a character object for the glyph to replace the mouse character, such as `:north-west-arrow` or `#\A`.

optional-font is an optional element that indicates the font for *new-glyph-descriptor*. If *optional-font* is not specified, `fonts:mouse` is used.

The keywords for *mouse-character-descriptor* and for *new-glyph-descriptor* come from the `w:mouse-name-to-position-map` constant; each keyword names the mouse character at a specific position. You should use the keyword for characters in the mouse font rather than its corresponding character object because the character object has no relation to the mouse character. For example, the north-west-arrow glyph in the mouse font is the `#\epsilon` character object.

Suppose you want the northwest mouse cursor, such as used in the Lisp Listener, to be a `w:mouse-glyph-hollow-box-pointer`. The following code sets up the mapping and associates it with the keyword `:hollow-box`. Wherever the northwest or the northeast mouse cursors had been used, the hollow box pointer will be used after you remap the mouse characters.

```
(w:define-mouse-char-mapping :hollow-box
  (list (list w:mouse-glyph-north-west-arrow
             w:mouse-glyph-hollow-box-pointer)
        (list w:mouse-glyph-north-east-arrow
             w:mouse-glyph-hollow-box-pointer)))
```

w:remap-mouse &rest *map-keywords*

Function

Remaps mouse characters to contain other glyphs. *map-keywords* is either `nil` to restore the mouse to its original mappings, or it is one or more keywords created by a `w:define-mouse-char-mapping` function. The remapping is performed for each keyword, left to right, in the order specified in the mapping. Thus, if you use several mappings that redefine the same character more than once, only the last mapping for that character is in effect. You can call `w:remap-mouse` repeatedly to change the character mappings.

NOTE: The mapping does *no* take effect until you leave an application or utility and then reenter it.

For example, this code implements the `:hollow-box` mapping defined in the previous example.























```
(w:remap-mouse :hollow-box)
```






















Standard Values for Mouse Characters

11.7.5 The `w:mouse-name-to-position-map` constant contains the standard mappings of mouse glyphs and characters. Normally you should use the keywords described in this constant rather than the character object or number to make the mappings easier to understand.













Glyph	Hex Value	Name Appended to w:mouse-glyph-	Glyph	Hex Value	Name Appended to w:mouse-glyph-
	0	-thin-up-arrow		18	Reserved
	1	-thin-right-arrow		19	-north-east-arrow
	2	-thin-down-arrow		1A	-circle-x
	3	-thin-left-arrow		1B	Reserved
	4	-thin-up-down-arrow		1C	-large-right-triangle-pointer
	5	-thin-left-right-arrow		1D	-medium-right-triangle-pointer
	6	-north-west-arrow		1E	-small-right-triangle-pointer
	7	-thin-cross		1F	-block-up-arrow
	8	-thick-up-arrow		20	-small-diamond
	9	-thick-right-arrow		21	-block-down-arrow
	A	-thick-down-arrow		22	-hollow-box-pointer
	B	-thick-left-arrow		23	-solid-box-pointer
	C	-thick-up-down-arrow		24	-hollow-circle-pointer
	10	-paragraph		25	-solid-circle-pointer
	11	-upper-left-corner		26	-thick-hollow-cross
	12	-lower-right-corner		27	-block-letter-t
	13	-hourglass		28	-hand-pointing-left
	14	-circle-plus		29	-double-up-arrow
	15	-paint-brush		2A	-hollow-arc-pointer
	16	-scissor		2B	-solid-arc-pointer
	17	-trident		2C	-spline-pointer

Continued

(Continued)		
Glyph	Hex Value	Name Appended to w:mouse-glyph-
	2D	-medium-diamond
	2E	-hollow-triangle-pointer
	2F	-solid-triangle-pointer
	30	-curtain
	31	-scale
	32	-6-3-arc-pointer*
	33	-9-6-arc-pointer*
	34	-3-12-arc-pointer*
	35	-12-9-arc-pointer*
	36	-ruler
	37	-polyline
	38	-double-up-arrow-lettered-d
	39	-thick-up-arrow-lettered-d
	3A	-thick-line-pointer
	3B	-question-mark
	3C	-thin-hollow-cross
	3D	-eye-glasses
	3E	-thin-hollow-plus
	3F	-rectangle-dots
	41	-west-rat-on-bottom
	42	-west-rat-on-top
	43	-north-rat-on-left

(Continued)		
Glyph	Hex Value	Name Appended to w:mouse-glyph-
	44	-north-rat-on-right
	45	-east-rat-on-bottom
	46	-east-rat-on-top
	47	-south-rat-on-left
	48	-south-rat-on-right
	49	-west-curly-rat-on-bottom
	4A	-west-curly-rat-on-top
	4B	-north-curly-rat-on-left
	4C	-north-curly-rat-on-right
	4D	-east-curly-rat-on-bottom
	4E	-east-curly-rat-on-top
	4F	-south-curly-rat-on-left
	50	-south-curly-rat-on-right
	51	-west-mouse-on-bottom
	52	-west-mouse-on-top
	53	-north-mouse-on-left
	54	-north-mouse-on-right
	55	-east-mouse-on-bottom
	56	-east-mouse-on-top
	57	-south-mouse-on-left
	58	-south-mouse-on-right
	59	Reserved

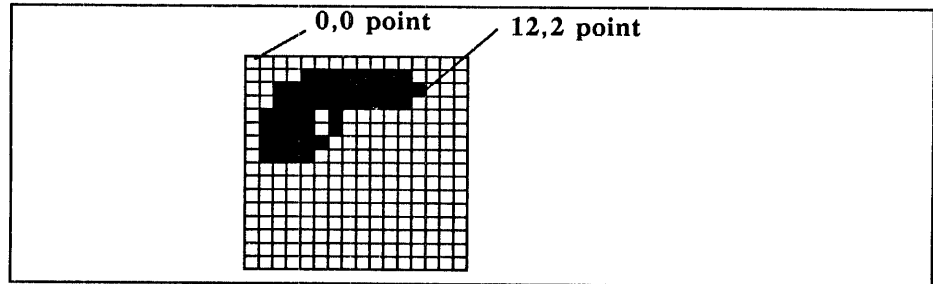
* The numbers indicate the hour positions of a clock.

(Continued)			(Continued)		
Glyph	Hex Value	Name Appended to w:mouse-glyph-	Glyph	Hex Value	Name Appended to w:mouse-glyph-
.	60	-small-dot		6A	-short-thin-down-border-arrow
	61	-thick-cross		6B	-short-thin-down-arrow
	62	-small-solid-circle		6C	-short-thin-up-border-arrow
	63	-medium-solid-circle		6D	-short-thin-up-arrow
	64	-hollow-circle		6E	-small-up-triangle
	65	-hollow-circle-minus		75	-small-down-triangle
	66	-hollow-circle-plus			

Creating a New Glyph 11.7.6 To create an entirely new glyph, follow these steps:

1. Using the font editor, create and store the new glyph in a font that has the same font height as the mouse font.
2. If you intend to reassign the offsets, compute the offsets of the new glyph from the upper left corner of the character box. Assuming the upper left pixel is the 0,0 point of a window coordinate system, obtain the x,y coordinates of the point that should be the mouse position when this glyph is used as the mouse blinker.
3. Assign a name to the glyph.
4. Use the `w:define-mouse-char-mapping` function to define a new mapping.
5. Use the `w:remap-mouse` function to remap the mouse.
6. If you want to reset the offsets for this particular blinker, define a special `:mouse-standard-blinker` method for each window that uses this blinker.

For example, suppose you create a right-pointing pistol and you want the muzzle of the pistol to indicate the cursor position. First, you create the glyph and save it in the p position of the my-mouse font using the font editor (see the *Explorer Tools and Utilities* manual for an explanation of how to use the font editor). If you want to create only one or two glyphs you could save them in the unassigned spaces in the mouse font. However, if you have several glyphs, such as a pistol pointing in each direction, you should create a new font so the new glyphs are easy to find and update.



The offsets for the cursor position would be 12,2.

Next, you assign a name to the glyph:

```
(defconstant w:mouse-glyph-pistol (charint #\p))
```

Define a new mapping where the pistol takes the place of both the northwest arrow and the northeast arrow:

```
(w:define-mouse-char-mapping :my-glyphs
  '(:north-west-arrow w:mouse-glyph-pistol fonts:my-font)
  (:north-east-arrow w:mouse-glyph-pistol fonts:my-font))
```

Actually remap the mouse characters:

```
(w:remap-mouse :my-glyphs)
```

Then, because the offsets are different for the new glyph, define a special **:mouse-standard-blinker** method for each window that uses this blinker. In this case, suppose that you want to use this method the flavor called **my-flavor**.

```
(defmethod (:mouse-standard-blinker my-flavor) ()
  (w:mouse-set-blinker-definition :character 12 3 :on
    :set-character w:mouse-glyph-pistol
    'fonts:mouse))
```

If you are setting up a new application that includes several blinkers in one window that have different offsets, set up a data structure to hold each of the blinker offsets for each of the glyphs in the font you are using.

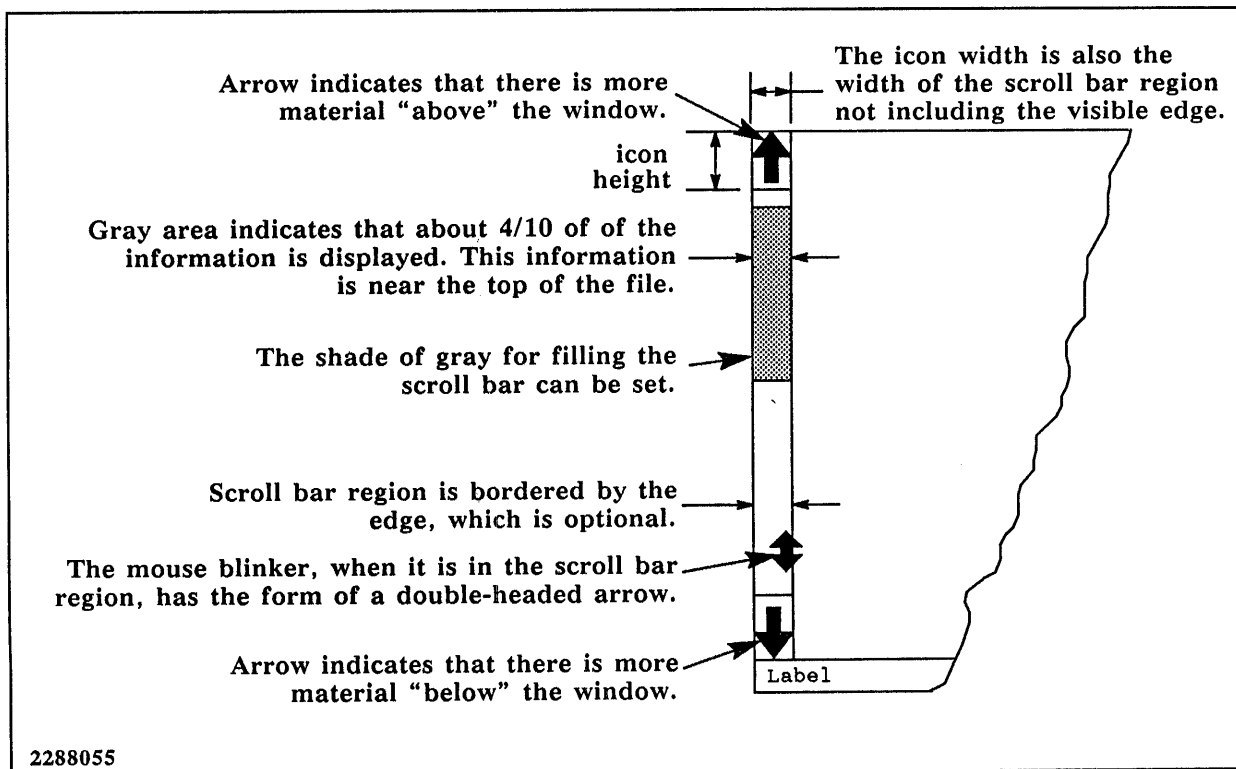
Mouse Scrolling

11.8 Some windows have the ability to *scroll*. They display only a portion of a virtual window that is (or may be) too big to be shown all at once. Scrolling means moving the portion of the window actually shown through the entire display. The window can be moved either up or down.

NOTE: Previous Explorer releases included several ways the mouse could be used to scroll a window, each implemented by a mixin. In Release 3.0, many of these special scrolling mixins have been combined into a single, simpler mixin that gives a better user interface, a better programmer interface, and better code. The Release 1.0 code is still available for use and is described in Appendix A, *Obsolete Symbols*; however, code marked as obsolete in this manual will be phased out in future releases.

Scroll Bars 11.8.1 The *scroll bar* is a graphic device used to indicate when more material is available than can be displayed at one time in a window. The scroll bar is displayed on either the left or right side of a window between the border and the border margin (the pixels of white space that separate the border from text). By default, a visible edge appears around the scroll bar region, which indicates where the mouse must be to execute scrolling with mouse clicks. When the mouse is in the scroll bar region, the mouse blinker is a double-headed arrow.

The scroll bar region can contain a more above icon (▲), a more below icon (▼), and a vertical bar. Unlike the **More above** and **More below** labels available in previous releases, the icons take up no vertical space in a window. Two lines are no longer needed to display labels. Thus, in windows such as Inspector panes, an application program can use all the vertical space to display information. The scroll bar is shown as a partially filled bar, where the length of the entire bar represents all the information and the shaded portion of the bar represents the length of the displayed information.



While the mouse is in the scroll bar, mouse clicks have the following special meanings. Note that the left, middle, and right clicks are the same for both Release 3 and previous releases. The special meanings for the other clicks have changed.

Mouse Click	Description
Left	This line to top
Left hold	Continuous next line
Double left	Next page
Middle	To fraction of buffer
Middle hold	Drag lines
Right	Top line to here
Right hold	Continuous previous line
Double right	Previous page

In addition to these clicks, the user can scroll the display one line at a time by positioning the mouse cursor over one of the icons and bumping the cursor against the line that the icon is pointing at.

A display mode determines what to display when more information is available above or below the buffer. In `:medium` mode (the default), one or both icons are displayed automatically, as appropriate (that is, the more above icon appears if there is more information above the window; the more below icon appears if there is more information below the window). The scroll bar is displayed only if you *bump* the mouse against the edge of the window, at which time the scroll bar appears. In `:maximum` mode, the scroll bar is displayed if either icon is displayed. In `:minimum` mode, the scroll bar and icons are displayed only after you bump the mouse against the edge of the window. If the mode is `nil`, the scroll bar is completely disabled, freeing the margin region space.

The mechanism of bumping the mouse against the window edge to display the scroll bar has an undesirable side effect: the user cannot easily move the mouse past that edge of the window. This difficulty can become important in the right-hand pane of a constraint frame, such as the Text area of the Glossary utility. To avoid the problem, you should display the scroll bar on the right side of the window instead of the left.

w:scroll-bar-mixin

Flavor

Provides a scroll bar and enables scrolling by using the mouse. This flavor requires `w:essential-window` and `w:borders-mixin` as component flavors, as well as requiring the methods `:scroll-to`, `:scroll-position`, and `:new-scroll-position` to be defined as for text scroll windows.

You can turn off the icons only by setting `w:scroll-bar-icon-height` to 0. You can completely disable the scroll bar by setting `scroll-bar-mode` to `nil`.

:scroll-bar-side *side*

Initialization Option of `w:scroll-bar-mixin`

Default: `w:*scroll-bar-default-side*`, initially `:left`

Determines which side of the window to display the scroll bar. Possible values are `:left` or `:right`.

- :scroll-bar** Method of **w:scroll-bar-mixin**
 Either returns **t** if there is a scroll bar on the left side of the window and there is something to scroll, or returns **nil**. This method is compatible with the Release 1.0 scroll bar mixins.
- :scroll-bar-on-right** Method of **w:scroll-bar-mixin**
 Either returns **t** if there is a scroll bar on the right side of the window and there is something to scroll, or returns **nil**.
- :handle-mouse-scroll** Method of **w:scroll-bar-mixin**
 Handles bumping with the mouse to display the scroll bar. The method changes the blinker to the character specified by **w:*scroll-bar-char-index***, changes the active state of the scroll bar, and then recursively calls the default mouse handler with a parameter of **:in**. This parameter specifies that the scroll bar should attempt to keep the mouse for a set distance after the user moves the mouse outside the scroll bar region. **:handle-mouse-scroll** is called from the default mouse handler when the scroll bar is invoked by bumping.
- w:*scroll-bar-char-index*** Variable
 Default: **w:mouse-glyph-thick-up-down-arrow**
 The index to the character in **fonts:mouse** that is used for the mouse when the mouse enters the scroll bar region.
- w:*scroll-bar-char-x-offset*** Variable
 Default: 6., which is the value used for the default character defined by **w:*scroll-bar-char-index***
- w:*scroll-bar-char-y-offset*** Variable
 Default: 6., which is the value used for the default character defined by **w:*scroll-bar-char-index***
 The x or y offset, respectively, for the mouse character blinker used by the scroll bar.
- :scroll-bar-icon-width** Method of **w:scroll-bar-mixin**
Settable. Default: **w:*scroll-bar-default-icon-width***, initially 7 pixels
 Returns the width of the icons in pixels, which is also the width of the scroll bar. The width does not include the pixels (3 by default) used to draw the box and the region edge border line around the scroll bar.
- :scroll-bar-icon-height** Method of **w:scroll-bar-mixin**
Settable. Default: **w:*scroll-bar-default-icon-height***, initially 9 pixels
 Returns the height of the icons. Setting the variable to 0 disables the icons.
- w:*scroll-bar-shade*** Variable
 Default: **w:50%-gray**
 The default shade to use to fill the highlighted area of the scroll bar. You can set this variable using **setq**. Any of the system-defined shades of gray discussed with the **w:make-gray** function in paragraph 12.4.2, Bit Block Transferring, are acceptable.

:scroll-bar-draw-edge-p Method of **w:scroll-bar-mixin**
Settable. Default: nil, do not draw the edges

Returns whether to draw (t) or not draw (nil) the edge of the scroll bar region.

:scroll-bar-mode Method of **w:scroll-bar-mixin**
Settable. Default: **w:*scroll-bar-default-mode***, initially **:medium**

Returns a value that specifies when to display the scroll bar and/or the arrow icons. Possible values are:

- **nil** — Permanently disable the scroll bar and free its margin region space.
- **:minimum** — Display the scroll bar and the icons only when the mouse is moved into the area of the scroll bar, that is, when you *bump* the mouse.
- **:medium** — If there is more material above the window, display the more above icon; if there is more material below the window, display the more below icon. Display the scroll bar only when it is bumped.
- **:maximum** — If there is more material above the window, display the more above icon; if there is more material below the window, display the more below icon. If either of the icons is displayed, also display the scroll bar. This mode may not be as fast as **:medium** because calculating the length of the scroll bar requires more time.

:scroll-bar-on-off Method of **w:scroll-bar-mixin**
Settable. Default: nil, which means to ignore this feature

Returns a flag that specifies whether to enable the **:decide-if-scrolling-necessary** method. If this flag is nil, scrolling is either always on or always off. If this flag is t, the **:decide-if-scrolling-necessary** method is enabled, which allows this window to enable or disable its scroll bar as needed. To decide whether to enable the scroll bar, use **:decide-if-scrolling-necessary** at the appropriate time, typically after setting the item list for a window.

:decide-if-scrolling-necessary Method of **w:scroll-bar-mixin**

Turns the scroll-bar regions on or off. This method should be called after changing the number of displayable items but before doing the redisplay. This method can change the inside size of the window unless the **:adjustable-size-p** method has been defined and returns t. If **:adjustable-size-p** returns t, you should set the outside size of the window before calling **:decide-if-scrolling-necessary**. Although the **:adjustable-size-p** method is not defined by the **w:scroll-bar-mixin** flavor, the method is used by this flavor when the method *is* defined.

w:*scroll-bar-default-clicks* Variable

Defines the methods to call for mouse clicks when the scroll bar is active. If you should want to change the default action of the mouse clicks for scrolling, you need to modify the methods listed in this variable or call different methods. Default actions are set up to give a consistent interface throughout the Explorer system, and typically you should not change them. If you do change them, you should also change the default documentation in the mouse documentation window that explains the actions.

w:*scroll-bar-who-line-documentation*

Variable

The default string that appears in the mouse documentation window when the mouse is in the scroll bar region, as shown in the following:

```

L: This line to top      LH: Continuous next line    L2: Next page      M: To fraction of buffer  MH: Drag lines
N: Top line to here    NH: Continuous previous line  N2: Previous page.  Bump top or bottom for single line scrolling.
08/19/87 10:04:03AM WEBB      USER: Keyboard      + Line: WEBB; IMAGE.XLW3 0

```

The following variables control how the mouse speed interacts with the scroll bar. These variables can be set by the **:handle-mouse** method.

w:scroll-bar-max-speed

Variable

Default: 7. units approximating inches per second

Minimum speed at which a user can move the mouse across a scroll bar. If the user moves the mouse into the scroll bar area at a speed less than **w:scroll-bar-max-speed**, the mouse stops against the scroll bar edge and does not move into the next area.

w:scroll-bar-max-exit-speed

Variable

Default: nil

Speed at which you leave the scroll bar.

w:scroll-bar-reluctance

Variable

Default: 1. pixels

The number of pixels the mouse must move beyond the inside edge of the window (that is, in the scrolling area) before scrolling is invoked.

Scrolling Protocol

11.8.2 You must define the following three methods for use by **w:scroll-bar-mixin**. Typically, you can use the definitions provided by text scroll windows for these methods, described in paragraph 16.2, Specifying the Item List.

```

:scroll-position
:scroll-to
:new-scroll-position

```

Methods Retained for Compatibility

11.8.3 These methods exist for compatibility with previous releases. They are not necessarily efficient.

```

:scroll-more-above
:scroll-more-below

```

Method of **w:scroll-bar-mixin**
Method of **w:scroll-bar-mixin**

Returns **t** if there is text to scroll up or down to, respectively. The default definitions use the **:scroll-position** method; some flavors redefine **:scroll-more-above** or **:scroll-more-below** for greater efficiency.

```

:scroll-relative from to

```

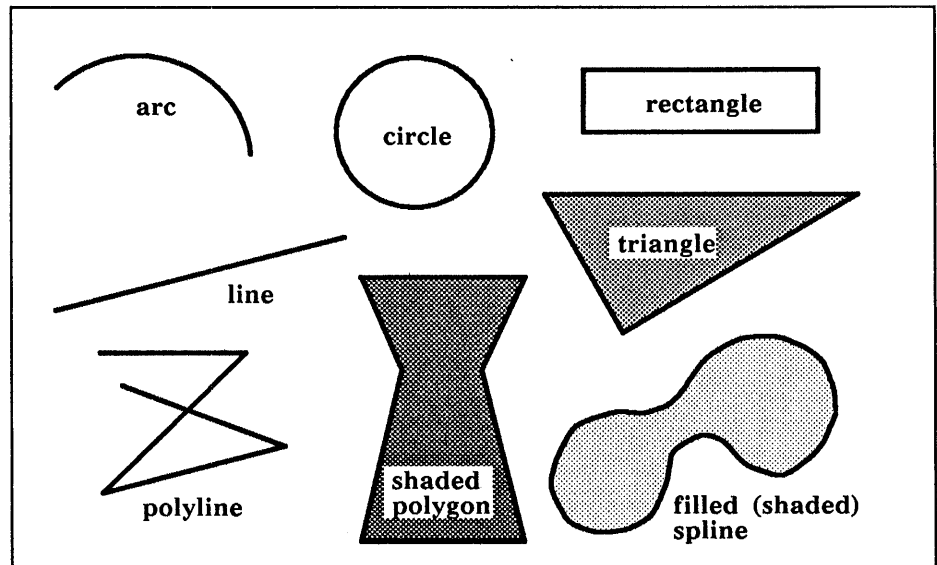
Method of **w:scroll-bar-mixin**

Scrolls the window to move what is now at the *y* position specified by the *from* argument to the *y* position specified by the *to* argument. The arguments can be numeric vertical cursor positions, or the symbols **:top** or **:bottom**. The **:scroll-position** and **:scroll-to** methods are used to accomplish the scrolling.

- :enable-scrolling-p** Method of **w:scroll-bar-mixin**
Returns *t* when the window has a scroll bar in active mode and there is something to scroll. This method is compatible with the old method.
- For the sake of the scrolling, the contents of the window are considered to be divided vertically into lines. A position for scrolling is expressed as the number of lines that are above the top of the window or below the bottom of the window. These do not have to be actual lines of text—though usually they are—but they must all have the same height. Usually this common height is the window's line height, but it need not be.
- :scroll-bar-lines** Method of **w:scroll-bar-mixin**
Settable. Default: **w:*scroll-bar-default-lines***, initially 1 line
Returns the number of lines to scroll each iteration when doing continuous scrolling. Although larger numbers may be faster in some cases, the effect is a jerky rather than a smooth movement of text.
- :scroll-bar-delay-time** Method of **w:scroll-bar-mixin**
Settable. Default: **w:*scroll-bar-default-delay-time***, initially zero 60ths of a second
Returns how many 60ths of a second to wait each iteration when doing continuous scrolling. This delay is needed by utilities that must switch processes to perform scrolling, such as the Zmacs editor.

Introduction

12.1 The window system features the ability to draw graphics on the video display. The simple graphics that you can draw are shown in the following figure.



In general, graphics are of two types:

- **Graphic images** — Images are drawn on the video display. When the user or a program executes a clear screen, or when the screen is scrolled up, the images disappear. To create the images again, the code must again be executed to create the images.
- **Graphic objects** — Objects, once created, are both drawn on the video display and exist in a graphics database known as a *world*. Thus, when the user or a program executes a clear screen, the image of the object disappears, but the object itself still exists in memory. To display the object again, you draw it rather than recreating the object.

The type of graphics you use for a particular application depends on the application. If your application requires only an image, then you should use the simpler graphic image methods, functions, and primitives. This code is in the W and the SYS packages. These methods execute somewhat faster and require less overhead than the graphic object methods, and you should use them when you need to draw the image only once.

If your application requires that you be able to store and retrieve the image, edit the image, or dynamically change the image, you should use graphic objects. The flavors and methods that implement graphic objects are in the GWIN package.

This section discusses the graphics capabilities of the window system and the graphics window system. This section *does not* attempt to explain the concepts behind graphics displays. For the theory of graphics and graphic displays, see a standard text on graphics such as *Fundamentals of Interactive Computer Graphics* by Foley and Van Dam (ISBN 0-201-14468-9; first published 1982, reprinted with corrections 1984, at Reading, Massachusetts; Addison-Wesley Publishing Company).

ALU Arguments

12.2 Most graphics methods, whether they draw graphic images or create graphic objects, use an arithmetic logic unit (ALU) argument. The ALU argument controls how the graphic being drawn intersects with images already on the window. In most cases, the ALU argument is optional; if no ALU argument is used, the method uses a default ALU argument, **w:normal** (which is **w:alu-seta** on both monochrome and color systems). Graphics methods support inclusive-or, exclusive-or, and, and-with-complement, set-all, and set-zero. On a color display, additional ALUs are available.

An ALU uses two arguments to control how a pixel is drawn on a screen. The two arguments are the *value* of the pixel being drawn and the *corresponding pixel* on the screen. On a monochrome display, the pixel is either on or off; combining the two arguments according to Boolean algebra determines whether to draw the pixel on the screen. Various logical combinations are available, as discussed in the following paragraphs.

On a color display, the pixel has an integer value that represents its color; combining two arguments is no longer a binary operation. For a discussion of color ALUs, refer to paragraph 19.6, Color ALU Functions.

General ALU Arguments 12.2.1 In most cases, instead of specifying a particular ALU argument, you use the following general purpose instance variables to specify how to draw or erase objects on the display.

w:char-aluf Instance Variable of *windows*
w:sheet-char-aluf *window* Function

Default: In a monochrome environment, the default is **w:alu-transp**, which functions exactly like **w:alu-ior** in that it merges the object being drawn with the existing object on the window. That is, **w:alu-ior** turns the pixel on if either input value is on; otherwise, the pixel remains off. In a color environment, the default is also **w:alu-transp**, which combines source and destination bits according to the rules of transparency. Refer to paragraph 19.6, Color ALU Functions, for details on transparency.

w:erase-aluf Instance Variable of *windows*
w:sheet-erase-aluf *window* Function

Default: In a monochrome environment, the default is **w:alu-back**, which functions exactly like **w:alu-andca** in that it turns off all bits in the destination. In a color environment, the default is also **w:alu-back**, which makes all bits in the destination the background color.

The typical ALU arguments used for drawing or erasing on a window, respectively. The **:tyo**, **:string-out**, and other methods use these instance variables. Other methods that you define to draw on or erase on a window should also use **w:char-aluf** and **w:erase-aluf** because these instance variables ensure that the output works properly for both standard and reverse video and also on both monochrome and color systems.

The macros access the respective instance variables, which you can set using **setf**.













When using ALU arguments, the most general approach is to use **w:combine** (**w:alu-transp**) and **w:erase** (**w:alu-back**), whether you are using a monochrome or a color system. **w:alu-transp** and **w:alu-back** behave as if they were **w:alu-ior** and **w:alu-andca** when used on a monochrome system, but **w:alu-ior** and **w:alu-andca** do not behave as if they were **w:alu-transp** and **w:alu-back** on a color system. Therefore, your application is upward compatible from monochrome to color when **w:alu-transp** and **w:alu-back** are used.

Whenever possible, the **w:char-aluf** and **w:erase-aluf** instance variables should be used instead of **w:combine** and **w:erase**. (Like **w:combine**, the value of **w:char-aluf** is **w:alu-transp** on both color and monochrome systems; like **w:erase**, the value of **w:erase-aluf** is **w:alu-back** on both color and monochrome systems.) If you do not use the **w:char-aluf** and **w:erase-aluf** instance variables, you should use symbolic names, such as **w:combine**, instead of logical names, such as **w:alu-transp**, because this will help in porting applications to color.







Refer to Table 12-2, ALU Values for Graphic Methods, for a discussion on **w:combine**, **w:erase**, **w:alu-transp**, and **w:alu-back**.

Available Monochrome ALU Arguments

12.2.2 The ALU arguments determine whether a pixel is turned on or off when an image is drawn. The following table discusses the effects of each ALU argument in a monochrome environment. In the center column, the first square shows the input that is drawn on the screen; the second square shows what the screen looks like to begin with; the third square shows the effects of the ALU arguments on the display.

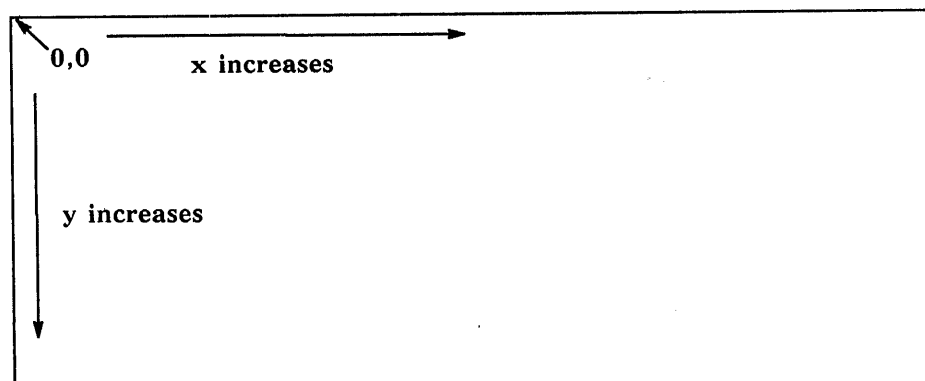
ALU Argument and Expanded Name	Input, Screen, Result	Description
w:alu-seta Set-all	  	Copies the object being drawn without regard to the existing image on the window.
w:alu-ior (w:alu-transp) Inclusive-or	  	Merges the object being drawn with the existing object on the window. w:alu-ior turns pixels on if either input value is on; otherwise, the pixel remains off. It is recommended that you use w:alu-transp in place of w:alu-ior in your applications regardless of whether you are using a monochrome or color system because this will help in porting applications from monochrome to color.
w:alu-and And	  	Turns on a pixel on the window only if the pixel is on in both the object being drawn and the object on the screen; otherwise, w:alu-and turns off the pixels at the window coordinates of the object being drawn.
w:alu-xor (w:alu-add) Exclusive-or	  	Turns on pixels if only one of the input values is on; otherwise, w:alu-xor turns the pixel off. A common use of w:alu-xor is to erase an object after it has been drawn on the screen. For example, when the mouse blinker moves from one position to another on the window, the blinker is first redrawn in the position it occupies using w:alu-xor (this erases the blinker), and the blinker is then drawn at the next position. It is recommended that you use w:alu-add in place of w:alu-xor in your applications regardless of whether you are using a monochrome or color system because this will help in porting applications from monochrome to color.

Continued

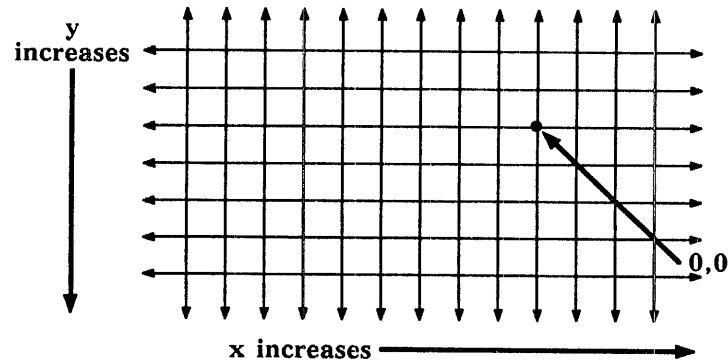
ALU Argument and Expanded Name	Input, Screen, Result	Description
w:alu-andca (w:alu-back) And-with-complement	  	Turns pixels off if the input is on; otherwise, w:alu-andca has no effect on the pixels. This is the w:erase-aluf for most windows. w:alu-andca is useful for erasing areas on the window, particular characters, or graphics. It is recommended that you use w:alu-back in place of w:alu-andca in your applications regardless of whether you are using a monochrome or color system because this will help in porting applications from monochrome to color.
w:alu-setz Set-to-zero	  	Turns off any pixel on the window.

Windows and Worlds

12.3 Graphic images are drawn on the video display using either window coordinates or world coordinates, depending on the functions or methods used to draw the image. *Window coordinates* are pixel locations on the video display, with the origin of the window coordinate system being the outside upper left corner of the main screen.



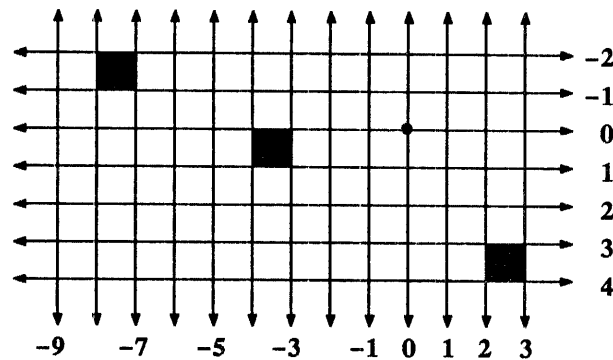
World coordinates, on the other hand, refer to an infinite, two-dimensional plane that can contain graphic objects. World coordinates are arbitrary units, such as inches, feet, centimeters, foobars, and so on, determined by the programmer or user. The *orientation* of the world determines the direction the units increase. World orientation is the same as window orientation. The *precision* of a world determines whether the dimensions of an object must consist of whole-unit increments or whether the dimensions can be in smaller units. (Basically, whether all points are specified by integers or whether you can use floating-point numbers.) For example, consider the following representation of a world with its origin slightly off-center. Its orientation is the same as the orientation of a window.



The contents of a world are contained in a *picture list*, sometimes called a *display list*. This picture list is simply a list of all the objects in the world.

To display the contents of a world, you use a window. A window gives a limited view of a world; it literally provides a window in which you can see the contents of the world. Each window is associated with only one world; a world, however, can have several windows that show portions of it. The coordinates of a window can be transformed into the coordinates of a world by using various methods and flavors.

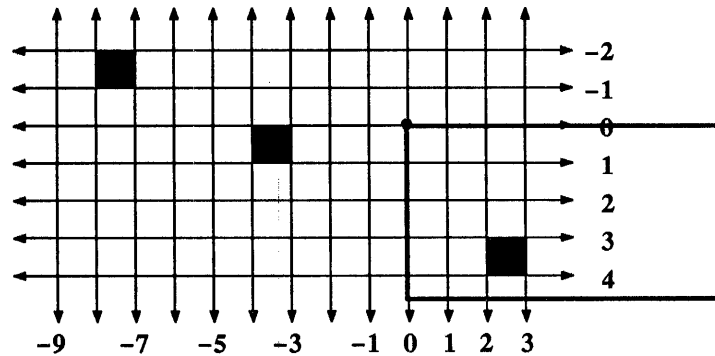
Suppose the following representation of a world is measured in centimeters and that a one-centimeter square has its upper left corner located at (2,3) centimeters from the origin of the world. Other squares are located with their upper left corners at (-8,-2) and (-4,0), respectively. Thus, this world's picture list includes three squares.



The position of the square on a window is determined by the transformations—that is, on where the window is in relation to the origin of the world, and what relation each world unit has to the pixels on the window.

Suppose you set:

- The origin of the window at the origin of the world
- Both the x and y coordinates to be 1 centimeter equal to 100 pixels
- The orientation to be the same between the world and the window

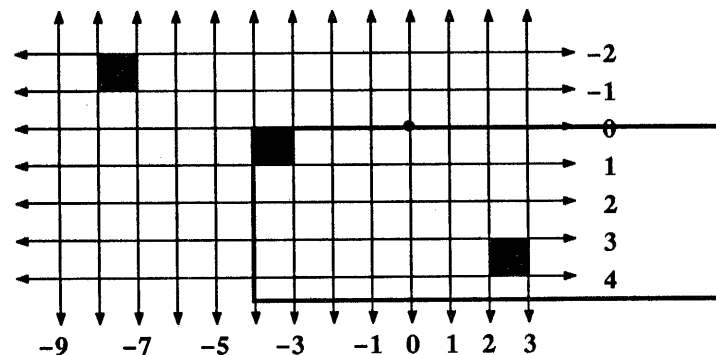


Thus, the window would appear similar to the following. The one square that appears has its upper left corner at world coordinates (2,3) and at window coordinates (200,300). The other squares still exist in the world, but are not displayed on the window.



However, suppose you set:

- The origin of the window at the world coordinate (-4,0)
- The x coordinate to be 1 centimeter equal to 50 pixels
- The y coordinate to be 1 centimeter equal to 100 pixels
- The orientation to be the same between the world and the window



Thus, the window would appear similar to the following. Two images appear in the window, but they are now rectangles (100 pixels high and 50 pixels wide). One square appears at window coordinates (0,0); the other appears at

window coordinates (100, 300). The other square exists in the world, but it is not displayed on the window.



The GWIN package provides the functionality for manipulating a world. Before you can use the capabilities provided by a world, you must load the GWIN package. See paragraph 12.6, Using Graphic Objects; paragraph 12.7, Functions Used with Graphic Objects; and paragraph 12.8, General Flavors Used With Graphic Objects.

Methods and Flavors to Draw Graphic Images

12.4 The following paragraphs discuss methods that create graphic images. Flavors define these methods, and the `send` function can be used to execute these methods. The following example shows how a `send` function appears:

```
(send window-instance :draw-line args)
```

where:

window-instance is an instance of the window flavor, such as `w:window`.

args are the arguments required for the line drawing method. The following code draws a line on a window from the coordinates 0,0 to 100,100.

```
(defun draw-a-line (from-x from-y to-x to-y)
  "Draws a line between the two specified points on my-window."
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (make-instance 'w:window))) ; Check 1: Does window exist?
  (if (not (send my-window :exposed-p))
      (send my-window :expose)) ; Check 2: Is window exposed?
  (send my-window :draw-line from-x from-y to-x to-y 1 w:black w:normal))
(draw-a-line 0 0 100 100)
```

The first nine lines of code define the `draw-a-line` function, which is invoked in the last line of code and passes parameters to the `:draw-line` method. The parameters passed are 0, 0, 100, and 100; these correspond to the `from-x`, `from-y`, `to-x`, and `to-y` arguments in the `draw-a-line` function definition; that is, the line is drawn from the upper left corner of the window to position 100,100 in the window.

Before `draw-a-line` draws the line, it makes two checks. First, `draw-a-line` checks to ensure that `my-window` exists; the second check ensures that `my-window` is exposed on the screen. If `my-window` does not exist, the `draw-a-line` function creates it; if `my-window` exists, the `draw-a-line` function does not have to create `my-window`. In either case, `my-window` must be exposed before `draw-a-line` can draw on `my-window`. If `my-window` is not exposed, the second check forces `my-window` to be exposed.

The code allows `draw-a-line` to be invoked as often as necessary to draw a graphics image and to create or expose `my-window`. The code prevents `my-window` from being created every time `draw-a-line` is invoked.

The last line of code produces a display similar to the following:



Methods That Use w:graphics-mixin

12.4.1 w:graphics-mixin contains the methods used to draw graphic images on a window. These methods use a world-coordinate system that extends infinitely in two dimensions.

w:graphics-mixin draws all images similarly. All images with edges draw themselves by constructing a polyline representation of themselves and then calling **:draw-polyline** with this representation.

In addition to the methods that actually draw on a window, **w:graphics-mixin** includes **:min-dot-delta** and **:min-nil-delta**, which are used in determining whether the drawn image will be big enough to actually be worth drawing. **w:graphics-mixin** also includes **:allow-interrupts?**, which is a flag that allows the drawing of a picture list to be interruptible.

w:graphics-mixin

Flavor

Required flavor: **w:minimum-window**

Enables a user to draw graphic images on a window.

Smallest Size **12.4.1.1** The following methods determine whether an image is large enough to be drawn.

:min-dot-delta *min* Initialization Option of **w:graphics-mixin**
Gettable, settable. Default: 6.


Sets the minimum size that an image can be and still be drawn in detail.


:min-nil-delta *min* Initialization Option of **w:graphics-mixin**
Gettable, settable. Default: 2.

Sets the minimum size that an image can be and still be drawn.

For example, suppose your world included the following image, which you call a frobboz:



:min-dot-delta sets the smallest size that the window attempts to draw the frobboz so that it is recognizable. It appears as a very small image, but still recognizable as a frobboz: .

:min-nil-delta, on the other hand, sets the smallest size that the image can be drawn and still be seen—but probably not recognizable as a frobboz: . This unrecognizable image is usually referred to as a *blob*.

Picture Lists 12.4.1.2 You can group graphic images into *picture lists*, lists of images that specify which images are to be drawn first and which are to be drawn last. Images that are drawn later can overlay and obscure images drawn earlier.

:draw-picture-list *items* &optional (*world* nil) Method of **w:graphics-mixin**
:undraw-picture-list *items* Method of **w:graphics-mixin**

Draws or erases, respectively, a list of graphic entities in the window. Clipping is performed on each item, and a check is done to see if the item is too small to bother drawing or undrawing in detail. The method returns **t** if the operation is performed without interruption; if the operation is interrupted, the method returns **nil**. *items* is the list of graphic entities; *world* specifies the world in which the entities are drawn.

:allow-interrupts? *t-or-nil* Initialization Option of **w:graphics-mixin**
Gettable, settable. Default: **nil**

Sets a flag that allows the drawing of a picture list to be interruptable. **nil** specifies that the entire picture list is to be drawn.

Methods That Draw Graphics 12.4.1.3 The following methods draw simple geometric figures on the video display. These figures include points, lines (including simple lines, dashed lines, polylines, and splines), arcs and circles, triangles, rectangles, polygons, raster images (which are drawn line by line), and strings.

In most cases, you should specify points for the figures in world coordinates. Exceptions to this are the **:point** and **:draw-point** methods, which take window coordinates. Most of these methods include several arguments in common:

thickness — The width of the edge in world coordinate units. This argument is not used for methods that draw filled images. You can specify a thickness so large that the image appears to be filled. The default is 1.

color — The color of the image drawn (for filled images) or the color of the edges of the image drawn (for images that are not filled). In a monochrome environment, the color argument serves as an index into a table of patterns that are various dithered shades of gray. Table 12-1 lists the possible values.

In a color environment, the color argument is the value or name of a color in the color map. An example of a name is **w:green**. The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map Table.

In a monochrome environment, the default color is black. In a color environment, the default color is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing).

The following variable can be used as the value of the *color* argument:

w:*default-foreground* Variable

The default foreground color of a window. The default value of this variable is black (8).

alu — The ALU used to draw the image on the screen. Table 12-2 lists several values for *alu*. (For information on the additional color ALUs, refer to paragraph 19.6, Color ALU Functions.) You can use either the symbolic name, such as **w:combine**, or the logical name, such as **w:alu-transp**. It is highly recommended that you use symbolic names because this will help in porting applications to color.

The default for all the methods is **w:normal**, which is **w:alu-seta** on both monochrome and color systems.

texture — An array suitable for bitbltting to the screen, such as the gray patterns in a monochrome environment. (Refer to paragraph 12.4.2, Bit Block Transferring, for more information on arrays that can be copied or merged by using the **bitblt** function.) In a color environment, the type of array determines how the texture appears:

- If the array is a one-bit array, the texture appears in the foreground and background color and is affected by the current pixel values and the *alu* argument.
- If the array is an eight-bit array, each eight-bit element of the array is treated as a color, the *color* argument is ignored, and the resultant screen display is a function of the contents of the texture array, current pixel values, and the *alu* operation.

A set of predefined textures is contained in the global array **w:*textures***. You can view these textures by using the **w:select-texture-with-mouse** function. This function returns the index into the **w:*textures*** array for the selected texture. The default is **nil**. The following shows an example:

```
(send w:selected-window :draw-filled-rectangle 50 50 100 100 w:red
  w:normal t (aref w:*textures* 27))
```


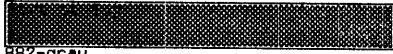
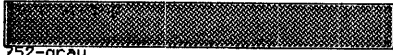





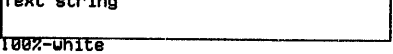
The following variable can be used as the value of the *texture* argument:

w:*default-texture* Variable

The default texture, or stipple pattern, used for an image on the screen. The value of this variable must be an array (or **nil**). The default value is **nil**.

Table 12-1

Color Values for Graphic Methods for Monochrome Environments

Value	% Gray	Screen Representation
8, w:black	Black	 100%-black
7, w:88%-gray-color	88% gray	 88%-gray
6, w:75%-gray-color	75% gray	 75%-gray
5, w:66%-gray-color	66% gray	 66%-gray
4, w:50%-gray-color	50% gray	 50%-gray
3, w:33%-gray-color	33% gray	 33%-gray
2, w:25%-gray-color	25% gray	 25%-gray
1, w:12%-gray-color	12% gray	 12%-gray
0, w:white	White	 100%-white
nil	No color, transparent	

Note:

Do not confuse the names of the values, such as w:12%-gray-color, with the names of textures, such as w:12%-gray. The value names are integers; the texture names are arrays. (Refer to the w:make-gray function in paragraph 12.4.2, Bit Block Transferring, for a list of texture names.)

Table 12-2

ALU Values for Graphic Methods

Graphics Window System	W Package in Monochrome Environment	W Package in Color Environment
w:opposite ¹	w:alu-xor	
w:combine ¹	w:alu-transp ²	w:alu-transp
w:normal ¹	w:alu-seta	
w:erase ¹	w:alu-back ³	w:alu-back

Note:

¹ It is highly recommended that you use symbolic names, such as **w:combine**, instead of logical names, such as **w:alu-transp**, because this will help to remain compatible with future Explorer systems.

² **w:alu-transp** can be used on both color and monochrome systems; it performs the same operation as **w:alu-ior** on a monochrome system.

³ **w:alu-back** can be used on both color and monochrome systems; it performs the same operation as **w:alu-andca** on a monochrome system.

:point *x y*

Method of **w:graphics-mixin**

Returns two values for the pixel located at the window coordinates specified by *x* and *y*. If the coordinates are outside the interior of the window, the returned values are 0 and **nil**; in other words, clipping causes a pixel to have a value of 0 and **nil**.

- In a monochrome environment, the first value is either 0 or 1, specifying whether the pixel is on or off.

In a color environment, the first value is to ensure compatibility with existing software written for monochrome systems. This value is 0 if the pixel is currently the same color as the window's background color. If the pixel is any color other than the background color, 1 is returned.

- The second value returned is **nil** on a monochrome system, or on a color system the eight-bit color value of the pixel is returned.

:draw-point *x y* &optional *alu color*

Method of **w:graphics-mixin**

Draws a pixel at the window coordinates specified by *x* and *y*. **:draw-point** combines the new value with the existing window pixel value at the specified coordinates according to *alu*. *alu* defaults to the value of the **w:char-aluf** instance variable for the window. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

:draw-line *from-x from-y to-x to-y*
 &optional (*thickness 1*) *color (alu w:normal)*
 (*draw-end-point t*) (*texture w:*default-texture**)

Method of **w:graphics-mixin**

Draws a line between *from-x*, *from-y* and *to-x*, *to-y* with the specified thickness. This line is actually a rectangle that is rotated and centered along the line. *draw-end-point* determines whether the end point is drawn. If several connecting lines are to be drawn, specifying **nil** for *draw-end-point* draws the connecting points correctly. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

`:draw-dashed-line` *x0 y0 x1 y1* Method of `w:graphics-mixin`
 &optional (*thickness* 1) *color* (*alu w:normal*) (*dash-spacing* 20)
space-literally-p (*offset* 0) (*dash-length* (`floor` *dash-spacing* 2))
 (*texture w:*default-texture**)

Draws a dashed line between *x0,y0* and *x1,y1* of the specified thickness. *dash-spacing* is the distance, in pixels, from the beginning of a dash to the end of the space following the dash. *space-literally-p* determines whether to adjust the spacing between the dashes. If *space-literally-p* is nil (the default), the spacing is adjusted so that the dashes fit evenly into the length of the line. If *space-literally-p* is non-nil, the dashes are spaced exactly as specified, even though the line may end in the spacing between dashes. The example shows the effect of *space-literally-p*.

offset is how far, in pixels, from *x0,y0* the first dash starts and how soon before *x1,y1* the last dash ends. If *offset* is 0, the first dash starts on *x0,y0*, and the last dash ends on *x1,y1*. The *offset* is specified as the number of pixels.

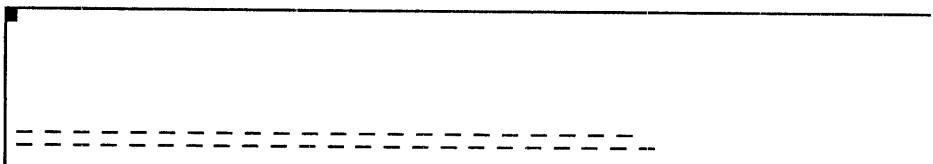
dash-length is the actual length of the dash. If *dash-length* is not specified, the default dash length is half the value specified by *dash-spacing*.

Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

The following code defines a function that draws a dashed line.

```
(defun draw-dashed-line (from-x from-y to-x to-y space-literally-p)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-dashed-line from-x from-y to-x to-y 1 w:black
    w:normal 22. space-literally-p)
  )

(progn
  (draw-dashed-line 10 100 500 100 t)
  (draw-dashed-line 10 110 500 110 nil))
```



:draw-polyline *x-points y-points* Method of **w:graphics-mixin**
 &optional (*thickness 1*) *color*
 (*num-points* (min (*length x-points*) (*length y-points*)))
 (*alu w:normal*) (*draw-end-point t*) (*texture w:*default-texture**)

Draws a polyline with a thickness. A polyline is actually a sequence of lines connected together. *x-points* and *y-points* are arrays of coordinates. *num-points* specifies the number of points to actually draw. If *num-points* is smaller than the length of *x-points* or *y-points*, only the first *num-points* points are used. If *num-points* is larger than the length of *x-points* or *y-points* (that is, if you specify fewer points in the array than you specify to use), the result is indeterminate.

draw-end-point determines whether the end point is drawn. If several connecting lines are to be drawn, specifying *nil* for *draw-end-point* draws the connecting points correctly. (Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.)

```
(defun draw-polyline (x-points y-points num-points draw-end-point)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-polyline x-points y-points 2 w:black num-points
    w:normal draw-end-point)
  )
```

For example, the following code produces a display similar to:

```
(draw-polyline '(28 32 46 35 39 28 17 21 10 24 28)
               '(20 32 32 40 53 45 53 40 32 32 20) 11 t)
```



Note that, in the following case, the star is not closed because the code only used 10 of the available 11 points.

```
(draw-polyline '(28 32 46 35 39 28 17 21 10 24 28)
               '(20 32 32 40 53 45 53 40 32 32 20) 10 nil)
```



:draw-arc *x-center y-center x-start y-start* Method of w:graphics-mixin
 &optional (*arc-angle* 360) (*thickness* 1)
color (*alu w:normal*) (*num-points* 29)
*(texture w:*default-texture*)*

:draw-filled-arc *x-center y-center x-start y-start* Method of w:graphics-mixin
 &optional (*arc-angle* 360) *color*
(alu w:normal) (*num-points* 29) (*draw-edge* t)
*(texture w:*default-texture*)*

Draws an arc (a portion of a circle) with its center at *center-x,center-y*, beginning at point *x-start,y-start* and continuing for *arc-angle* degrees. **:draw-arc** draws a hollow arc with an edge of *thickness*. **:draw-filled-arc** draws a solid, filled-in arc. If *draw-edge* is nil, the method does not draw the edges. *num-points* is the number of points used to draw the arc. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

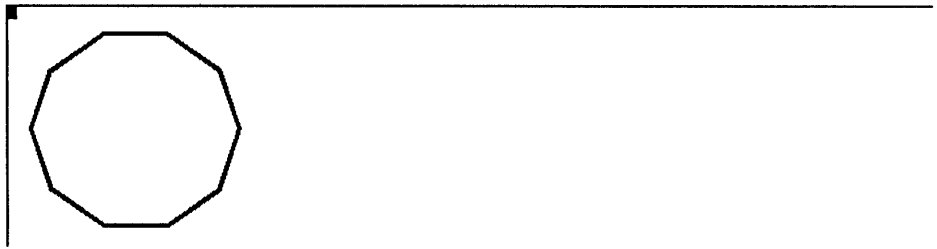
:draw-circle *x-center y-center radius* Method of w:graphics-mixin
 &optional (*thickness* 1)
color (*alu w:normal*) (*num-points* 29)
*(texture w:*default-texture*)*

:draw-filled-circle *x-center y-center radius* Method of w:graphics-mixin
 &optional *color* (*alu w:normal*)
(num-points 29) (*draw-edge* t) (*texture w:*default-texture*)*

Draws a circle with its center at *center-x* and *center-y*, with a radius of *radius*. **:draw-circle** draws a hollow circle with an edge of *thickness*. **:draw-filled-circle** draws a solid, filled-in circle. If *draw-edge* is nil, **:draw-filled-circle** does not draw the edges. *num-points* is the number of points used to draw the arc. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

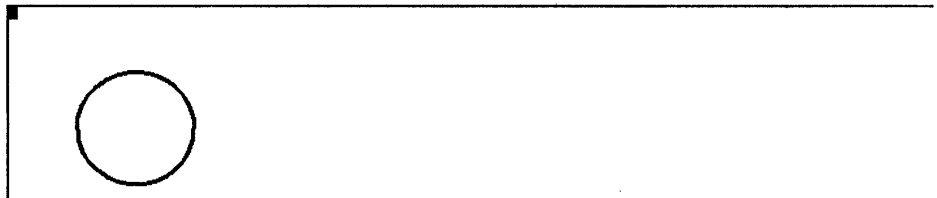
```
(defun draw-circle (x-center y-center radius num-points)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-circle x-center y-center radius
                  2 w:black w:normal num-points)
)
```

```
(draw-circle 100 100 80 10)
```



Compare the smoothness of the following figure, drawn with 30 points, with the angularity of the preceding figure, drawn with only 10 points.

```
(draw-circle 100 100 45 30)
```



```
:draw-triangle x1 y1 x2 y2 x3 y3                                Method of w:graphics-mixin
  &optional (thickness 1)
  color (alu w:normal)
  (texture w:*default-texture*)
:draw-filled-triangle x1 y1 x2 y2 x3 y3                          Method of w:graphics-mixin
  &optional color (alu w:normal)
  (draw-third-edge nil) (draw-second-edge t) (draw-first-edge t)
  (texture w:*default-texture*)
```

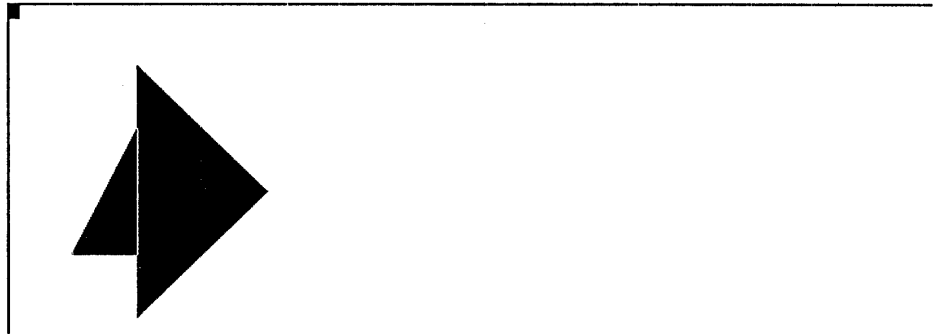
Draws a triangle with vertices at $x1,y1$; $x2,y2$; and $x3,y3$. **:draw-triangle** draws a hollow triangle with an edge of *thickness*. **:draw-filled-triangle** draws a solid, filled-in triangle. *draw-third-edge*, *draw-second-edge*, and *draw-first-edge* specify whether to draw the respective edges. If one of these arguments is *nil*, that edge is not drawn. Whether an edge is drawn is only important when you are drawing triangles whose edges overlap and when you are using an *alu* value of **w:opposite**. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

```
(defun draw-filled-triangle (x1 y1 x2 y2 x3 y3 draw-third-edge
                             draw-second-edge draw-first-edge alu)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
    (send my-window :expose))

  (send my-window :draw-filled-triangle x1 y1 x2 y2 x3 y3
    w:black alu draw-third-edge
    draw-second-edge draw-first-edge)
)
```

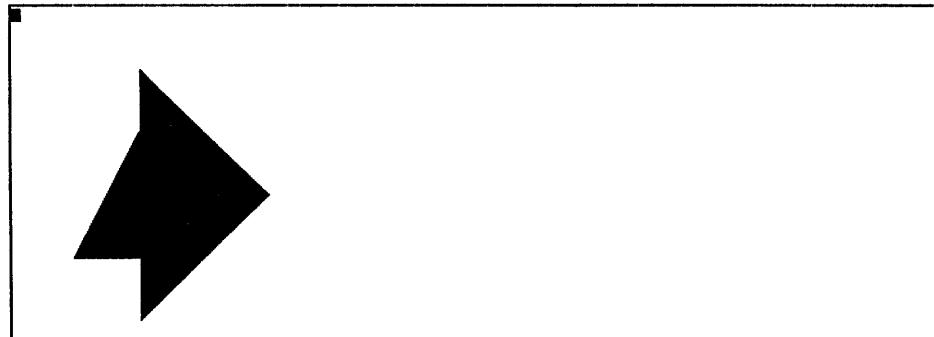
The following example shows the problem that can be caused when you draw with triangles with overlapping edges with an ALU value of `w:opposite`. Note that, where the edges overlap, a one-pixel wide gap is left.

```
(progn
  (draw-filled-triangle 100 100 100 200 50 200 t t t w:opposite)
  (draw-filled-triangle 100 50 100 250 200 150 t t t w:opposite))
```



When you specify *not* to draw the first edge of the second triangle, this gap is closed. (You could achieve the same result by not drawing the third edge of the first triangle.)

```
(progn
  (draw-filled-triangle 100 100 100 200 50 200 t t t w:opposite)
  (draw-filled-triangle 100 50 100 250 200 150 t t nil w:opposite))
```



```
:draw-rectangle left top r-width r-height
  &optional (thickness 1)
  color (alu w:normal)
  (texture w:*default-texture*)
```

Method of `w:graphics-mixin`

```
:draw-filled-rectangle left top r-width r-height
  &optional color (alu w:normal)
  (draw-edge t) (texture w:*default-texture*)
```

Method of `w:graphics-mixin`

Draws a rectangle with its upper left corner at *left,top*, whose width is *r-width*, and whose height is *r-height*. The sides of the rectangle are always parallel to the window edges. `:draw-rectangle` draws a hollow rectangle with an edge of *thickness*. `:draw-filled-rectangle` draws a solid, filled-in rectangle. If *draw-edge* is `nil`, `:draw-filled-rectangle` does not draw the edges. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

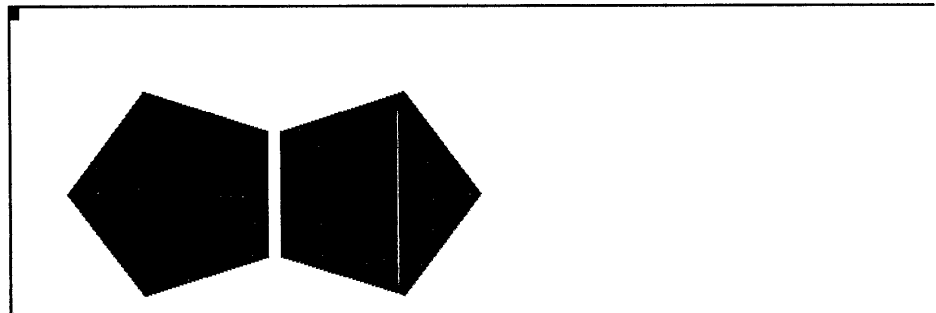
:draw-regular-polygon *x1 y1 x2 y2 n* Method of **w:graphics-mixin**
 &optional *color (alu w:normal)*
 (*draw-edge t*) (*texture w:*default-texture**)

Draws a solid, filled-in, regular polygon of *n* sides with its first edge specified by the line drawn from *x1,y1* to *x2,y2*. Because the polygon is regular, all sides are the same length, and all interior angles are equal. The sign of *n* determines which side of the line the figure is drawn on. If *n* is positive, **:draw-regular-polygon** draws the polygon in a clockwise direction. If *n* is negative, **:draw-regular-polygon** draws the polygon in a counterclockwise direction. For example, if the value of *n* is positive, **:draw-regular-polygon** starts at the *x1,y1* window coordinates and draws a line to *x2,y2*; then **:draw-regular-polygon** turns in a clockwise direction and draws the next side. Alternately, if the value of *n* is negative, **:draw-regular-polygon** turns in a counterclockwise direction before drawing the next side of a polygon. (Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.)

The following example defines a function that draws two regular polygons: one is drawn with a positive value for *n*; the other uses a negative value for *n*.

```
(defun draw-polygon (x1 y1 x2 y2 n-sides)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
    (send my-window :expose))
  (send my-window :draw-regular-polygon x1 y1 x2 y2 n-sides))

(progn
  (draw-polygon 200. 100. 200. 200. 5)
  (draw-polygon 210. 100. 210. 200. -5))
```

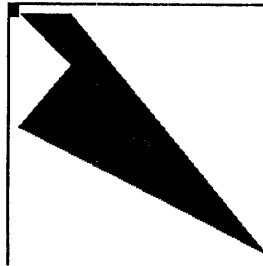


:draw-filled-polygon *x-center y-center x-points y-points* Method of **w:graphics-mixin**
 &optional *color*
 (*num-points* (min (length *x-points*) (length *y-points*)))
 (*alu w:normal*) (*texture w:*default-texture**)

Draws a solid, filled-in convex polygon that is not necessarily regular. This polygon has no inward pointing vertices and is solid from the center point to the edges between the vertices. The center of the polygon is at *x-center*, *y-center*. The *x-points* and *y-points* arguments are arrays of x or y coordinates, respectively, that specify the vertices of the polygon. *num-points* specifies the number of points to actually draw. If *num-points* is smaller than the length of *x-points* or *y-points*, only the first *num-points* points are used. If *num-points* is larger than the length of *x-points* or *y-points* (that is, if you specify fewer points in the array than you specify to use), the result is indeterminate. (Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.)

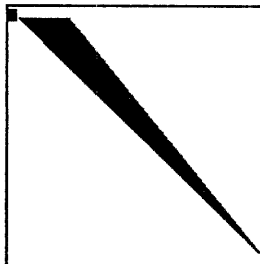
```
(defun draw-filled-polygon (x-center y-center x-points y-points
                          num-points)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (w:make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-filled-polygon x-center y-center
    x-points y-points w:black num-points w:normal)
  )

(draw-filled-polygon 200 200 '(10 50 30 100 10 50)
                      '(10 10 30 100 100 50) 6)
```



Compare the preceding figure with the following figure; the difference is caused by specifying a different number of points to use from the array.

```
(draw-filled-polygon 200 200 '(10 50 30 100 10 50)
                      '(10 10 30 100 100 50) 4)
```



:draw-string *font text x y* Method of **w:graphics-mixin**
 &optional *color*
 (*left* 0) (*tab* 8) (*scale* 1) *alu*

Draws a string of text (*text*) in the specified font. *font* must be a graphics font object (a raster font); that is, *font* must be one of the fonts listed in the **w:*font-list*** variable. *left* sets the left margin (the x position that the cursor returns to after a line feed). *tab* is the number of spaces used to determine the amount of space that the text moves when you press the TAB key. *scale* is an inverse scaling factor that determines the size that the text is drawn. With *scale* 1, the text prints at its normal size. With *scale* 2, the text prints at half its normal size. With *scale* .5, the text prints at twice its normal size. The *alu* argument defaults to the value of the **w:char-aluf** instance variable for the window. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

```
(defun draw-string (font text x y scale)
  (declare (special my-window))
  (unless (boundp 'my-window)
    (setq my-window (w:make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
    (send my-window :expose))
  (send my-window :draw-string font text x y w:black 0 8 scale))

(progn
  (draw-string w:medfnt-font "This is a full-sized string (scale = 1)." 30 30 1)
  (draw-string w:medfnt-font "This is a half-sized string (scale = 2)." 30 60 2)
  (draw-string w:medfnt-font "This is a double-sized string
    (scale = 0.5)." 30 90 .5)
```

```
This is a full-sized string (scale = 1).
```

```
This is a half-sized string (scale = 2).
```

```
This is a double-sized
(scale = 0.5).
```

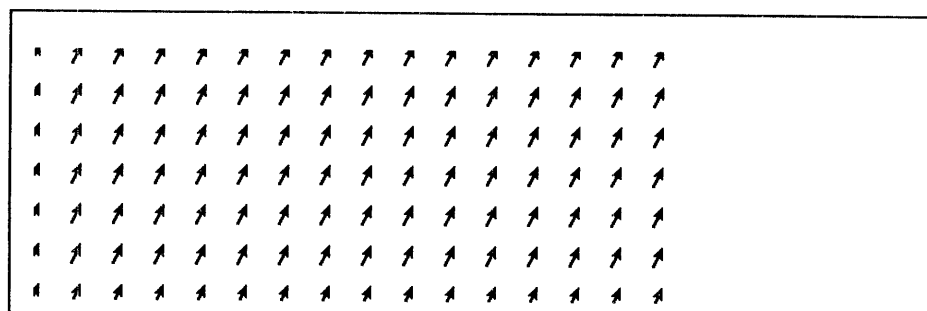
:draw-raster *x y raster start-x start-y r-width r-height* Method of **w:graphics-mixin**
 &optional *color* (*alu* **w:normal**)

Draws a raster image in the window by copying the pixel data. This is just like a **bitblt** operation except that the starting location is in world coordinates, and the image is clipped to the window edges.

x and *y* are the dimensions of the rectangle to be copied; *raster* is the two-dimensional array from which **:draw-raster** copies the bits to the window. *start-x*, *start-y* specifies the subscripts of the array being copied, that is, the upper left corner of the rectangle to be copied. *r-width* and *r-height* are the window coordinates for the upper left corner where the raster image is to be copied. Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.

The following code draws fills an area of the screen from 100,100 to 500,500 with copies of the bit array in **sys:mouse-cursor-pattern**. In the Zmacs editor, this array is a pattern of northeast arrows. To redisplay the previous contents of the screen, select Refresh from the System menu.

```
(send w:selected-window :draw-raster 32 32
  sys:mouse-cursor-pattern 100 100 500 200)
```



:draw-cubic-spline *px py z* Method of **w:graphics-mixin**
 &optional (*thickness 1*) *color (alu w:normal)*
 (*c1 :relaxed*) (*c2 c1*) *p1-prime-x p1-prime-y pn-prime-x pn-prime-y*
 (*texture w:*default-texture**)

Draws a cubic spline curve that passes through a sequence of points called *knots*. The **:draw-cubic-spline** method draws a smooth-line curve through the knots; this method is different from **:draw-polyline**, which draws lines to connect points. (Refer to the beginning of paragraph 12.4.1.3, Methods That Draw Graphics, for details on the *color* argument.)

Arguments: *px, py* — Arrays that contain the window coordinates for the points the curve connects. The number of points is determined by the length of *px*.

Because there are no derivatives for the end points, you can specify the boundary conditions for these points. The *c1* and *c2* arguments determine the boundary conditions for the starting and ending points, respectively.

z — The number of computed knots to use when drawing the curved line between each *px,py* pair. The **:draw-cubic-spline** method uses these computed knots to draw the curved line between each *px,py* point.

c1, c2 — How the **:draw-cubic-spline** method specifies the derivative at the end points. The possible values for *c1* and *c2* are **:relaxed**, **:clamped**, **:cyclic**, and **:anti-cyclic**:

- **:relaxed** makes the derivative at this point 0.
- **:clamped** allows the caller to specify the derivative. The arguments *p1-prime-x* and *p1-prime-y* specify the derivative at the starting point and are used only when *c1* is **:clamped**. The *pn-prime-x* and *pn-prime-y* arguments specify the derivative at the ending point and are used only if *c2* is **:clamped**.
- **:cyclic** specifies that the derivatives for both end points are equal. The *p1-prime-x* and *p1-prime-y* arguments determine the derivatives. If *c1* is specified as **:cyclic**, then *c2* is ignored. If you use **:cyclic** to draw a closed curve through *n* points, then the *px* and *py* arrays must contain *n + 1* points. The first and last points in the *px* and *py* arrays must be equal.
- **:anti-cyclic** makes the derivative of the starting point the negative of the derivative of the ending point. The *p1-prime-x* and *p1-prime-y* arguments determine the derivatives. If *c1* is specified as **:anti-cyclic**, then *c2* is ignored.

p1-prime-x, p1-prime-y — When **:clamped** is specified for the *c1* argument, values passed to *p1-prime-x* and *p1-prime-y* specify the derivative for the starting point. If **:clamped** is not specified for *c1*, *p1-prime-x* and *p1-prime-y* are ignored.

The *pl-prime-x* and *pl-prime-y* arguments define a vector that pulls the first part of the spline in a certain direction when the spline is drawn. The vector is the same as that of a line segment between the window coordinates 0,0 and *pl-prime-x*,*pl-prime-y*. The direction of the vector is the same as that of the line segment just described. The first part of the spline is pulled in the direction of the vector, with the magnitude of the pull relative to the length of the vector. The larger the vector, the farther the spline is pulled.

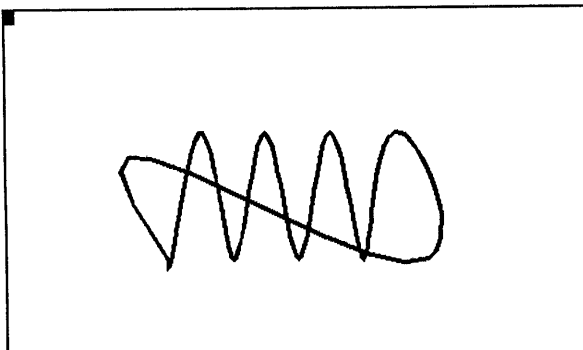
pn-prime-x, *pn-prime-y* — When **:clamped** is specified for *c2*, *pn-prime-x* and *pn-prime-y* specify the derivative for the ending point. If **:clamped** is not specified for *c2*, *pn-prime-x* and *pn-prime-y* are ignored.

The effects of *pn-prime-x* and *pn-prime-y* are the same as the effects of *pl-prime-x* and *pl-prime-y*, except that *pn-prime-x* and *pn-prime-y* affect the last part of the spline.

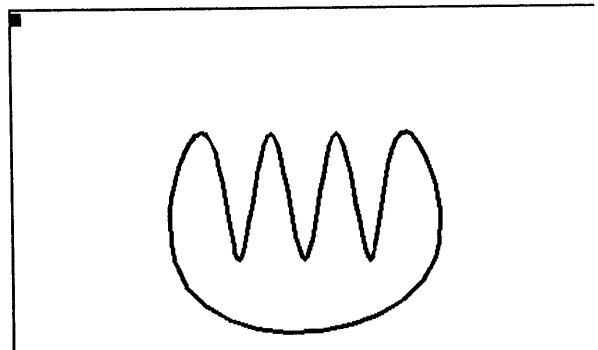
```
(defparameter x-array '(125 150 175 200 225 250 275 300 325 125))
(defparameter y-array '(200 100 200 100 200 100 200 100 200 200))
```

```
(defun draw-cubic-spline (z c1 c2 pl-prime-x pl-prime-y
                          pn-prime-x pn-prime-y)
  (unless (boundp 'my-window)
    (setq my-window (w:make-instance 'w:window)))
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-cubic-spline x-array y-array z
    2 w:black w:normal c1 c2 pl-prime-x pl-prime-y
    pn-prime-x pn-prime-y
  ))
```

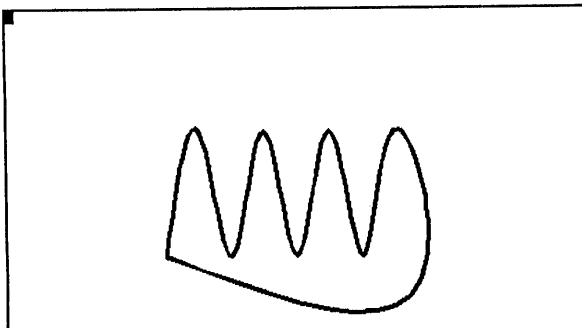
```
(draw-cubic-spline 10
  :clamped :clamped 0 1 2 3)
```



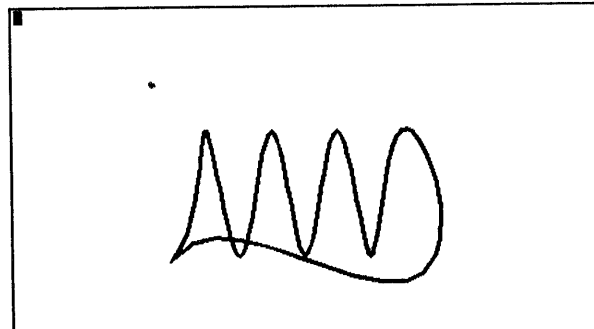
```
(draw-cubic-spline 10
  :cyclic nil 2 1 2 1)
```



```
(draw-cubic-spline 10
  :relaxed :relaxed 0 1 2 3)
```



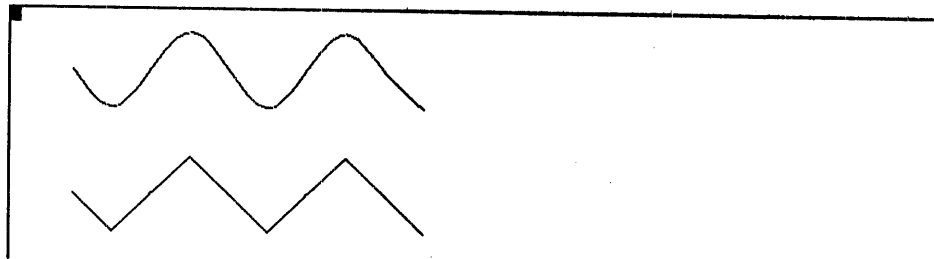
```
(draw-cubic-spline 10
  :anti-cyclic nil 2 1 2 1)
```



The following code creates two lines that show the difference between how the `:draw-cubic-spline` and the `:draw-polyline` methods draw an image on a window. The `:draw-cubic-spline` method draws the upper image, and `:draw-polyline` draws the lower image.

```
(defvar x-array (make-array 10 :type 'art-q))
(defvar y-array (make-array 10 :type 'art-q))
(defun compare-spline-and-polyline ()
  "Draw a spline and then a polyline from an arbitrary
  set of x,y points."
  (declare (special my-window))
  (unless my-window
    (setq my-window (make-instance 'w:window)))
  ; Create x and y points.
  (dotimes (point 10)
    (aset (+ 50. (* point 30.))
          x-array point)
    (aset (+ 50. (floor (* 30. (sind (* point 90.))))))
          y-array point ))
  ; Draw the spline.
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-cubic-spline x-array y-array 5)
  ; Shift the y points down so we can see both images.
  (dotimes (point 10)
    (aset (+ 100. (aref y-array point)) y-array point))
  ; Draw the polyline.
  (if (not (send my-window :exposed-p))
      (send my-window :expose))
  (send my-window :draw-polyline x-array y-array))

(compare-spline-and-polyline)
```



Bit Block Transferring 12.4.2 When used in the window system, one of the arrays passed as an argument to a `bitblt` function is usually a window's screen array. When this is true, the array should be prepared with `w:prepare-sheet`. The *Explorer Lisp Reference* manual discusses the `bitblt` function in detail.

However, you can also save bit arrays in files, read them later, display them, and rotate them.

`bitblt alu width height from-array from-x from-y to-array to-x to-y` Function

Copies or merges a rectangular portion of `from-array` to a congruent portion of `to-array`. The microcode requires that the width of the arrays `from-array` and `to-array` times the number of bits per element must be a multiple of 32.

`bitblt` does not simply copy from one array to another; the bits in `from-array` can be merged with the bits in `to-array` using `alu` to control the way the bits are combined.

If you copy a one-bit array to an eight-bit array in a color environment, the contents are properly expanded for color usage. For example, with `w:normal`, if a bit copied from the one-bit array is equal to 1, it is changed to the eight-bit foreground color and placed in the corresponding position in the destination eight-bit array. If the bit is equal to 0, the background color is placed in the destination array. For a discussion of one-bit to eight-bit expansion for color, refer to Section 19, Using Color.

Arguments: `alu` — How the bits are copied onto the window, that is, whether each bit turns a pixel on or off in a monochrome environment or what the color of the pixel is in a color environment.

`width, height` — The dimensions of the rectangle obtained by adding the width or height to either the `to-x` or `to-y` argument.

`from-array` — The array from which the bits are copied. The width of this array must be a multiple of 32.

`from-x, from-y` — The upper left corner of the rectangle.

`to-array` — The array being copied to. The width of this array must be a multiple of 32.

`to-x, to-y` — The upper left corner in the window.

`w:make-sheet-bit-array window x y &rest make-array-options` Function

Creates a two-dimensional array for copying bits to and from windows using the `bitblt` function. `w:make-sheet-bit-array` first creates an array with one dimension that is at least the size specified by `x`. If `x` is not an even multiple of 32, as required by the `bitblt` microcode, the dimension is increased to the next even multiple of 32 that is larger than the `x` argument.

The array that is created is of the same type as the window specified.

Arguments: `window` — The screen array whose type is used to specify the type of the newly created array.

`x, y` — The minimum dimensions of the array.

`make-array-options` — The options passed to the `make-array` function to allow explicit control.

:bitblt *alu width height from-array from-x from-y to-x to-y* Method of
w:stream-mixin

Copies a rectangle of bits from a two-dimensional array onto the window. The **:bitblt** method differs from the **bitblt** function in that the **:bitblt** method performs clipping; the **bitblt** function does not.

Arguments: **alu** — The ALU argument used to copy the bits onto the window, that is, whether each bit turns a pixel on or off in a monochrome environment or what the color of the pixel is in a color environment..

width, **height** — The dimensions of the rectangle.

from-array — The two-dimensional array from which the **:bitblt** method copies the bits to the window.

If the array specified by the **from-array** argument is too small, it is replicated automatically; the *Explorer Lisp Reference* manual discusses this special case with its description of the **bitblt** function.

from-x, **from-y** — The upper left corner of the rectangle to be copied. The **from-x** and **from-y** arguments are the subscripts for that corner.

to-x, **to-y** — The upper left corner in the window. The **to-x** and **to-y** arguments are the window coordinates for that corner. The window coordinates of **to-x** and **to-y** correspond to the subscripts for **from-x** and **from-y**.

:bitblt-from-sheet *alu width height from-x from-y to-array to-x to-y* Method of
w:stream-mixin

Like the **:bitblt** method, except that **:bitblt-from-sheet** sends data in the opposite direction. **:bitblt-from-sheet** copies data from the window to the **to-array**. All other arguments operate the same as with the **:bitblt** method.

:bitblt-within-sheet *alu width height from-x from-y to-x to-y* Method of
w:stream-mixin

Like the **:bitblt** method, except that **:bitblt-within-sheet** copies data from one location to another on the window.

The order in which values are copied is important because the two rectangles may overlap. When the value of **width** is negative, the array is copied in reverse. The copying begins with the last column in the array and ends with the first column of the array. When **height** is negative, **:bitblt-within-sheet** performs a similar action for the subscripts.

w:make-gray *height width &rest rows* Function

Returns a **bitblt** array containing a specified shade of gray (also called texture or stipple pattern) for fill patterns. The pattern's size is **height** by **width**. **width** must be 8 or less; **height** is a multiple of **width** and **height** must be a multiple of 32. **rows** is a list of numbers, each of which describes one row of the pattern, one octal digit per pixel, left to right. For example:

```
(w:make-gray 4 4 #o1010 #o1111 #o0011 #o0101)
```


The named shades of gray that are defined by the system are the following:

w:100%-black	w:66%-gray	w:25%-gray
w:88%-gray	w:50%-gray	w:12%-gray
w:75%-gray	w:33%-gray	w:100%-white

Examples of these shades are shown in Table 12-1. The global array `w:*textures*` contains several other shades. You can view these textures by using the `w:select-texture-with-mouse` function. This function returns the index into the `w:*textures*` array for the selected texture. The default is `nil`.

NOTE: Do not confuse the names of the textures, such as `w:12%-gray`, with the names of color values, such as `w:12%-gray-color`. The texture names are arrays; the value names are integers.

w:mouse-save-image *pathname* Function

Saves a rectangular image specified by the mouse to *pathname*. This function prompts the user to specify an upper left corner and a lower right corner of a rectangle on the video display to be saved.

w:write-bit-array-file *pathname array* Function

&optional (*width* (array-dimension *array* 1))
(*height* (array-dimension *array* 0)) (*x* 0) (*y* 0)

Writes a bitmap to disk in a format that can be read quickly. *pathname* is the file in which to save the bitmap. *array* contains the bitmap with dimensions *width* and *height*. *x* and *y* are offsets into the array that specify the upper left corner of the image to save.

w:read-bit-array-file *pathname* Function

Reads a bit array from the file specified by *pathname*. `w:read-bit-array-file` returns three values: the bitmap and its width and height.

w:show-bit-array *bitmap* Function

&key (:window nil) (:increment 100.)

Displays a bit array specified by *bitmap* on the window specified by the argument to `:window`. If `:window` is not specified, the array is displayed on the currently selected window. When *bitmap* is displayed, it obscures the entire contents of `:window` except for blinkers that are actively blinking. To redisplay the selected window, press any alphanumeric key. If `:window` does not have a bit-save array, its contents are lost. The user can move the bitmap image in steps of `:increment` (by default, 100 pixels) by pressing one of the arrow keys.

This function is primarily used for developing applications that use bitmaps rather than actually displaying bitmaps. If you want to display a bitmap on a window, you should use one of the `bitblt` methods or functions.

The `w:show-bit-array` function returns the x,y location of the upper left corner of the image after the user has moved it.

You can print bit-arrays with the `print-bitmap` and the `print-bitmap-and-wait` functions described in the *Explorer Input/Output Reference* manual.

The following functions rotate bit arrays. These rotations require a number of computations and may require a few moments to complete.

`w:rotate-90` *from-array to-array* Function
 &optional (*width* (array-dimension *from-array* 1))
 (*height* (array-dimension *from-array* 0))
 (*from-x* 0) (*from-y* 0) (*to-x* 0) (*to-y* 0)

Copies and rotates a bitmap clockwise 90 degrees. *from-array* is the original bitmap, which is of dimensions *width* and *height*; *to-array* is the rotated bitmap. *from-array* and *to-array* must be separate arrays. *from-x* and *from-y* specify the index into the array. *to-x* and *to-y* specify the index into the array.

`w:rotate-180` *from-array to-array* Function
`w:rotate-270` *from-array to-array* Function

Copies and rotates a bitmap clockwise 180 degrees or 270 degrees, respectively.

For example, suppose the video display appears as follows:

```
(w:mouse-save-image "mh:webb:image.bitmap")

(setq bitmap
  (w:read-bit-array-file "mh:webb:image.bitmap"))
(w:show-bit-array bitmap)

(setq bitmap90
  (w:make-sheet-bit-array w:selected-window
    (first (array-dimensions bitmap))
    (second (array-dimensions bitmap))))
(w:rotate-90 bitmap bitmap90)
(w:show-bit-array bitmap90)

(setq bitmap180
  (w:make-sheet-bit-array w:selected-window ; Note that the dimensions are reversed for 180.
    (second (array-dimensions bitmap))
    (first (array-dimensions bitmap))))
(w:rotate-180 bitmap bitmap180)
(w:show-bit-array bitmap180)

(setq bitmap270
  (w:make-sheet-bit-array w:selected-window
    (first (array-dimensions bitmap))
    (second (array-dimensions bitmap))))
(w:rotate-270 bitmap bitmap270)
(w:show-bit-array bitmap270)
```

Suppose you save about a fourth of the image in a file by evaluating the following code and clicking left to determine the upper left and lower right corners of the image to be saved. Then, read the bit-array in the file into a variable and display it. Press the space bar or CLEAR SCREEN to redisplay the original contents of the screen. Note that the following sample screens were produced by evaluating the code in a Zmacs buffer.

```
(w:mouse-save-image "mh:webb:image.bitmap")      ; Save the bit-array
(setq bitmap                                     ; Read the file
  (w:read-bit-array-file "mh:webb:image.bitmap"))
(w:show-bit-array bitmap)                       ; Display the bit-array
```

```

(setq bitmap
  (w:read-bit-array-file "mh:webb:image.bitmap"))
(w:show-bit-array bitmap)

(setq bitmap90
  (w:make-sheet-bit-array w:selected-window
    (first (array-dimensions bitmap))
    (second (array-dimensions bitmap))))
(w:rotate-90 bitmap bitmap90)
(w:show-bit-array bitmap90)

(setq bitmap180
  (w:make-sheet-bit-array w:selected-window ; Note that the di
    (second (array-dimensions bitmap))
    (first (array-dimensions bitmap))))
(w:rotate-180 bitmap bitmap180)

```

The following code creates an array to hold the rotated bit-array, rotates the array, and displays it. Note that, for the 180-degree rotation, the dimensions of the array are reversed; for the 90- and 270-degree rotations, the dimensions of the final array are the same as of the initial array.

```

(setq bitmap180
  (w:make-sheet-bit-array w:selected-window ; The dimensions
    (second (array-dimensions bitmap)) ; are reversed for 180.
    (first (array-dimensions bitmap))))
(w:rotate-180 bitmap bitmap180)
(w:show-bit-array bitmap180)

```

```

(w:rotate-180 bitmap bitmap180)
  (first (array-dimensions bitmap)))
  (second (array-dimensions bitmap)))
  (w:make-sheet-bit-array w:selected-window
    (second (array-dimensions bitmap))
    (first (array-dimensions bitmap)))
  (w:rotate-180 bitmap bitmap180)
  (w:show-bit-array bitmap180)

```

Press the space bar or CLEAR SCREEN to redisplay the original contents of the video display.

Similarly, for the 90-degree and 270-degree rotations, the following code produces the figures below:

```
(setq bitmap90
  (w:make-sheet-bit-array
   w:selected-window
   (first (array-dimensions bitmap))
   (second (array-dimensions bitmap))))
(w:rotate-90 bitmap bitmap90)
(w:show-bit-array bitmap90)
```

```
(setq bitmap270
  (w:make-sheet-bit-array
   w:selected-window
   (first (array-dimensions bitmap))
   (second (array-dimensions bitmap))))
(w:rotate-270 bitmap bitmap270)
(w:show-bit-array bitmap270)
```

```
(setf bitmap
  (w:read-bit-array-file "mh:webb;image.bitmap"))
(w:show-bit-array bitmap)
(setf bitmap90
  (w:make-sheet-bit-array w:selected-window
   (first (array-dimensions bitmap))
   (second (array-dimensions bitmap))))
(w:rotate-90 bitmap bitmap90)
(w:show-bit-array bitmap90)
(setf bitmap180
  (w:make-sheet-bit-array w:selected-window
   (first (array-dimensions bitmap))
   (second (array-dimensions bitmap))))
(w:rotate-180 bitmap bitmap180)
(w:show-bit-array bitmap180)
(setf bitmap270
  (w:make-sheet-bit-array w:selected-window
   (first (array-dimensions bitmap))
   (second (array-dimensions bitmap))))
(w:rotate-270 bitmap bitmap270)
(w:show-bit-array bitmap270)
```

```
(setf bitmap
  (w:read-bit-array-file "mh:webb;image.bitmap"))
(w:show-bit-array bitmap)
(setf bitmap90
  (w:make-sheet-bit-array w:selected-window
   (first (array-dimensions bitmap))
   (second (array-dimensions bitmap))))
(w:rotate-90 bitmap bitmap90)
(w:show-bit-array bitmap90)
(setf bitmap180
  (w:make-sheet-bit-array w:selected-window ; Note that the di
   (second (array-dimensions bitmap))
   (first (array-dimensions bitmap))))
(w:rotate-180 bitmap bitmap180)
```

Press the space bar or CLEAR SCREEN to redisplay the original contents of the video display.

Drawing Graphic Images Using Subprimitives

12.5 Graphics are usually drawn on a window by sending messages to the window. However, this operation involves some overhead. If your application requires more speed than is provided by the normal procedure, you can write your own methods to perform graphics methods. Using these or any subprimitives in an application program is discouraged.

New methods must be associated with a flavor. For graphics, you should associate the new method with a mixin flavor. Mixin flavors use other methods for primitive methods and inherit attributes from other flavors. The following code shows a mixin flavor that is built from the w:essential-window flavor.

```

(defflavor circus-mixin () ()
  ;;This makes the instance variables of w:essential-window
  ;;accessible.
  (:required-flavors w:essential-window))

(defmethod (circus-mixin :draw-clown) (size weight happy-p)
  ...)

(defmethod (circus-mixin :draw-tent)
  (height &optional (number-of-rings 3))
  ...)

(defflavor circus-window () (circus-mixin w:window))

(defvar barnum-and-bailey (w:make-instance 'circus-window))

```

This code defines `circus-mixin` as the mixin that contains the new methods, so that you can create a window. The variable `barnum-and-bailey` contains an instance of the window.

When you define primitive output methods, use the graphics subprimitives, such as `sys:%draw-character`, rather than methods described in other sections of this manual. Subprimitives contain minimal error-checking procedures, so you should invoke subprimitives from window methods that first perform error checking and then invoke the subprimitive.

Additionally, graphics subprimitives should be used only within the body of the `w:prepare-sheet` macro. Using graphics subprimitives elsewhere does not ensure that the window is locked, and the image may be corrupted by other output.

CAUTION: Because all interrupts are off, extreme care must be taken in writing the body of `w:prepare-sheet`. You should provide some way to control execution time in case you inadvertently send the computer into an infinite loop.

Unlike methods, microcode subprimitives use coordinates that are relative to the outside of the window.

NOTE: These subprimitives may change in future releases. If you use these subprimitives, you should ensure that you can find these and update your code as necessary.

Also note that the subprimitives do not take a color argument. Color programming is obtained by the proper manipulation of the CSIB hardware and/or source array arguments to the drawing primitives. If you use a lot of subprimitives when developing code for a monochrome system, moving to a color system is more tedious.

Another place you can use subprimitives is inside the `:blink` method of a blinker. The `:blink` method always uses a suitable environment for calling subprimitives, including interrupts being turned off. Because `:blink` uses the

XOR ALU to draw blinkers, it does not matter whether other blinkers are present.

Preparing the Sheet 12.5.1 Before you use any of the subprimitives for drawing, you should prepare the sheet using the following macro.

w:prepare-sheet (*window-instance*) *body* Macro

Provides a safe environment for drawing on a window. The **w:prepare-sheet** macro first waits until the window's output-hold flag is clear, then opens, or turns off, all blinkers so that they do not interfere with the output. Next, **w:prepare-sheet** turns off all interrupts so that the window remains unlocked and the blinkers remain open. Finally, after the *body* completes execution, interrupts are turned back on and the blinkers return to their initial state.

Arguments: *window-instance* — The instance of the window on which the subprimitive is to draw an image. If you use **w:prepare-sheet** within the definition of a *window-oriented* method, the window instance can be specified as **self**.

body — The Lisp code to be executed. The **w:prepare-sheet** macro initializes a special environment that exists when *body* executes. *body* contains the Lisp code that invokes the subprimitive, and possibly other code as well.

Defining the Clipping Rectangle for Drawing 12.5.2 The subprimitives draw only within the array bounds of the destination array. Normally this array is the entire window. You can, however, specify an array—a rectangular portion of the window—that is smaller than the entire window. For example, you can specify that the subprimitives can only draw to the area within the inner edges of the window. Images that would appear outside the array are *clipped*, that is, are truncated without warning. You specify such a rectangle with the **w:with-clipping-rectangle** macro.

w:with-clipping-rectangle (*left top right bottom*) *body* Macro

Executes *body* with the clipping rectangle variables bound to *left*, *top*, *right*, and *bottom*; the variables are restored when the macro exits *body*. These arguments are outside edge coordinates; they must be integers. If you attempt to draw outside the rectangle defined by *left*, *top*, *right*, and *bottom*, the microcode clips (discards) any output that would appear outside the rectangle.

sys:clipping-rectangle-left-edge Variable
sys:clipping-rectangle-top-edge Variable
sys:clipping-rectangle-right-edge Variable
sys:clipping-rectangle-bottom-edge Variable

The edges that define the rectangle that can actually be drawn upon.

**Subprimitives
for Drawing**

12.5.3 Some of these subprimitives can accept an array or a window. In window system applications, the argument is usually a window, but any suitable two-dimensional numeric array will do. Any window must be prepared by using **w:prepare-sheet** before it is used with subprimitives. If the window is not so prepared, an error is signaled.

sys:%draw-character *font char char-width x y alu window-or-array* Function

Draws *char* of *font* on *window-or-array* using *alu*. *font* should be a standard Explorer system font, such as **fonts:cptfont**. The upper left corner of *char* begins at (*x*,*y*). *x* and *y* are relative to *window-or-array*'s outside edges. *char* must have a width less than or equal to 32.

char-width specifies the width of the character to draw. If *char-width* is narrower than the actual width of the character, the character is truncated on the right. In most cases, you should use either the character width returned for that character by **w:font-char-width-table**, or, if the table is **nil**, the width returned by **w:font-char-width**.

To print a string of characters, you can use repeated calls to **sys:%draw-character**.

sys:%draw-character does not use the indexing table for a wide font, so when this function is used on a wide font, the character specified by *char* is not the character you want to draw. For most purposes, you should use **w:draw-char** rather than **sys:%draw-character**.

sys:%draw-rectangle *width height x-bitpos y-bitpos alu window-or-array* Function

Draws a filled, black rectangle of size *width* by *height* using *alu* on *window-or-array*. The upper left corner of the rectangle corresponds to the coordinates *x-bitpos*, *y-bitpos* relative to the upper left corner of the window (or array). To get a rectangle with a different fill color, use **sys:%draw-shaded-triangle** or **sys:%draw-shaded-raster-line**.

sys:%draw-line *x1 y1 x2 y2 alu draw-end-point-p window-or-array* Function

Draws a line from the window coordinates *x1*,*y1* to *x2*,*y2* on *window-or-array* using *alu*. The end point (*x2*,*y2*) is drawn only if *draw-end-point-p* is non-**nil**.

sys:%draw-shaded-raster-line *x1 x2 y alu draw-last-point shade window-or-array* Function

Draws a line one pixel high from the window coordinates *x1*,*y* to *x2*,*y* on *window-or-array* using *alu* and fill color *shade*. The end point (*x2*,*y*) is drawn only if *draw-last-point* is non-**nil**. *shade* must be a two-dimensional array of bits that **bitblt** can replicate; the array's width must be a multiple of 32. Possible values include **w:100%-black**, **w:12%-gray**, **w:75%-gray**, and so on, and those in **w:*textures***.

This function is useful for drawing images that are not easily created from the other subprimitives, such as filled ellipses.

sys:%draw-shaded-triangle *x1 y1 x2 y2 x3 y3 alu draw-first-edge draw-second-edge draw-third-edge shade window-or-array* Function

Draws a triangle with the vertices at the specified coordinates on *window-or-array* using *alu* and fill color *shade*. *shade* must be a two-dimensional array of bits that `bitblt` can replicate; the array's width must be a multiple of 32. Possible values include `w:100%-black`, `w:12%-gray`, `w:75%-gray`, and so on, and those in `w:*textures*`.

The first edge connects *x1,y1* and *x2,y2*; the second edge connects *x2,y2* and *x3,y3*; the third edge connects *x3,y3* and *x1,y1*. The *draw-first-edge*, *draw-second-edge*, and *draw-third-edge* arguments are flags that determine whether a particular edge is drawn. `t` means to draw the edge; `nil` means not to draw the edge.

Using Graphic Objects

12.6 Graphic objects are entities, such as rectangles, triangles, circles, or portions of text, that exist in a graphics database known as a *world*. Even if a graphic object is not currently drawn on the video display in a window, the object exists in memory. To display the object, you draw it rather than recreating it.

The code used to manipulate graphic objects associated with a world is contained in the GWIN package. To access the GWIN code, you must load GWIN by executing the following code:

```
(make-system 'GWIN :noconfirm)
```

This code creates the GWIN package and loads the GWIN system.

NOTE: GWIN must be loaded before you can use most of the code described through the end of this section unless they are listed as belonging to the W package.

Windows With Graphic Object Capabilities

12.6.1 The GWIN package includes instantiable flavors to create windows or panes that allow graphic objects: the `gwin:graphics-window` flavor and the `gwin:graphics-window-pane`. If you want to create a special window flavor that has graphics capabilities, you can use the `gwin:graphics-window-mixin` flavor to include the components, methods, and instance variables necessary to display the contents of a world.

Each window is associated with one world through the `gwin:world` instance variable.

`gwin:graphics-window-mixin` has three components.

- `w:graphics-mixin` provides the ability to draw graphics images.

The list of graphics objects is part of the world; however, the insert methods for `gwin:world` flavor do not draw objects in the window. The draw methods in `w:graphics-mixin` draw objects in the window.

Methods for drawing each type of graphics object, for drawing a list of objects, and for undrawing a list of objects are included.

- **gwin:mouse-handler-mixin** provides the ability to handle the mouse, the mouse cursor, a crosshair cursor, and a grid. The grid is a series of dots that appear on the video display to aid in accurate drawing. Points are placed on this grid (*gridified*) by a method in this component.

The primary difference between this component and the standard mouse handler in the window system is that this component translates between world coordinates and window coordinates.

- **w:transform-mixin** provides the ability to transform coordinates from the world coordinate system to window coordinates. The unit for window coordinates is always pixels. The unit for world coordinates depends on the current transformation. Essentially, world coordinates are in an arbitrary unit (such as inches, pixels, centimeters, and so on).

Graphic Objects

12.6.2 As explained previously, graphic objects are the entities that you draw in the window. Each type of object has a separate flavor, but all graphics objects have a common component, **gwin:basic-graphics-mixin**. This component includes methods for setting edge and fill colors and the ALU function, returning the extents, and highlighting selected objects.

Circles, lines, rectangles, and triangles are the most straightforward graphics objects. With the graphics window system, you can also draw polylines, splines, and text strings. You can draw rulers to aid in drawing other objects. You can save raster objects, which are created like paintings, as bit arrays. You can also define subpictures, which are combinations of objects, for insertion into multiple worlds. You can use background pictures for templates.

Graphic Characters

12.6.3 Text objects are drawn using the graphics window flavor **gwin:font**. You can use the **gwin:create-gwin-fonts** function to turn Lisp machine fonts into graphics window system fonts.

Graphic Cursors

12.6.4 You can use the flavors and mixins for cursors in many ways; several examples are available in the graphics editor.

The mouse cursor, an instantiation of **gwin:cursor**, is the tracker cursor in **gwin:mouse-handler-mixin**. The markers that indicate selected points in the graphics editor are also instantiations of **gwin:cursor**. Instantiations of this flavor are always represented by a character from a font.

The mouse cursor can also have a crosshair associated with it. The crosshair is created by a method in **gwin:mouse-handler-mixin** that draws lines from the tracker cursor to the edges of the display.

The cursor that you use to enter the characters for text objects in the graphics editor is a **gwin:block-cursor**.

The blinking copy of the object that is used for Drag-Copy and Drag-Move in the graphics editor is an instantiation of **gwin:sprite-cursor**.

The **gwin:bitblt-blinker** flavor uses window coordinates instead of world coordinates.

Example 12.6.5 An example using the graphics window system is included in the system. Copy the file `SYS:GWIN.STARTER-KIT;-READ-ME-.LISP` into a Zmacs buffer, and follow the instructions.

As you evaluate the forms in this file, you will see the effects in a window at the top of the display.

Functions Used With Graphic Objects

12.7 In general, the functions used with graphic objects either return a distance, verify a position, or perform a transformation between world and window coordinates. A few miscellaneous functions do other tasks.

Functions That Return Distances **12.7.1** The following functions return either distances between graphic entities or the point on an entity that is nearest to a specified point.

w:dist *x1 y1 x2 y2* Function

Returns the distance between two points (*x1*, *y1*) and (*x2*, *y2*).

gwin:dist-from-rectangle *from-x from-y left top right bottom* Function
&optional (*solid? nil*)

Returns the distance between a point and a rectangle. If *solid?* is *t* and the point lies inside the rectangle, the returned value is negative.

gwin:dist-from-segment *from-x from-y x-start y-start x-end y-end* Function
&optional *delta-x delta-y length-squared*

Returns the distance between a point and a line segment. It also returns the coordinates of the point on the line that is closest to the specified point.

The default for *delta-x* is the difference between *x-end* and *x-start*. The default for *delta-y* is the difference between *y-end* and *y-start*. The default for *length-squared* is the sum of *delta-x* squared and *delta-y* squared.

gwin:nearest-circle-pt *x-point y-point x-center y-center radius* Function

Returns the *x* and *y* coordinates of the point on the circumference of the circle that is nearest the specified point. A line from the center of the circle to the specified point is defined, and the point on the line that is the radius length from the center is returned.

gwin:nearest-pt-on-arc *x-point y-point radius x-center x-start* Function
y-center y-start x-end y-end

gwin:nearest-rectangle-pt *from-x from-y height left top width* Function

gwin:nearest-triangle-pt *from-x from-y x1 y1 x2 y2 x3 y3* Function

Returns the *x* and *y* coordinates of the point on the specified object that is nearest to the specified point.

Functions That Verify Position 12.7.2 The following functions verify a position. Most return a Boolean value (t if the condition is true; nil if the condition is false). **w:sector-code** returns a numeric value that indicates the portion of the video display that contains the point.

gwin:lines-intersect-p *x-start1 y-start1 x-end1 y-end1* Function
x-start2 y-start2 x-end2 y-end2

Returns a Boolean value indicating whether the two specified line segments intersect.

gwin:off-window *x-point y-point window* Function

Returns a Boolean value indicating whether the point is outside the window. Because the point is in world coordinates, it is first transformed to window coordinates.

gwin:point-in-extents-p *x y left top right bottom* Function

Returns a Boolean value indicating whether a point is within the specified rectangular extents.

gwin:point-in-polygon-p *x y x-points y-points &optional num-points* Function

Returns a Boolean value indicating whether the point is in the interior of the closed polygon.

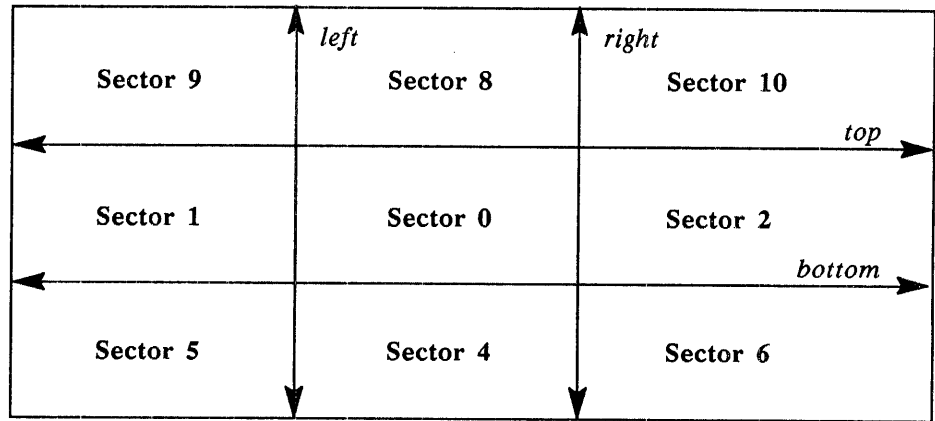
The default for *num-points* is the minimum of the number of elements of *x-points* and *y-points*.

w:sector-code *x y left top right bottom* Function

Returns a bit-encoded value for the location of a point in relation to the specified rectangle. The specified rectangle is the central sector in the following figure. Each sector has a different value; the value for the sector is returned as follows:

- If the point is in a sector (that is, is not on the boundary of a sector), the value of the sector is returned.
- If the point is on the edge of the rectangle, the point is considered to be in sector 0, and 0 is returned.
- If the point is on the boundary of sector 8 but not on the boundary of sector 0, the point is considered to be in sector 8, and 8 is returned.
- If the point is on the boundary of sector 4 but not on the boundary of sector 0, the point is considered to be in sector 4, and 4 is returned.

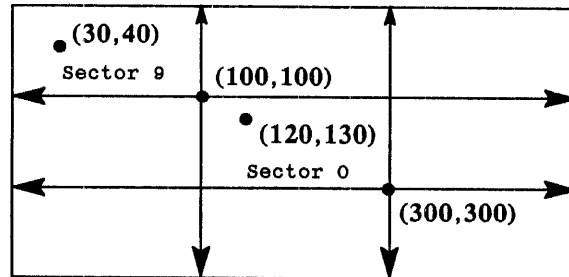
This function uses the Cohen-Sutherland algorithm.



For example, consider the following graphic representation of the function call:

```
(w:sector-code 120 130 100 100 300 300)
=> 0
```

```
(w:sector-code 30 40 100 100 300 300)
=> 9
```



Functions That Perform Transformations

12.7.3 The following functions perform transformations among window and world coordinates.

w:transform-point *x y transform*

Function

Returns the transformed *x* and *y* coordinates for the specified point. The transformation is from world to window coordinates. *transform* is a 3-by-3 matrix that contains scaling, translating, and rotational information for the translating points.

w:untransform-deltas *dx dy transform*

Function

Returns two transformed deltas. A *delta* is the distance between two points. Thus, this function transforms the distance between two window coordinates to the distance between two world coordinates. *transform* is a 3-by-3 matrix that contains scaling, translating, and rotational information for the translating points.

w:untransform-point *x y transform* Function

Returns the transformed *x* and *y* coordinates for the specified point. The transformation is from window to world coordinates. *transform* is a 3-by-3 matrix that contains scaling, translating, and rotational information for the translating points.

Miscellaneous **12.7.4** The following functions perform miscellaneous tasks associated with
 Functions graphics objects.

gwin:calculate-string-motion *font text x y &optional (left 0)* Function
(tab 8) (create-scale 1)

Returns the end point for a text string at the specified point as well as the minimum *x* and *y*, and the maximum *x* and *y* coordinates for the text string are returned. If the text string is drawn, you must move the cursor to the *x* and *y* coordinates returned.

w:create-w-fonts *&optional (fonts-to-convert w:*default-w-fonts*)* Function

Converts a Lisp machine font to a raster font for the graphics window system. The argument can be a single font description or a list of font descriptions. A font description has the following form:

(fonts:symbol-name w:new-symbol-name printable-name-string)

For example, the following is a font description:

(fonts:medfnb w:medfnb-font "Medium Font Bold")

The *w:*font-list** variable (described in paragraph 12.12, Fonts and Text Objects) contains a list of all the available raster fonts.

w:determinant *x y x1 y1 x2 y2* Function

Returns the determinant of a matrix formed from three points. The matrix is as follows:

<i>x</i>	<i>x1</i>	<i>x2</i>
<i>y</i>	<i>y1</i>	<i>y2</i>
1	1	1

This determinant is also the value of the equation for the line through the points *(x1, y1)* and *(x2, y2)* when the point *(x, y)* is substituted.

gwin:find-corresponding-y *x1 y1 x2 y2 x* Function

Finds the point-slope formula and returns the matching *y*-coordinate to *x*.

gwin:load-picture *pathname* Function

Loads a picture file and sets the graphics window system variable *picture* to the list of objects in the picture. If the picture file is invalid or if an error occurred in file handling, *picture* is *nil*.

print-graphics *picture-file &rest options* Function
 &key (:printer-name sys:*default-screen-image-printer*)
 (:dots-per-inch 144.) &allow-other-keys

Prints a picture from a file. Other keywords include the following:

Keyword	Default	Description
:copies	1	The number of copies to be printed
:rotation	nil	Whether the picture will be rotated 90 degrees
:header	t	Whether a header page is printed

If you are printing a picture with colors in a color environment, this function translates the colors in the picture to stipple patterns of gray, based on values in the **printer:color-to-gray-scale-table** variable. For more information on this variable, refer to paragraph 19.9, Printing a Color Screen on a Monochrome Printer.

For more general information about this function, see the *Explorer Input/Output Reference* manual.

gwin:rasterize-objects *window object-list* Function
 Draws a list of objects from *window* into the **w:sprite-window**. The **gwin:rasterize-objects** returns the width and height of the raster equivalent.

gwin:make-sprite-from-objects *window object-list &rest options* Function
 Creates a sprite cursor by drawing the specified graphics objects. The objects are drawn, and the resulting bitmap is saved so any further changes to the objects do not affect the sprite cursor. *window* is the window that the sprite should be displayed in. *object-list* is a list of the objects that comprise the sprite cursor.

options are initialization options for the **gwin:sprite-cursor** flavor. These *options* are passed to the **:add-cursor** method of **gwin:graphics-window-mixin**. The **:array**, **:height**, **:width**, and **:window** initialization options are passed automatically. Other options are described in paragraph 12.8.10, Sprite Cursor Flavor. For an example of using sprites, see the file **SYS:GWIN.STARTER-KIT;-READ-ME-.LISP**.

General Flavors Used With Graphic Objects

12.8 The following flavors are used, in general, to manipulate the world and windows that contain graphic objects.

World Flavor **12.8.1** The graphics window system uses the **gwin:world** flavor to create worlds for graphics objects.

gwin:world Flavor

A world collects a set of graphics objects. The world is a data structure that stores a picture. A world can be saved in file form and restored from a file.

Parameters **12.8.1.1** The following initialization options create the default parameters for this world.

- :current-alu** *current-alu* Initialization Option of **gwin:world**
Gettable, settable. Default: **w:normal**
 Sets the ALU used by default for drawing all graphic objects in this world. Several values are given in Table 12-2, ALU Values for Graphic Methods, and in paragraph 12.4.1.3, Methods That Draw Graphics. For information on ALU arguments in a color environment, refer to paragraph 19.6, Color ALU Functions.
- :current-edge-color** *current-edge-color* Initialization Option of **gwin:world**
Gettable, settable. Default: **w:black**
- :current-thickness** *current-thickness* Initialization Option of **gwin:world**
Gettable, settable. Default: **2**.
 Sets edge color or thickness, respectively, of an object. Edge color is the color of the edge of an object; thickness is width of the edge in world coordinate units. Possible values for *current-edge-color* in a monochrome environment are given in Table 12-1, Color Values for Graphic Methods for Monochrome Environments, and in paragraph 12.4.1.3, Methods That Draw Graphics. Possible values for *color* in a color environment include the value or name of any color from the color map. The system-defined color names are given in Table 19-2, Named Colors in the Default Color Map Table.
- :current-fill-color** *current-fill-color* Initialization Option of **gwin:world**
Gettable, settable. Default: **nil**
 Sets the default fill color of a graphic object. Fill color is the color within an object. *current-fill-color* can be any of the values listed previously for *current-edge-color*.
- :current-font** *current-font* Initialization Option of **gwin:world**
Gettable, settable. Default: **w:standard-font**
 Sets the default font drawn for text objects.
- :current-tab-width** *current-tab-width* Initialization Option of **gwin:world**
Gettable, settable. Default: **8**.
 Sets the number of spaces that are inserted into a text object when the TAB key is pressed.
- :current-margin-width** *current-margin-width* Initialization Option of **gwin:world**
Gettable, settable. Default: **3**.
 Sets the amount of space between a text object or a subpicture and the border that surrounds it.
- :current-pick-tolerance** *current-pick-tolerance* Initialization Option of **gwin:world**
Gettable, settable. Default: **20**.
 Sets how close the mouse cursor must be to an object to select that object when the user clicks the mouse button. If the current pick tolerance is small, a user must point to an object exactly before selecting that object. If the current pick tolerance is large, the user can point somewhere near the object, but not necessarily on it, to be able to select it.

Display Lists **12.8.1.2** The following initialization options and methods manipulate display lists—lists of images that specify which images are to be drawn first and which are to be drawn last. Images that are drawn later can overlay and obscure images drawn earlier.

:display-list *display-list* Initialization Option of **gwin:world**
Gettable, settable. Default: **nil**
 Sets the contents and order of the display list for this world.

:read-display-list *pathname* Method of **gwin:world**
 Restores the display list for this world from a file written out by the **:write-display-list** method.

:write-display-list *pathname* &optional (*compile? t*) Method of **gwin:world**
 (*attributes nil*)

Saves the contents of the world display list to a file. This allows later retrieval by the **:read-display-list** method. *pathname* is the file where the display list is to be stored.

The *compile?* flag indicates whether the file is to be written in compiled form or interpreted form. The file attribute list contains the following values plus any attributes specified through *attributes*:

- **:mode** Common-lisp
- **:package** gwin
- **:base** 10

Refer to the *Explorer Zmacs Editor Reference* manual for more information about file attribute lists.

:objects-in-window *list-of-objects* Initialization Option of **gwin:world**
Gettable, settable. Default: **nil**

Sets a list of the objects in **gwin:display-list** that are inside the most currently refreshed window associated with this world. Thus, you can limit the objects a method searches through. For example, **:pick** searches only the objects identified by **gwin:objects-in-window** to find an object that was selected.

:tick *tick* Initialization Option of **gwin:world**
Gettable, settable. Default: 0.

Sets how many changes have been made to the display list since this picture was created or read from a file.

Entities 12.8.1.3 The following methods manipulate entities (graphic objects).

:create-and-add-entity *type* &rest *options* Method of **gwin:world**
:create-and-add-entity-to-front *type* &rest *options* Method of **gwin:world**

Creates an entity of the specified *type* and adds it to the display list. **:create-and-add-entity-to-front** adds the entity to the front of the display list and is used for the background pictures. *type* can be the name of any graphics object flavor.

:delete-entity *entity* Method of **gwin:world**
 Deletes an entity or list of entities from the display list. World extents are updated if necessary.

:replace-entity *entity new-entity* Method of **gwin:world**
 Replaces a graphics entity in the display list with another graphics entity.

After replacement, world extents are updated if necessary. If the arguments are lists, each element of the first list is replaced by the corresponding element in the second list. If only the first argument is a list, the entities listed are deleted, and the new entity is added to the end of the display list. In these two cases, world extents are always updated.

- :pick** *x y* Method of **gwin:world**
- Returns the graphics object that is closest to the specified point. Specifically, **:pick** searches through the objects in the window if the window contains any objects; if not, **:pick** searches through the entire world for the closest object. **:pick** identifies objects that are within the pick tolerance (that is, objects whose closest distance between the object and the pick point is less than the value of **gwin:current-pick-tolerance**). From these objects, **:pick** selects the object whose edge is closest to the pick point.
- :insert-arc** *x y x1 y1 theta* Method of **gwin:world**
 &optional (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*alu gwin:current-alu*)
- :insert-backgroundpic** *entities name* Method of **gwin:world**
 &optional (*x-scale 1.*) (*y-scale x-scale*) (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*margin gwin:current-margin-width*) (*thickness gwin:current-thickness*)
 (*alu gwin:current-alu*) (*x-origin 0.*) (*y-origin 0.*)
- :insert-circle** *x y r* Method of **gwin:world**
 &optional (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*alu gwin:current-alu*)
- :insert-line** *x1 y1 x2 y2* Method of **gwin:world**
 &optional (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*alu gwin:current-alu*)
- :insert-polyline** *x-points y-points* Method of **gwin:world**
 &optional (*closedp nil*) (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*alu gwin:current-alu*)
- :insert-raster** *xbeg ybeg xend yend bitarray* Method of **gwin:world**
 &optional (*xstart 0.*) (*ystart 0.*) (*width 5.*) (*height 5.*) (*xscale 1.*)
 (*yscale 1.*) (*alu w:combine*)
- :insert-rectangle** *x y w h* Method of **gwin:world**
 &optional (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*alu gwin:current-alu*)
- :insert-ruler** *x1 y1 x2 y2 spacing* Method of **gwin:world**
 &optional (*transform gwin:identity-array*) (*start-value 0*)
 (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*alu gwin:current-alu*)
- :insert-spline** *x-points y-points* Method of **gwin:world**
 &optional (*closedp nil*) (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*alu gwin:current-alu*)
- :insert-subpicture** *x y entities name* Method of **gwin:world**
 &optional (*x-scale 1.*) (*y-scale x-scale*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*margin gwin:current-margin-width*)
 (*thickness gwin:current-thickness*) (*alu gwin:current-alu*)

Continued

:insert-text *x y text-string* Method of **gwin:world**
 &optional (*scale 1.*) (*font gwin:current-font*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*thickness gwin:current-thickness*)
 (*border-color nil*) (*tab gwin:current-tab-width*) *alu*

:insert-triangle *x1 y1 x2 y2 x3 y3* Method of **gwin:world**
 &optional (*thickness gwin:current-thickness*)
 (*edge-color gwin:current-edge-color*) (*fill-color gwin:current-fill-color*)
 (*alu gwin:current-alu*)

Creates a new entity of the specified kind and adds it to the world. The instance of the entity is returned.

For **:insert-raster**, the value for *edge-color* is **gwin:current-edge-color** on a color system; on a monochrome system, the value is black.

For **:insert-ruler**, if the value for *edge-color* is **nil**, the ruler is black instead of transparent. See the flavors for the various entities for an explanation of the other arguments.

For **:insert-text**, the value for *alu* is **w:combine** on a color system; on a monochrome system, the value is **gwin:current-alu**.

Extents 12.8.1.4 The following methods manipulate extents. *Extents* are the minimum and maximum x and y values that completely include all the objects in a world. In other words, this is the smallest amount of the world that can be displayed so that all objects are visible in the window.

An object also has extents, which are the values that completely include that object.

:calculate-extents &optional (*object-list gwin:display-list*) Method of **gwin:world**
 (*update-limits? t*)

Returns the maximum and minimum extents of the graphics objects. Four values are returned: the bottom, left, right, and top limits.

:bottom-limit *bottom-limit* Initialization Option of **gwin:world**
Gettable. Default: 0.

:left-limit *left-limit* Initialization Option of **gwin:world**
Gettable. Default: 0.

:right-limit *right-limit* Initialization Option of **gwin:world**
Gettable. Default: 0.

:top-limit *top-limit* Initialization Option of **gwin:world**
Gettable. Default: 0.

Sets the extreme of the respective coordinates of the objects in the world. Limits are the same as extents.

Graphics Window 12.8.2 To create graphics windows, you can use the **gwin:graphics-window**
 Flavors flavor, the **gwin:graphics-window-pane** flavor, or the **gwin:graphics-window-mixin** flavor.

gwin:graphics-window Flavor
 An instantiable form of the **graphics-window-mixin**.

gwin:graphics-window-pane Flavor
 A pane for structured frames that display graphics.

- :blinker-p** *blinker-p* Initialization Option of **gwin:graphics-window**
Default: nil
- :blinker-p** *blinker-p* Initialization Option of **gwin:graphics-window-pane**
Default: nil
- Sets the **w:blinker-p** instance variable of **w:minimum-window** when the graphics flavor is instantiated. Unlike blinkers in most windows, the cursor-following blinker for graphic windows is disabled by default.
- :borders** *borders* Initialization Option of **gwin:graphics-window**
Default: 3.
- :borders** *borders* Initialization Option of **gwin:graphics-window-pane**
Default: 1.
- Sets the **w:borders** instance variable of **w:borders-mixin** when the flavor is instantiated. The value is the width of the border in pixels.
-
- gwin:graphics-window-mixin** Flavor
Required flavor: **w:minimum-window**
- The basic window mixin for graphics window applications. This mixin provides the required methods for drawing the supported graphics entities. The capability of transforming all drawing operations is also provided.
- :cursor-list** *list* Initialization Option of **gwin:graphics-window-mixin**
Gettable, settable. Default: nil
- Sets the list of the cursors in this window.
- :prompt-text** *prompt-text* Initialization Option of **gwin:graphics-window-mixin**
Gettable, settable. Default: " "
- Sets the list of keywords and values for use in the mouse documentation window. See the description of the **:who-line-documentation-string** method in paragraph 11.6, How Windows Handle the Mouse, for possible keywords.
- :world** *world* Initialization Option of **gwin:graphics-window-mixin**
Gettable, settable. Default: A new instantiation of the **gwin:world** flavor
- Sets the world that the window is associated with.
- :world-extents-window** Method of **gwin:graphics-window-mixin**
- Changes the transformation to display all objects. The returned value is either non-nil if the transformation has been changed or nil if it was not changed.
- :add-cursor** *&rest options* Method of **gwin:graphics-window-mixin**
- Creates a cursor and adds it to the list of cursors that are defined in this window. The argument that is passed, *options*, is the flavor name for the type of cursor that is being added, followed by initialization options for that flavor.
- :delete-cursor** *cursor* Method of **gwin:graphics-window-mixin**
- Removes a cursor from the list of cursors that are defined in this window.

:refresh-area *left top right bottom* &optional (*world-coords t*) Method of **gwin:graphics-window-mixin**
 Redraws all objects that overlap the specified rectangle. If *world-coords* is not *t*, *left*, *top*, *right*, and *bottom* are specified in window coordinates rather than in world coordinates.

:who-line-documentation-string Method of **gwin:graphics-window-mixin**
 Returns the current mouse button documentation. See the description of the **:who-line-documentation-string** method in paragraph 11.6, How Windows Handle the Mouse, for possible keywords.

Cache Window Flavor 12.8.3 The graphics window system uses these flavors to create deexposed windows.

w:cache-window Flavor
 This flavor is used by graphics window fonts to temporarily draw images in scaled form.

w:sprite-window Flavor
 This flavor is used to temporarily draw raster objects in scaled form. **w:sprite-window** is almost identical to **w:cache-window**. **w:sprite-window** contains an eight-bit bit-save array (**:bit-array**) when used on a color system. **w:cache-window** always has a one-bit **:bit-array**.

:set-scales *x-scale* &optional (*y-scale x-scale*) Method of **w:cache-window** and **w:sprite-window**
 Sets the scale values in the transformation for the window. Any global data that is derived from the transformation is updated.

In addition to the method previously discussed, **w:cache-window** and **w:sprite-window** use several default values for initialization options that are different from the default values normally associated with **w:minimum-window**, as follows:

:bit-array *bit-array* Initialization Option of **w:cache-window** and **w:sprite-window**
 Default: An empty 1024-by-1024 pixel array

:deexposed-typeout-action *deexposed-typeout-action* Initialization Option of **w:cache-window** and **w:sprite-window**
 Default: **:permit**

:height *height* Initialization Option of **w:cache-window** and **w:sprite-window**
 Default: 1024.

:width *width* Initialization Option of **w:cache-window** and **w:sprite-window**
 Default: 1024.

w:cache-window and **w:sprite-window** also use several default values that are different from the default values normally associated with **w:graphics-mixin**, as follows:

:min-dot-delta *min-dot-delta* Initialization Option of **w:cache-window** and **w:sprite-window**
 Default: 1.

:min-nil-delta *min-nil-delta* Initialization Option of **w:cache-window** and **w:sprite-window**
 Default: 0.

Transform Mixin 12.8.4 This flavor is a component for graphics windows.

w:transform-mixin Flavor
 Required flavor: **w:minimum-window**

This mixin provides mapping between world and device coordinates. Also, basic methods for modifying the transformation of a window are included.

:identity? *identity?* Initialization Option of **w:transform-mixin**
Gettable, settable. Default: **t**
 Sets a Boolean value that indicates whether the world and window transformations are identical.

:transform Initialization Option of **w:transform-mixin**
Gettable, settable. Default: An array that performs no transformations
 Creates a 3 by 3 matrix that contains scaling, translating, and rotational information for the translating points.

:default-window *&rest ignore* Method of **w:transform-mixin**
 Changes the transformation for the window to the identity transformation. The identity transformation is the default window condition at initialization.

:new-window *x y dx dy* Method of **w:transform-mixin**
 Changes the transformation to fit a specified rectangular area in the window. The *x* and *y* scales are set equal to prevent distortion. If the transformation is not changed, **nil** is returned.

:pan *dx dy* Method of **w:transform-mixin**
 Changes the transformation by translating both horizontally and vertically.

:transform-deltas *dx dy* Method of **w:transform-mixin**
:untransform-deltas *dx dy* Method of **w:transform-mixin**

Transforms a pair of distances between world and window coordinates. A *delta* is the distance between two points. **:transform-deltas** changes from world to window coordinates; **:untransform-deltas** changes from window to world coordinates. The distances are not relative to any fixed point in space, so the translation part of the transformation is not applied to them.

- :transform-point** *x y* Method of **w:transform-mixin**
:untransform-point *x y* Method of **w:transform-mixin**
- Transforms a point between world and window coordinates. **:transform-point** changes from world to window coordinates; **:untransform-point** changes from window to world coordinates.
- :world-edges** Method of **w:transform-mixin**
- Returns four values: the inside left, top, right, and bottom edges of the window in world coordinates.
- :zoom** *sx sy* Method of **w:transform-mixin**
- Changes the transformation by scaling both horizontally and vertically. The center of the window remains at the same world location.

Mouse Handler Mixin 12.8.5 This flavor is a component for graphics windows.

- gwin:mouse-handler-mixin** Flavor
- Required flavor: **w:minimum-window**
 Required instance variable: **w:io-buffer**
 Required methods: **:transform-deltas**, **:transform-point**,
:untransform-point
- This flavor has methods for handling the mouse and window properties that affect the mouse. The mouse cursor uses world coordinates.
- :crosshair-mode** *mode* Initialization Option of **gwin:mouse-handler-mixin**
Gettable, settable. Default: **nil**
- Sets a Boolean value that indicates whether the mouse cursor has a crosshair cursor—**t** indicates the existence of a crosshair cursor, and **nil** indicates that no crosshair cursor exists.
- :grid-on** *t-or-nil* Initialization Option of **gwin:mouse-handler-mixin**
Gettable, settable. Default: **nil**
- Sets a value that indicates whether the grid is on—**t** indicates that the grid is on, and **nil** indicates that the grid is off.
- :grid-x** *distance* Initialization Option of **gwin:mouse-handler-mixin**
Gettable, settable. Default: **nil**
- :grid-y** *distance* Initialization Option of **gwin:mouse-handler-mixin**
Gettable, settable. Default: **nil**
- Sets the distance between grid points along the x or y axis.
- :tracker-cursor** *cursor* Initialization Option of **gwin:mouse-handler-mixin**
Gettable, settable. Default: **nil**
- Enables the mouse cursor.
- :draw-crosshair** *x y* Method of **gwin:mouse-handler-mixin**
- Draws an xor full-screen crosshair centered at the specified point. The area around the center is left blank for the mouse cursor icon.

- :draw-grid** *x-space* Method of **gwin:mouse-handler-mixin**
 &optional (*y-space x-space*) (*alu w:opposite*)
 (*color w:*default-foreground**)
 Draws a grid of dots corresponding to the specified grid spacing. The grid covers the entire window.
 The *color* argument is used only on a color system. The color is XORed with the colors on the screen to obtain the actual color of the grid dots.
- :get-mouse-position** Method of **gwin:mouse-handler-mixin**
 Returns the x and y coordinates of the current mouse position relative to this window.
- :gridify-point** *x y* Method of **gwin:mouse-handler-mixin**
 Returns the x and y coordinates of the grid location nearest the specified point.
- :mouse-click** *button x y* Method of **gwin:mouse-handler-mixin**
 Provides the mouse button interface for this window. This operates like the **:mouse-click** method of the **w:list-mouse-buttons-mixin** flavor, except that the points are in world coordinates. If another window is currently selected, the window that receives this message is selected because the user is interacting with it.
- :mouse-standard-blinker** Method of **gwin:mouse-handler-mixin**
 Sets the mouse blinker for this window. This method is called by the window manager when the mouse enters the screen area occupied by the window.
- :redraw-crosshair** *x y* Method of **gwin:mouse-handler-mixin**
 Erases the last crosshair and draws a new crosshair at the specified point.
- :resume-crosshair** Method of **gwin:mouse-handler-mixin**
:suspend-crosshair Method of **gwin:mouse-handler-mixin**
 Temporarily turns off or turns on the automatic crosshair tracking of the mouse. These methods are used to halt the crosshair while a pop-up menu is displayed.
- :turn-off-crosshair** Method of **gwin:mouse-handler-mixin**
:turn-on-crosshair Method of **gwin:mouse-handler-mixin**
 Turns off or turns on the automatic crosshair tracking of the mouse.
- :undraw-crosshair** Method of **gwin:mouse-handler-mixin**
 Erases the crosshair.
- :update-crosshair** Method of **gwin:mouse-handler-mixin**
 Redraws the crosshair if a crosshair is in the window. The position of the crosshair does not change.

Basic Cursor Mixin 12.8.6 This flavor is a component for cursor flavors that use world coordinates.

gwin:basic-cursor-mixin

Flavor

This mixin must be included in each type of cursor flavor. This mixin includes the basic capabilities for world coordinate cursors.

The origin of the cursor can be offset from the x and y coordinates of the cursor. The offset instance variables determine the offset; the position instance variables are for the coordinates.

:visibility *visibility* Initialization Option of **gwin:basic-cursor-mixin**
Gettable, settable. Default: **t**

Sets the visibility of the cursor. If *visibility* is **nil**, the cursor is not visible.

:window *window* Initialization Option of **gwin:basic-cursor-mixin**
Gettable. Default: **w:selected-window**

Sets the window that the cursor is in.

:x-offset *offset* Initialization Option of **gwin:basic-cursor-mixin**
Gettable. Default: **0**.

:y-offset *offset* Initialization Option of **gwin:basic-cursor-mixin**
Gettable. Default: **0**.

Sets the x or y offset of the cursor. The offsets are the difference between the center of the cursor representation and the coordinates for the cursor.

:x-position *position* Initialization Option of **gwin:basic-cursor-mixin**
Gettable. Default: **0**.

:y-position *position* Initialization Option of **gwin:basic-cursor-mixin**
Gettable. Default: **0**.

Sets the x or y coordinate of the cursor.

:delete Method of **gwin:basic-cursor-mixin**

Deletes the cursor object and removes the blinker from the window.

:offset Method of **gwin:basic-cursor-mixin**

Returns the x and y offsets for the cursor object. The offsets are the difference between the center of the cursor representation and the coordinates for the cursor.

:set-offset *x y* Method of **gwin:basic-cursor-mixin**

Changes the offset of the cursor and redisplay the cursor if necessary. A Boolean value is returned, which indicates whether the cursor is inside the window.

:position Method of **gwin:basic-cursor-mixin**

Returns the x and y coordinates of the cursor object.

:set-position *x y* Method of **gwin:basic-cursor-mixin**

Changes the position of the cursor and redisplay the cursor if necessary. A Boolean value is returned, which indicates whether the cursor is inside the window.

:redraw Method of **gwin:basic-cursor-mixin**
 Displays the cursor at its current location if the cursor is in the window. A Boolean value is returned, which indicates whether the cursor is inside the window.

Cursor Flavor 12.8.7 The graphics window system uses this flavor for character cursors.

gwin:cursor Flavor
 A character cursor that is positioned in world coordinates. The character can be any character from a standard Lisp machine font.

:character *character* Initialization Option of **gwin:cursor**
 Default: 0.

:font *font* Initialization Option of **gwin:cursor**
 Default: **fonts:mouse**
 Sets the character or font, respectively, that defines this cursor object.

:character Method of **gwin:cursor**
:set-character *new-character new-font* Method of **gwin:cursor**
 Returns or sets, respectively, the character and font that define this cursor object.

Bitblt Blinker Flavor 12.8.8 The graphics window system uses this flavor to create bitblt blinkers.

gwin:bitblt-blinker Flavor
 The bitblt type of blinker that allows an ALU. This flavor is similar to **w:bitblt-blinker** except that it does not assume that the blinker is drawn with an xor. This blinker can be positioned outside the window and can clip to window bounds. The blinker uses window coordinates.

:alu *alu* Initialization Option of **gwin:bitblt-blinker**
Gettable. Default: **w:combine**
 Sets the default ALU value for drawing the bitblt blinker.

:array *array* Initialization Option of **gwin:bitblt-blinker**
 Default: **nil**
 Sets a bitblt image. The dimensions of *array* must be multiples of 32.

:height *height* Initialization Option of **gwin:bitblt-blinker**
Gettable. Default: **nil**

:width *width* Initialization Option of **gwin:bitblt-blinker**
Gettable. Default: **nil**

Sets the height or width of the blinker. The value must be a multiple of 32. If you supply a value for *height*, that dimension of *array* is overridden.

:blink Method of **gwin:bitblt-blinker**
 Blinks the bitblt blinker. The blinker is drawn if it is currently off and erased if it is currently on.

- :set-cursorpos** *x y* Method of **gwin:bitblt-blinker**
 Sets the position of the bitblt blinker in window coordinates. The blinker does not follow the cursor after it receives this message. The position can be outside the window bounds.
- :size** Method of **gwin:bitblt-blinker**
 Returns the width and height of the blinker.
-
- Block Cursor Flavor** **12.8.9** The graphics window system uses this flavor for block cursors.
- gwin:block-cursor** Flavor
 A solid, black rectangular cursor that is positioned in world coordinates.
- :height** *height* Initialization Option of **gwin:block-cursor**
Gettable. Default: 10.
- :width** *width* Initialization Option of **gwin:block-cursor**
Gettable. Default: 10.
 Sets the height or width, respectively, of the block cursor object.
- :set-size** *new-width new-height* Method of **gwin:block-cursor**
 Changes the height and width of the block cursor object.
-
- Sprite Cursor Flavor** **12.8.10** The graphics window system uses this flavor to create sprite cursors. For example, the images that are drag-moved or drag-copied in the graphics editor are sprite cursors.
- gwin:sprite-cursor** Flavor
 A cursor that is positioned in world coordinates. This cursor saves the bits that it overwrites so that these bits can be restored.
- Sprite cursors can either reflect (**:reflect**) or wrap (**:wrap**) at each limit depending on which keyword is the value of the **gwin:bottom-flag**, **gwin:right-flag**, **gwin:left-flag**, and **gwin:top-flag** instance variables. If one of these instance variables is **nil**, the sprite cursor continues moving beyond the limit.
- :alu** *alu* Initialization Option of **gwin:sprite-cursor**
 Default: **w:combine**
 Sets the default ALU value for drawing the sprite blinker.
- :array** *array* Initialization Option of **gwin:sprite-cursor**
 Default: 32-by-32 bit array of 1s
 Sets a bitblt image. The dimensions of *array* must be multiples of 32.

- :top-flag** *top-flag* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **:reflect**
- :bottom-flag** *bottom-flag* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **:reflect**
- :left-flag** *left-flag* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **:reflect**
- :right-flag** *right-flag* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **:reflect**

Sets the flag for the specified side. Flags are meaningful only when the cursor is not frozen. Possible values are as follows:

Values	Action Taken When the Sprite Cursor Reaches the Edge of the Window
:reflect	Reverses direction and moves away from that edge.
:wrap	Disappears and reappears on the opposite edge of the window. The sprite cursor continues moving in the same direction it was originally moving in.
nil	Continues moving beyond the limit (that is, off the screen).

- :top-limit** *top-limit* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **nil**
- :bottom-limit** *bottom-limit* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **nil**
- :left-limit** *left-limit* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **nil**
- :right-limit** *right-limit* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **nil**

Sets the limits within which the sprite cursor moves.

- :x-step** *x-step* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: 10.
- :y-step** *y-step* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: 10.

Sets the x and y distances, respectively, that the sprite cursor moves each time.

- :frozen?** *t-or-nil* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: **t**

Sets a Boolean value to indicate whether the sprite cursor moves automatically; for example, the sprite cursor in the starter kit is not frozen, and the sprite cursor for the Drag-Copy and Drag-Move commands in the graphics editor is frozen.

- :height** *height* Initialization Option of **gwin:sprite-cursor**
- :width** *width* Initialization Option of **gwin:sprite-cursor**

Sets the height or width, respectively, of the sprite.

- :time-between-moves** *time* Initialization Option of **gwin:sprite-cursor**
Gettable, settable. Default: 10.
 Sets the number of 60ths of a second that elapse between sprite cursor movements.
- :move** *&optional (dx gwin:x-step) (dy gwin:y-step)* Method of **gwin:sprite-cursor**
 Moves the sprite cursor the specified distance.
- :set-position** *x y* Method of **gwin:sprite-cursor**
 Changes the location of the sprite cursor and redisplay the sprite cursor if necessary. This method does wrapping and reflection if required.
- :redraw** Method of **gwin:sprite-cursor**
 Redraws the sprite cursor. The cursor is clipped to the bounds of the window.
- :size** Method of **gwin:sprite-cursor**
 Returns the width and height of the sprite cursor.
- :update** *delta-time* Method of **gwin:sprite-cursor**
 Moves the sprite cursor by the step values, if enough time has passed.

Standard Operations on Graphic Objects

12.9 The following methods provide operations on most graphic objects. The methods for each object are implemented separately. For conciseness in documentation, however, the typical methods are discussed in detail here. Only the nonstandard operations and the differences between the standard operation and the operation of an individual method are discussed with the particular object.

In addition to these standard operations, all methods of **gwin:basic-graphics-mixin** also apply.

- :distance** *from-x from-y* Method of *graphic object*
 Returns the minimum distance from the specified point to the edge of the object. If the specified point is in the interior of the object, the returned value is negative, and the absolute value of the returned value is the distance to the edge of the object.
- :draw** *window* Method of *graphic object*
 Draws the entire object entity in the specified window. The typical use for **:draw** is illustrated in the starter kit. Usually, you send this method to the returned value of an **:insert-object** method.
- :edge-point** *window &optional transform* Method of *graphic object*
 Returns the coordinates for a point on the edge of the object for highlighting. This method is defined separately for each graphic object. If *transform* is non-nil, the object is part of a subpicture, and *transform* holds the subpicture translations.

:edit-parameters Method of *graphic object*

Displays a menu of the object parameters and allows editing of the parameters. The method returns either a non-nil value if the parameters were edited or nil if the parameters were not edited.

In a color environment, you can specify any color from the color map for edge and fill colors. Clicking on either the edge color or fill color parameters displays a menu of choices for specifying a color. You can choose from a list of standard color names, select a color from the color map display, or enter an integer value for a color.

Clicking on Draw ALU displays a list of standard ALU operations plus additional color ALU operations.

NOTE: The default ALU operations work differently in a color environment, as explained in paragraph 19.6, Color ALU Functions.

:fasd-form Method of *graphic object*

Returns a form that recreates this object instance when the form is evaluated. This message is sent to an object when the object is being written in compiled form to a file.

:move *delta-x delta-y* Method of *graphic object*

Moves this object the specified distances in the x and y directions and updates the extents.

:scale *sx* &optional (*sy sx*) (*scale-thickness?* *t*) Method of *graphic object*

Scales this object instance by the specified factors. For example, scaling by 2 doubles the size of the object, and scaling by 0.5 makes the object half as large. The extents of the world are **not** updated.

:undraw *window* Method of *graphic object*

Erases the entire object entity. You should use the **:refresh-area** method of **graphics-window-mixin** to refresh the area of the video display that contained the object.

Basic Graphics Mixin

12.10 This flavor is the component for all of the graphics object flavors used with worlds.

If you add a new graphics object, that object should be able to handle all of the methods for **gwin:basic-graphics-mixin** and for the standard methods defined for all graphic objects, described in paragraph 12.9, Standard Operations on Graphic Objects. If you create a new object, you may need to redefine some of the standard methods defined in **gwin:basic-graphics-mixin** to make the methods work properly for the new object flavor.

gwin:basic-graphics-mixin Flavor

This mixin includes the methods and instance variables that are common to all graphics object flavors.

:alu <i>alu</i>	Initialization Option of gwin:basic-graphics-mixin <i>Gettable, settable</i> . Default: w:normal
	Sets the ALU used by default for drawing this type of graphic object. Several values are given in Table 12-2, ALU Values for Graphic Methods. For information on ALU values in a color environment, refer to paragraph 19.6, Color ALU Functions.
:copy	Method of gwin:basic-graphics-mixin Returns a copy of the object.
:edge-color <i>color</i>	Initialization Option of gwin:basic-graphics-mixin <i>Gettable, settable</i> . Default: w:black
:fill-color <i>color</i>	Initialization Option of gwin:basic-graphics-mixin <i>Gettable, settable</i> . Default: nil
	Sets the edge color or fill color, respectively, for the object. Edge color is the color of the edge of the object; fill color is the color of the interior of the object. Possible values for <i>color</i> in a monochrome environment are given in Table 12-1, Color Values for Graphic Methods for Monochrome Environments. Possible values for <i>color</i> in a color environment include the value or name of any color from the color map. The system-defined color names are given in Table 19-2, Named Colors in the Default Color Map.
:extents	Method of gwin:basic-graphics-mixin Returns four values: the minimum x and y and maximum x and y values that completely include the object.
:highlight <i>window</i>	Method of gwin:basic-graphics-mixin
:highlight-2 <i>window</i>	Method of gwin:basic-graphics-mixin Highlights the object in this window, :highlight points a hand at the object; :highlight-2 makes the object blink.
:inside-p <i>left top right bottom</i>	Method of gwin:basic-graphics-mixin Returns a Boolean value indicating whether the object is completely inside the rectangle.
:markers	Method of gwin:basic-graphics-mixin Returns the list of associated cursors.
:nearest-x <i>x</i>	Method of gwin:basic-graphics-mixin <i>Gettable, settable</i> .
:nearest-y <i>y</i>	Method of gwin:basic-graphics-mixin <i>Gettable, settable</i> . Returns the appropriate coordinate of the mouse cursor.
:origin	Method of gwin:basic-graphics-mixin Returns two values: the minimum x and y coordinates. These are also the coordinates for the upper left corner of the extents.
:outside-p <i>left top right bottom</i>	Method of gwin:basic-graphics-mixin Returns a Boolean value indicating whether the object is completely outside the specified rectangle.

- :overlap-p** *entity* Method of **gwin:basic-graphics-mixin**
Returns a Boolean value indicating whether the extents of this object and the specified entity overlap.
- :set-origin** *origin-x origin-y* Method of **gwin:basic-graphics-mixin**
Moves the graphics object so that the upper left corner of the object is at the specified point.
- :unhighlight** *window* Method of **gwin:basic-graphics-mixin**
Removes the highlighting from this object in the specified window.
- :x-max** *x-max* Method of **gwin:basic-graphics-mixin**
Gettable, settable. Default: nil
- :x-min** *x-min* Method of **gwin:basic-graphics-mixin**
Gettable, settable. Default: nil
- :y-max** *y-max* Method of **gwin:basic-graphics-mixin**
Gettable, settable. Default: nil
- :y-min** *y-min* Method of **gwin:basic-graphics-mixin**
Gettable, settable. Default: nil
- Returns the value of the respective extent of the graphic object. The extents of the graphics object are as follows:

Instance Variable	Extent
gwin:x-min	left
gwin:x-max	right
gwin:y-min	top
gwin:y-max	bottom

Simple Graphic Objects

12.11 The following flavors and methods manipulate simple graphic objects: arcs, circles, lines, polylines, rectangles, splines, and triangles.

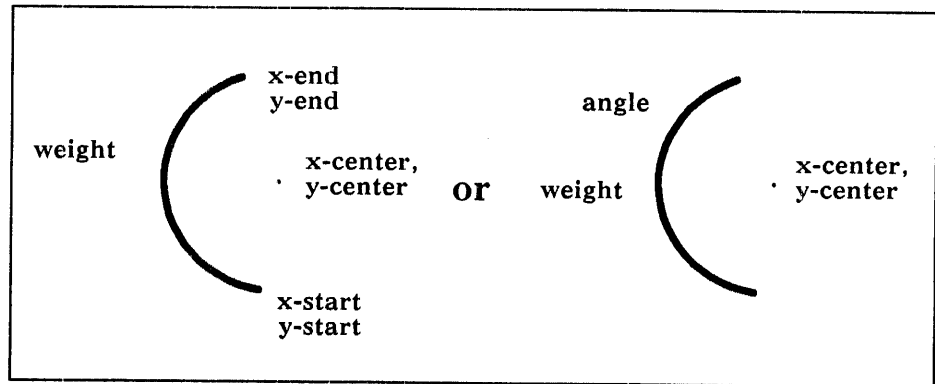
Arc Flavor 12.11.1

gwin:arc

Flavor

The arc graphics entity. An arc is a piece of a circle. An arc can have a fill color; if it has a fill color, the area inside the including angle is solid.

Creating the Object 12.11.1.1 The following initialization options and methods define an arc. All of these update the extents of the object when set or initialized. To create an arc, you must specify the center, the starting point, and either the angle or the ending point of the arc.



- :angle** *angle* Initialization Option of gwin:arc
Gettable, settable. Default: 360.
 Sets the degree of the including angle.
- :weight** *weight* Initialization Option of gwin:arc
Gettable, settable. Default: 1.
 Sets the line thickness for the arc edge.
- :x-center** *x-center* Initialization Option of gwin:arc
Gettable, settable. Default: 20.
- :y-center** *y-center* Initialization Option of gwin:arc
Gettable, settable. Default: 20.
 Sets the x or y coordinate of the center of the including circle.
- :x-start** *x-start* Initialization Option of gwin:arc
Gettable, settable. Default: 20.
- :y-start** *y-start* Initialization Option of gwin:arc
Gettable, settable. Default: 0.
 Sets the x or y coordinate of the start of the arc. Arcs are drawn counter-clockwise from the starting point.
- :x-end** *x-end* Initialization Option of gwin:arc
:y-end *y-end* Initialization Option of gwin:arc
 Sets the x or y coordinate of the end of the arc.

Manipulating the Object 12.11.1.2 The following standard methods are defined for arcs. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

<code>:distance from-x from-y</code>	Method of <code>gwin:arc</code>
<code>:draw window</code>	Method of <code>gwin:arc</code>
<code>:edit-parameters</code>	Method of <code>gwin:arc</code>
<code>:fasd-form</code>	Method of <code>gwin:arc</code>
<code>:move delta-x delta-y</code>	Method of <code>gwin:arc</code>
<code>:scale sx &optional (sy sx) (scale-thickness? t)</code>	Method of <code>gwin:arc</code>
<code>:undraw window</code>	Method of <code>gwin:arc</code>
<code>:edge-point window &optional transform</code>	Method of <code>gwin:arc</code>

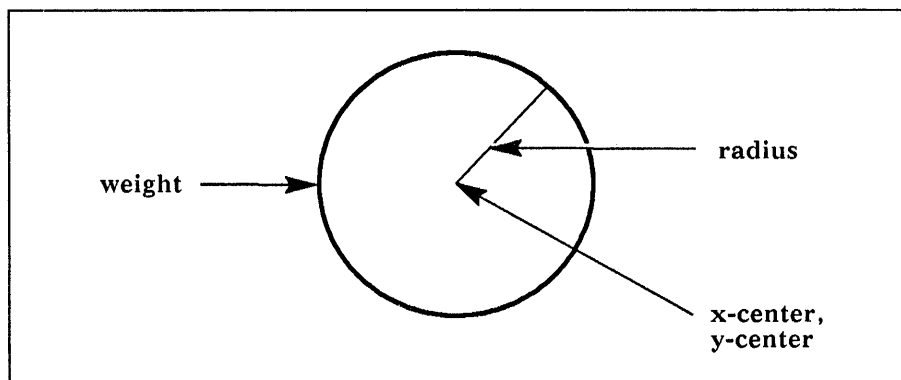
Returns the coordinates for a point on the edge of the arc for highlighting. If the midpoint of the edge of the arc is inside the window, that point is returned. Otherwise, the point on the arc that is nearest to the mouse cursor is returned. If *transform* is non-nil, the arc is part of a subpicture, and *transform* holds the subpicture translations.

Circle Flavor 12.11.2

`gwin:circle` Flavor

The definition of the circle graphics entity. A circle can have a fill color, which causes the circle to be solid.

Creating the Object 12.11.2.1 The following initialization options and methods define a circle. All of these update the extents of the object when set or initialized.



<code>:radius radius</code>	Initialization Option of <code>gwin:circle</code>
<i>Gettable, settable.</i> Default: 20.	
Sets the radius of this circle instance.	
<code>:weight weight</code>	Initialization Option of <code>gwin:circle</code>
<i>Gettable, settable.</i> Default: 2.	
Sets the line thickness of the circle.	
<code>:x-center x-center</code>	Initialization Option of <code>gwin:circle</code>
<i>Gettable, settable.</i> Default: 0.	
<code>:y-center y-center</code>	Initialization Option of <code>gwin:circle</code>
<i>Gettable, settable.</i> Default: 0.	
Sets the x or y coordinate of the center of the circle.	

Manipulating the Object 12.11.2.2 The following standard methods are defined for circles. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

<code>:distance from-x from-y</code>	Method of <code>gwin:circle</code>
<code>:draw window</code>	Method of <code>gwin:circle</code>
<code>:edit-parameters</code>	Method of <code>gwin:circle</code>
<code>:fasd-form</code>	Method of <code>gwin:circle</code>
<code>:move dx dy</code>	Method of <code>gwin:circle</code>
<code>:scale sx &optional (sy sx) (scale-thickness? t)</code>	Method of <code>gwin:circle</code>
<code>:undraw window</code>	Method of <code>gwin:circle</code>
<code>:edge-point window &optional transform</code>	Method of <code>gwin:circle</code>

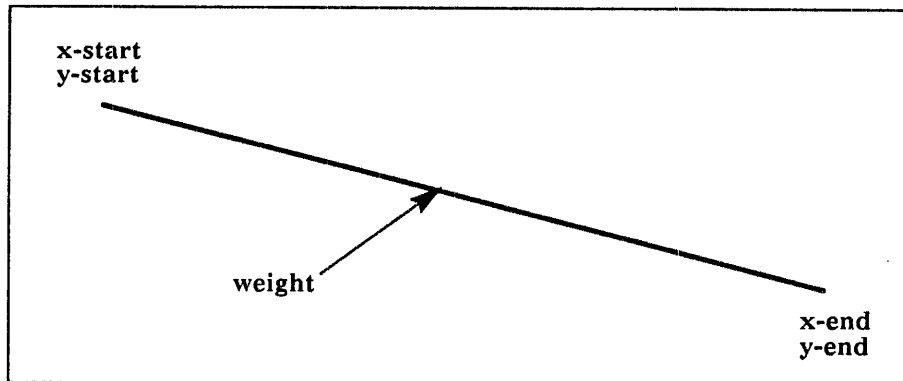
Returns a point to be used for highlighting. The point on the rightmost outside edge is returned unless that point is out of the window; when that point is out of the window, the point closest to the mouse is returned. If *transform* is non-nil, this circle is part of a subpicture, and *transform* holds the subpicture translations.

Line Flavor 12.11.3

`gwin:line` Flavor

The line graphics entity.

Creating the Object 12.11.3.1 The following initialization options and methods define a line. All of these update the extents of the object when set or initialized.



<code>:weight weight</code>	Initialization Option of <code>gwin:line</code>
<i>Gettable, settable.</i> Default: 2.	
Sets the line thickness to a new value.	
<code>:x-end x-end</code>	Initialization Option of <code>gwin:line</code>
<i>Gettable, settable.</i> Default: 100.	
<code>:x-start x-start</code>	Initialization Option of <code>gwin:line</code>
<i>Gettable, settable.</i> Default: 50.	
<code>:y-end y-end</code>	Initialization Option of <code>gwin:line</code>
<i>Gettable, settable.</i> Default: 100.	
<code>:y-start y-start</code>	Initialization Option of <code>gwin:line</code>
<i>Gettable, settable.</i> Default: 50.	
Sets the x or y coordinate of either the start or end of a line.	

Manipulating the Object 12.11.3.2 The following standard methods are defined for lines. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

<code>:distance from-x from-y</code>	Method of <code>gwin:line</code>
<code>:draw window</code>	Method of <code>gwin:line</code>
<code>:edit-parameters</code>	Method of <code>gwin:line</code>
<code>:fasd-form</code>	Method of <code>gwin:line</code>
<code>:move dx dy</code>	Method of <code>gwin:line</code>
<code>:scale sx &optional (sy sx) (scale-thickness? t)</code>	Method of <code>gwin:line</code>
<code>:undraw window</code>	Method of <code>gwin:line</code>
<code>:edge-point window &optional transform</code>	Method of <code>gwin:line</code>

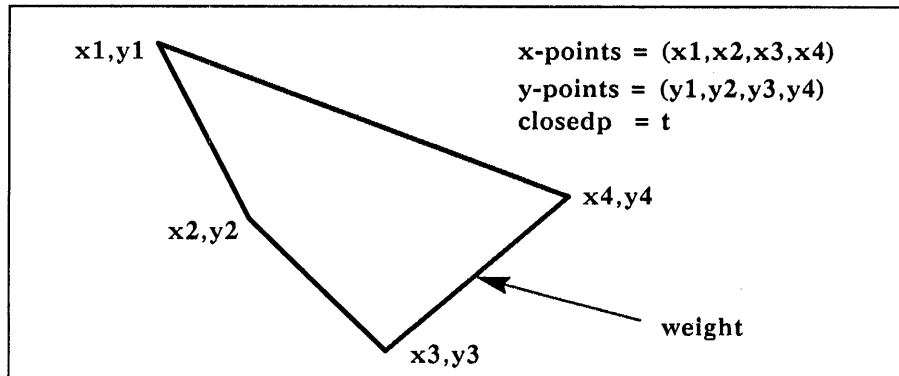
Returns the coordinates for a point on the edge of the line for highlighting. If the midpoint of the edge of the line is inside the window, that point is returned. Otherwise, the point on the line that is nearest to the mouse cursor is returned. If *transform* is non-`nil`, the line is part of a subpicture, and *transform* holds the subpicture translations.

Polyline Flavor 12.11.4

`gwin:polyline` Flavor

Polyline graphics entity. A polyline is a connected set of line segments. If the polyline has a fill color, the polyline is solid inside the polygon that is defined by the polyline and a line segment connecting the starting and ending points.

Creating the Object 12.11.4.1 The following initialization options and methods define a polyline. All of these update the extents of the object when set or initialized.



`:closedp t-or-nil` Initialization Option of `gwin:polyline`
Gettable. Default: `nil`

Sets a Boolean value that indicates whether the polyline is closed. If `:closedp` is `t`, the polyline is closed regardless of whether the first and last points coincide. Similarly, if `:closedp` is `nil`, the polyline is *not* closed. Even if the first and last points coincide, no line segment is drawn back to the first segment.

`:weight weight` Initialization Option of `gwin:polyline`
Gettable, settable. Default: 2.

Sets the line thickness of the polyline.

:x-points *x-points* Initialization Option of **gwin:polyline**
Gettable. Default: (100. 500.)

:set-x-points *new-x-points* &optional (*update-self* *t*) Method of **gwin:polyline**

:y-points *y-points* Initialization Option of **gwin:polyline**
Gettable. Default: (100. 500.)

:set-y-points *new-y-points* &optional (*update-self* *t*) Method of **gwin:polyline**

Initializes the set of x or y coordinates for the polyline. **:set-x-points** and **:set-y-points** define new coordinates and, if *update-self* is non-nil, updates the polyline instance.

Manipulating the Object 12.11.4.2 The following standard methods are defined for polylines. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance *from-x from-y* Method of **gwin:polyline**

:draw *window* Method of **gwin:polyline**

:edit-parameters Method of **gwin:polyline**

:fasd-form Method of **gwin:polyline**

:move *dx dy* Method of **gwin:polyline**

:scale *sx* &optional (*sy sx*) (*scale-thickness?* *t*) Method of **gwin:polyline**

:undraw *window* Method of **gwin:polyline**

:edge-point *window* &optional *transform* Method of **gwin:polyline**

Returns a point to be used for highlighting. The midpoints of the line segments are checked in order from the first line through the last line until one in the window is found. The coordinates of that point are returned. If *transform* is non-nil, this polyline is part of a subpicture, and *transform* holds the subpicture translations.

:copy Method of **gwin:polyline**

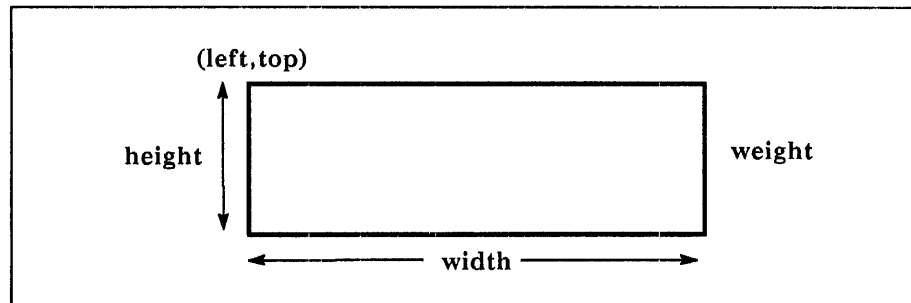
Returns a copy of the polyline. This overrides the **:copy** method from **gwin:basic-graphics-mixin**.

Rectangle Flavor 12.11.5

gwin:rectangle Flavor

The rectangle graphics entity. The rectangle can have a fill color, which causes the interior to be solid.

Creating the Object 12.11.5.1 The following initialization options and methods define a rectangle. All of these update the extents of the object when set or initialized.



:height <i>height</i>	Initialization Option of gwin:rectangle
<i>Gettable, settable. Default: 100.</i>	
:width <i>width</i>	Initialization Option of gwin:rectangle
<i>Gettable, settable. Default: 100.</i>	
	Sets the rectangle height or width.
:left <i>left</i>	Initialization Option of gwin:rectangle
<i>Gettable, settable. Default: 0.</i>	
:top <i>top</i>	Initialization Option of gwin:rectangle
<i>Gettable, settable. Default: 0.</i>	
	Sets the left or top edge of the rectangle.
:weight <i>weight</i>	Initialization Option of gwin:rectangle
<i>Gettable, settable. Default: 2.</i>	
	Sets the line thickness of the rectangle.

Manipulating the Object **12.11.5.2** The following standard methods are defined for rectangles. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

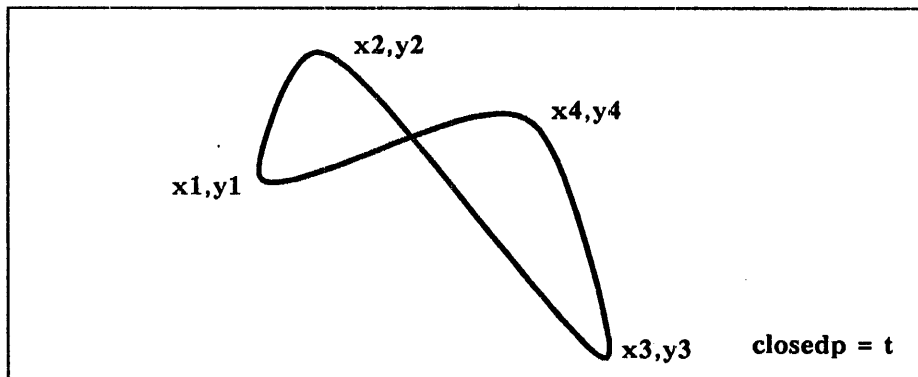
:distance <i>from-x from-y</i>	Method of gwin:rectangle
:draw <i>window</i>	Method of gwin:rectangle
:edit-parameters	Method of gwin:rectangle
:fasd-form	Method of gwin:rectangle
:move <i>dx dy</i>	Method of gwin:rectangle
:scale <i>sx &optional (sy sx) (scale-thickness? t)</i>	Method of gwin:rectangle
:undraw <i>window</i>	Method of gwin:rectangle
:edge-point <i>window &optional transform</i>	Method of gwin:rectangle
	Returns the x and y coordinates of a point that is used for highlighting. The midpoint of the outside edge of the right side is returned unless that point is out of the window; when that point is out of the window, the point closest to the mouse is returned. If <i>transform</i> is non-nil, this rectangle is part of a subpicture, and <i>transform</i> holds the subpicture translations.

Spline Flavor 12.11.6**gwin:spline**

Flavor

The spline graphics entity. A spline is a smooth curve connecting a set of points. If the spline has a fill color, it is solid inside the closed spline or inside the area that is defined by the open spline and a straight line connecting the starting and ending points.

Creating the Object 12.11.6.1 The following initialization options and methods define a spline. All of these update the extents of the object when set or initialized.

**:closedp**Initialization Option of **gwin:spline**

Default: nil

Sets a Boolean value that indicates whether the spline is closed. If **:closedp** is **t**, the spline is closed regardless of whether the first and last points coincide. Similarly, if **:closedp** is **nil**, the spline is *not* closed. Even if the first and last points coincide, the curve connecting the two is not drawn.

:weight weightInitialization Option of **gwin:spline***Settable*. Default: 2.

Sets the line thickness of the spline.

:x-points x-pointsInitialization Option of **gwin:spline***Settable*. Default: (0 50 100 150 200)**:y-points y-points**Initialization Option of **gwin:spline***Settable*. Default: (50 0 50 100 50)

Sets a list of the knots—the x or y coordinates—that are used to specify the spline. The argument can be either a list or an array.

Manipulating the Object

12.11.6.2 The spline flavor includes standard methods and methods that return the computed points for the spline.

The following standard methods are defined for splines. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance *from-x from-y* Method of **gwin:spline**
:draw *window* Method of **gwin:spline**
:edit-parameters Method of **gwin:spline**
:fasd-form Method of **gwin:spline**
:move *dx dy* Method of **gwin:spline**
:scale *sx &optional (sy sx) (scale-thickness? t)* Method of **gwin:spline**
:undraw *window* Method of **gwin:spline**

:copy Method of **gwin:spline**

Returns a copy of the spline. This overrides the **:copy** method from **basic-graphics-mixin**.

:edge-point *window &optional transform* Method of **gwin:spline**

Returns the x and y coordinates for the first knot inside the window that is used for highlighting. If *transform* is non-**nil**, this spline is part of a subpicture, and *transform* holds the subpicture translations.

:curve-x-points *curve-x-points* Method of **gwin:spline**

:curve-y-points *curve-y-points* Method of **gwin:spline**

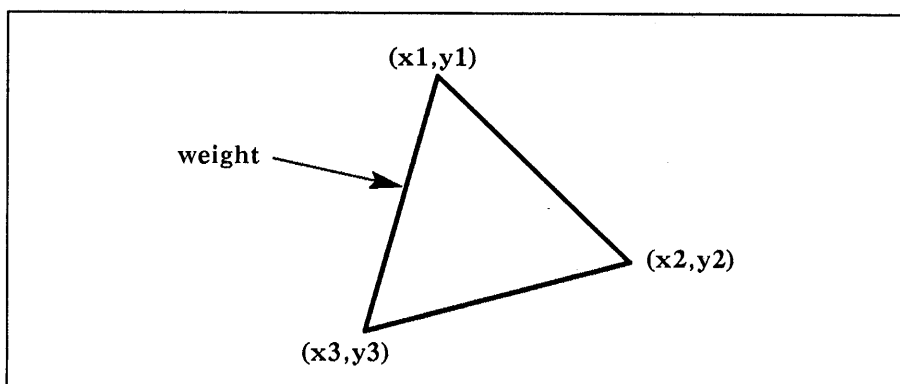
Returns a list of the computed x or y coordinates, as appropriate, of the points that are used to draw the spline. The list includes both the specified coordinates (the *knots*) and the computed coordinates that lie between the knots.

Triangle Flavor 12.11.7

gwin:triangle Flavor

The triangle graphics entity. If the triangle has a fill color, it is solid inside the perimeter.

Creating the Object 12.11.7.1 The following initialization options and methods define a triangle. All of these update the extents of the object when set or initialized.



:weight *weight* Initialization Option of **gwin:triangle**

Gettable, settable. Default: 2.

Sets the line thickness of the triangle.

:x1 x1	Initialization Option of gwin:triangle
:y1 y1	Initialization Option of gwin:triangle
:x2 x2	Initialization Option of gwin:triangle
:y2 y2	Initialization Option of gwin:triangle
:x3 x3	Initialization Option of gwin:triangle
:y3 y3	Initialization Option of gwin:triangle
	<i>Gettable, settable. Default: 0.</i>
	Sets the coordinates of the vertices of the triangle.

Manipulating the Object **12.11.7.2** The following standard methods are defined for triangles. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance from-x from-y	Method of gwin:triangle
:draw window	Method of gwin:triangle
:edit-parameters	Method of gwin:triangle
:fasd-form	Method of gwin:triangle
:move dx dy	Method of gwin:triangle
:scale sx &optional (sy sx) (scale-thickness? t)	Method of gwin:triangle
:undraw window	Method of gwin:triangle
:edge-point window &optional transform	Method of gwin:triangle

Returns the x and y coordinates for a point to be used for highlighting. The midpoint of the outside edge of the first side is returned if it is inside the window. Otherwise, the closest point to the mouse cursor is returned. If *transform* is non-nil, this triangle is part of a subpicture, and *transform* holds the subpicture translations.

Fonts and Text Objects

12.12 Fonts used to create text objects are graphic objects called *raster fonts*. As such, they can be scaled (enlarged or reduced) and they can be displayed in various fill colors and background colors. As graphic objects, font objects in the graphics window system are different from the font objects typically used for output on windows. When you create the graphic window system, it automatically generates raster fonts analogous to the fonts used elsewhere in the window system. A list of all the loaded raster fonts is stored in the **w:*font-list*** variable.

For example, the graphics font object **gwin:cptfont-font** is analogous to the font object **fonts:cptfont**. You should specify raster fonts as font arguments when working with graphics objects.

w:*font-list*	Variable
	A list of all raster fonts currently loaded in the system. This list is in a form suitable for a menu item list.
w:*default-w-fonts*	Variable
	Default: w:medfndb-font , w:cptfont-font
	A list of all raster fonts available before the GWIN package was created.

gwin:*default-gwin-fonts* Variable
 A list of all raster fonts that were created automatically when the GWIN package was created.

Font Flavor 12.12.1 Font flavor is used for fonts in the graphics window system.

gwin:font Flavor
 Used for fonts in the graphics window system. A font can use any type of character objects that can be scaled.

:blinker-height *height* Initialization Option of **gwin:font**
Gettable, settable. Default: nil

:blinker-width *width* Initialization Option of **gwin:font**
Gettable, settable. Default: nil

Sets the height and width of the block cursor that is used during text input.

:character-size *char-index* &optional (*create-scale* 1) Method of **gwin:font**
 Returns two values: the horizontal and vertical spacing for the specified character.

:characters *characters* Initialization Option of **gwin:font**
Gettable, settable. Default: An empty 256-element array
 Sets the array of characters.

:draw-character *char-index x y window* Method of **gwin:font**
 &optional (*color w:black*) (*create-scale* 1) (*alu w:combine*)
 Draws a character in this font in the specified window. The returned values are the x and y increments for the cursor.

:draw-string *text x y window* Method of **gwin:font**
 &optional (*color w:black*) (*start* 0) (*end (length text)*) (*create-scale* 1.)
 (*alu w:combine*)
 Draws a string of characters in this font in the specified window. The returned values are the x and y increments for the cursor.

:fasd-form Method of **gwin:font**
 Returns a form that recreates this font when evaluated. This message is sent to an object when the object is being written in compiled form to a file.

:horz-spacing *spacing* Initialization Option of **gwin:font**
Gettable, settable. Default: 10.
 Sets the minimum number of pixels that the cursor moves horizontally when a character is drawn.

:set-character *char-index object* Method of **gwin:font**
 Changes the definition of a character within this font.

:vert-spacing *spacing* Initialization Option of **gwin:font**
Gettable, settable. Default: 12.
 Sets the spacing between lines of text in this font.

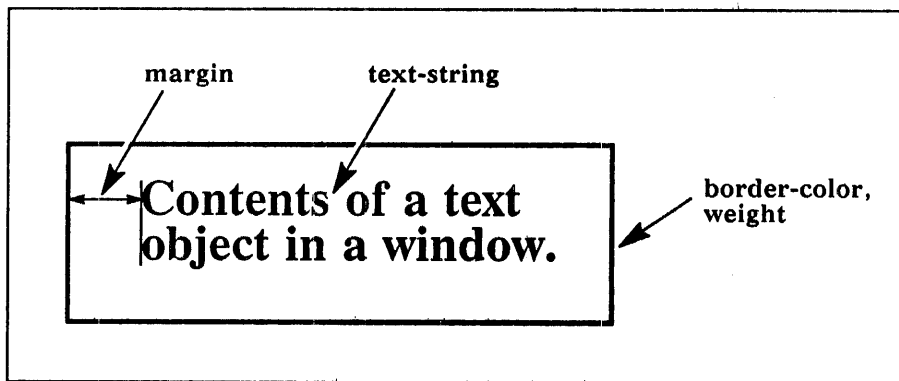
Text Flavor 12.12.2

gwin:text

Flavor

The text graphics entity. A text entity is a string of characters that is contained in a rectangle. If the text entity has a fill color, the characters are drawn on a rectangle of the specified fill color. If the text entity has a border color, a rectangular border of that color is drawn around the text entity.

Creating the Object 12.12.2.1 The following initialization options and methods define a text object. All of these update the extents of the object when set or initialized.



- :border-color** *border-color* Initialization Option of gwin:text
Gettable, settable. Default: nil
 Sets the color of the border that surrounds the text.
- :font-name** *font-name* Initialization Option of gwin:text
Gettable, settable. Default: gwin:standard-font
 Sets the font name used for the text string. *font-name* must be a GWIN font object, such as w:medfnb-font rather than fonts:medfnb.
- :margin** Initialization Option of gwin:text
Gettable, settable. Default: 3.
 Sets the width in world coordinates of the white space between the text object and the border of the rectangle that surrounds it.
- :tab-width** *tab-width* Initialization Option of gwin:text
Gettable, settable. Default: 8.
 Returns the number of spaces that are inserted in a text object when the TAB key is pressed.
- :text-string** *text-string* Initialization Option of gwin:text
Gettable. Default: An empty 5-character string array
- :set-text-string** *string &optional (update-self t)* Method of gwin:text
 Sets the string that actually appears in the text object. **:set-text-string** sets *text-string* to a new value and updates the text object if *update-self* is non-nil.

:weight *weight* Initialization Option of `gwin:text`
Gettable, settable. Default: 2.
 Sets the line thickness for the border of the text.

:x-end *x-end* Initialization Option of `gwin:text`
Gettable, settable.

:y-end *y-end* Initialization Option of `gwin:text`
Gettable, settable.
 Sets the x or y coordinate of the end of the text entity `:set-x-end` or `:set-y-end` moves the text entity so that it ends at a new coordinate.

:x-start *x-start* Initialization Option of `gwin:text`
Gettable, settable. Default: 0.

:y-start *y-start* Initialization Option of `gwin:text`
Gettable, settable. Default: 0.
 Sets the appropriate starting coordinate of the text entity.

Manipulating the Object 12.12.2.2 The text flavor includes standard methods, as well as other methods unique to text objects.

Standard Methods The following standard methods are defined for texts. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance *from-x from-y* Method of `gwin:text`
:draw *window* Method of `gwin:text`
:edit-parameters Method of `gwin:text`
:fasd-form Method of `gwin:text`
:move *dx dy* Method of `gwin:text`
:scale *sx &optional (sy sx) (scale-thickness? t)* Method of `gwin:text`
:undraw *window* Method of `gwin:text`

:copy Method of `gwin:text`
 Returns a copy of the text entity. This overrides the `:copy` method from `gwin:basic-graphics-mixin`.

:edge-point *window &optional transform* Method of `gwin:text`
 Returns the x and y coordinates for a point that is used for highlighting. The end point of the last character is returned if that point is in the window. Otherwise, the point closest to the mouse cursor is returned. If *transform* is non-nil, this text entity is part of a subpicture, and *transform* holds the subpicture translations.

Other Methods The following methods are also defined for text objects.

:append *new-chars window* Method of `gwin:text`
 Appends characters to the text entity.

:chop *n window* Method of `gwin:text`
 Removes the *n*th and all following characters from this text object.

Basic Character Mixin 12.12.3 The graphics window system uses this flavor as a basis for both vector and raster characters. In most cases, which type of character you use depends on whether the character is normally used with a transformation that makes the character appear larger than it was originally defined to be. Vector characters look better when you zoom in on them; raster characters look better when viewed at their original transformation.

gwin:basic-character-mixin Flavor

This mixin contains common data needed by all types of characters.

:horizontal-spacing *spacing* Initialization Option of **gwin:basic-character-mixin**
Gettable, settable. Default: nil

Returns the horizontal spacing of this character only. The value is the distance between this character and the preceding character; if the value is nil, the standard spacing for the font is used.

:thickness *thickness* Initialization Option of **gwin:basic-character-mixin**
Gettable, settable. Default: 0.

Sets the thickness of the font.

:vertical-spacing *spacing* Initialization Option of **gwin:basic-character-mixin**
Gettable, settable. Default: nil

Returns the vertical spacing of this character only. The value is the distance between this character and the line above the character; if the value is nil, the standard spacing for the font is used.

Vector Character Flavor 12.12.4

gwin:vector-character Flavor

A character defined by a set of ordered coordinates that specify thick lines. The representation of a vector character has two arrays: one contains the x coordinates for the end points of the lines, and the other contains the y coordinates. The origin of these coordinates is the upper left corner of the character box.

:draw *x y window &optional color (alu w:combine)* Method of **gwin:vector-character**

Draws a vector character at the specified point in the specified window. First, the character cache is checked for a bit-map image of the correct character in the correct size. If the image is correct, it is bitblt'd into the window. Otherwise, the character is drawn in the cache and then bitblt'd into the window. The scales for the cache window must be set to the correct drawing scale for this character; you should use the **:set-scales** method in **w:cache-window** flavor.

In a monochrome environment, the default value of *color* is **w:black**. In a color environment, the default value of *color* is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing).

- :fasd-form** Method of **gwin:vector-character**
Returns a form that recreates this vector character instance when evaluated. This message is sent to an object when an object is being written in compiled form to a file.
- :x-points** *x-points* Initialization Option of **gwin:vector-character**
Gettable. Default: None
- :y-points** *y-points* Initialization Option of **gwin:vector-character**
Gettable. Default: None
- Sets a list of the x or y coordinates for the character.

Raster Character Flavor 12.12.5

- gwin:raster-character** Flavor
- A character that is defined by a raster bit pattern. A raster character includes a two-dimensional array, **gwin:identity-cache**, of the points that defines the character. The dimensions of the character in this array are **gwin:identity-height** and **gwin:identity-width**.
- :draw** *x y window &optional color (alu w:combine)* Method of **gwin:raster-character**
Draws a raster character at the specified point in the specified window. First the character cache is checked for a bitmap image of the correct character in the correct size. If the image is correct, it is bit block transferred (bitbltd) into the window. Otherwise, the character is drawn in the cache and then bitbltd into the window. The scales for the cache window must be set to the correct drawing scale for this character; you should use the **:set-scales** method in the **w:cache-window** flavor.
- In a monochrome environment, the default value of *color* is **w:black**. In a color environment, the default value of *color* is the value of **w:sheet-foreground-color** (the foreground color of the window to which you are drawing).
- :fasd-form** Method of **gwin:raster-character**
Returns a form that recreates the raster character when evaluated. This message is sent to an object when the object is being written in compiled form to a file.
- :identity-cache** *cache* Initialization Option of **gwin:raster-character**
Gettable. Default: An empty **cache-size-by-cache-size** pixel array
- :identity-height** *height* Initialization Option of **gwin:raster-character**
Gettable. Default: 0.
- Sets the zeroth dimension of the array **gwin:identity-cache**.
- :identity-width** *width* Initialization Option of **gwin:raster-character**
Gettable. Default: 0.
- Sets the first dimension of the array **gwin:identity-cache**.

:left-kern *left-kern* Initialization Option of **gwin:raster-character**
Gettable. Default: 0.
 Sets the size of the overlap between this character and the previous character. See the description of **w:font-left-kern-table** in paragraph 9.8, Format of Fonts, for a detailed description of left kerning.

Ruler Flavor

12.13 The graphics window system uses this flavor to create rulers.

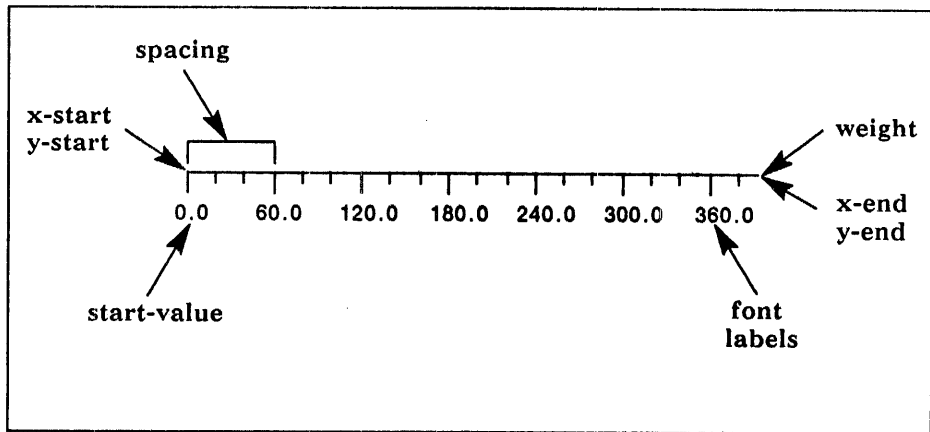
gwin:ruler

Flavor

The ruler graphics entity. A ruler is always parallel to one of the edges of the window, but it can increment from either end.

Creating the Object

12.13.1 The following instance variables, initialization options, and methods define a ruler.



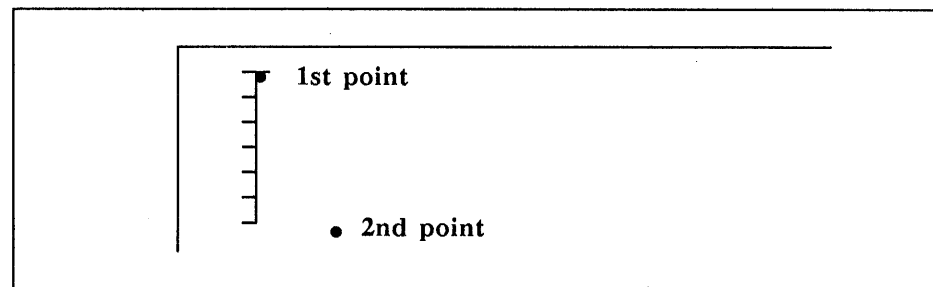
- :font** *font* Initialization Option of **gwin:ruler**
 Default: **fonts:cptfont**
 Sets the font for the labels for the tick marks.
- :labels** *labels* Initialization Option of **gwin:ruler**
Gettable. Default: **nil**
 Sets a list of real numbers used to label the tick marks.
- :spacing** *spacing* Initialization Option of **gwin:ruler**
 Default: 20.
 Sets the spacing between tick points.
- :start-value** *start-value* Initialization Option of **gwin:ruler**
 Default: 0.
 Sets the starting value for the tick marks.
- gwin:tick-x-points** Instance Variable of **gwin:ruler**
- gwin:tick-y-points** Instance Variable of **gwin:ruler**
 Sets a list of the x or y coordinates for the tick marks.

:weight <i>weight</i>	Initialization Option of gwin:ruler
<i>Gettable, settable.</i> Default: 2.	
Sets the line thickness for the ruler.	
:x-start <i>x-start</i>	Initialization Option of gwin:ruler
Default: 50.	
:y-start <i>y-start</i>	Initialization Option of gwin:ruler
Default: 50.	
:x-end <i>x-end</i>	Initialization Option of gwin:ruler
Default: 100.	
:y-end <i>y-end</i>	Initialization Option of gwin:ruler
Default: 100.	

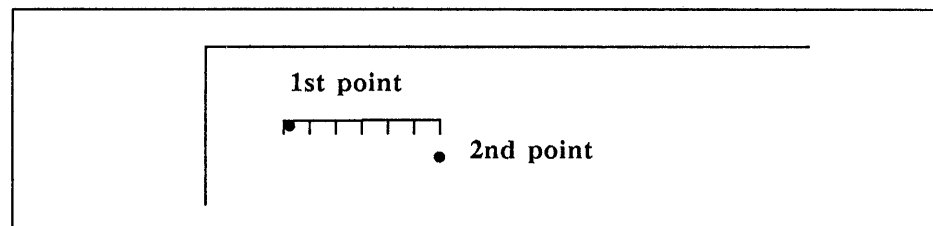
Sets the approximate x and y coordinates for the starting and ending points. If the line defined by *x-start,y-start* and *x-end,y-end* is not parallel to one of the sides of the window, the end points of the ruler are adjusted to make it parallel, as follows:

- If the absolute value of the difference between *x-start* and *x-end* is greater than the absolute value of the difference between *y-start* and *y-end*, the ruler is parallel to the horizontal sides of the window, and the length of the ruler is the absolute value of the difference between *x-start* and *x-end*.
- Otherwise, the ruler is parallel to the vertical sides of the window, and the length of the ruler is the absolute value of the difference between *y-start* and *y-end*.

For example, with the following end points, the ruler would appear as follows:



The ruler appears with its starting point at the first point drawn. Similarly:



Manipulating the Object 12.13.2 The following standard methods are defined for rulers. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance *from-x from-y* Method of **gwin:ruler**
:draw *window* Method of **gwin:ruler**
:edit-parameters Method of **gwin:ruler**
:fasd-form Method of **gwin:ruler**
:move *dx dy* Method of **gwin:ruler**
:scale *sx &optional (sy sx) ignore* Method of **gwin:ruler**
:undraw *window* Method of **gwin:ruler**

:edge-point *window &optional transform* Method of **gwin:ruler**

Returns the x and y coordinates for a point that is used for highlighting. The lower right corner of the extents is returned if that point is in the window. Otherwise, the closest point to the mouse cursor is returned. If *transform* is non-nil, this ruler is part of a subpicture, and *transform* holds the subpicture translations.

:copy Method of **gwin:ruler**

Returns a copy of the ruler. This overrides the **:copy** method from **basic-graphics-mixin**.

The ruler flavor includes a separate **:transform** method because a ruler is actually a subpicture (a collection of several objects).

:transform *transform* Initialization Option of **gwin:ruler**
Gettable, settable. Default: An array that performs no transformations

Sets the initial transformation to be used. *transform* is a 3-by-3 matrix that contains scaling, translating, and rotational information for the translating points.

Raster Object Flavor

12.14 The graphics window system uses this flavor to create raster objects.

gwin:raster-object Flavor

The definition of the raster graphics entity. The raster pattern is determined by the pixels set in **gwin:bitarray**. In **gwin:bitarray**, the array of pixels for the object start at the coordinates (**gwin:xstart**, **gwin:ystart**) and have the dimensions **gwin:width** and **gwin:height**.

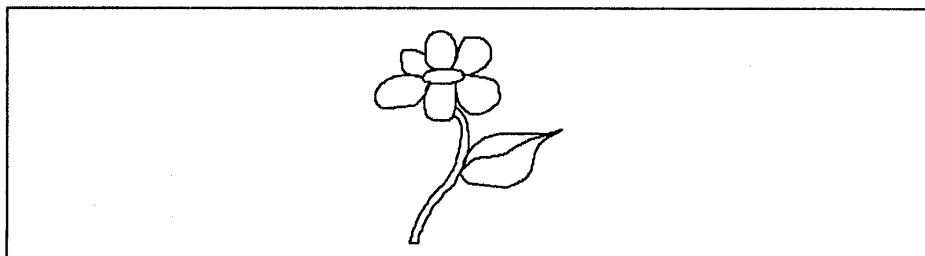
gwin:*current-cache-for-raster-objects*

Defparameter

When this variable is **t** (the default), raster objects contain a current cache, which is used to hold a scaled version of the raster image so that it can be **bitblt**ed to the screen. This prevents redrawing the image more than necessary. The current cache is a 1024 x 1024 x 1 bit array on a monochrome system and a 1024 x 1024 x 8 bit array on a color system. Eliminating the use of the current cache by setting this variable to **nil** saves on memory usage but causes the image to be redrawn more often when scaling.

NOTE: Raster objects always require large amounts of memory and disk space on color systems so their use should be limited.

Creating the Object **12.14.1** The following instance variables, initialization options, and methods define a raster object.



:bitarray <i>bitarray</i>	Initialization Option of gwin:raster-object
Default: nil	
Sets the raster pattern of the raster object.	
:height <i>height</i>	Initialization Option of gwin:raster-object
Default: 0.	
:width <i>width</i>	Initialization Option of gwin:raster-object
Default: 0.	
Sets the total height or width, respectively, of the raster object.	
:xscale <i>xscale</i>	Initialization Option of gwin:raster-object
Default: 1.	
:yscale <i>yscale</i>	Initialization Option of gwin:raster-object
Default: 1.	
:cur-height <i>cur-height</i>	Initialization Option of gwin:raster-object
Default: nil	
:cur-width <i>cur-width</i>	Initialization Option of gwin:raster-object
Default: nil	
Sets the scaling factor for the x or the y direction. This scaling factor, when applied to the total height or width, produces the current height or width.	

:xstart *xstart* Initialization Option of **gwin:raster-object**
 Default: 0.

:ystart *ystart* Initialization Option of **gwin:raster-object**
 Default: 0.

Sets the coordinates of the upper left corner of the raster object.

Manipulating the Object 12.14.2 The following standard methods are defined for raster objects. See the paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance *from-x from-y* Method of **gwin:raster-object**

:draw *window* Method of **gwin:raster-object**

:edit-parameters Method of **gwin:raster-object**

:fasd-form Method of **gwin:raster-object**

:move *dx dy* Method of **gwin:raster-object**

:scale *sx &optional (sy sx) &rest ignore* Method of **gwin:raster-object**

:undraw *on-window* Method of **gwin:raster-object**

:edge-point *the-window &optional transform* Method of **gwin:raster-object**

Returns the x and y coordinates of a point that is used for highlighting. The lower right corner of the raster rectangle is returned unless that point is outside the window; if that point is outside the window, the point on the rectangle closest to the mouse is returned. If *transform* is non-nil, this raster object is part of a subpicture, and *transform* holds the subpicture translations.

:copy Method of **gwin:raster-object**

Returns a copy of the raster object. This overrides the **:copy** method from **basic-graphics-mixin**. The bitblt arrays are copied separately.

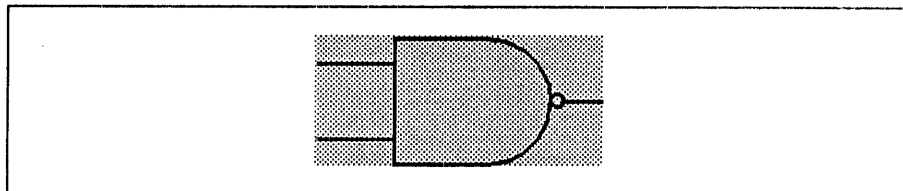
Picture Object 12.15 Pictures are a way of grouping graphic entities. Subpictures are used to group several entities into a single, larger unit. Background pictures are used for reference.

Subpicture Flavor 12.15.1 A *subpicture* is a collection of other graphics entities that are drawn relative to the subpicture origin. Subpictures are used for constructing an object that is a composite of several objects. For example, in a circuit design application, a subpicture could be used for creating each type of gate once.

gwin:subpicture Flavor

The subpicture graphics entity. If the subpicture has a fill color, it is drawn on a rectangle of the specified fill color.

Creating the Object 12.15.1.1 The following initialization options and methods define a subpicture.



- :edge-color** *edge-color* Initialization Option of **gwin:subpicture**
 Default: **nil**
 Sets the color of the border that surrounds the subpicture.
- :entities** *entities* Initialization Option of **gwin:subpicture**
Gettable. Default: An instance of the **gwin:rectangle** flavor
- :set-entities** *new-entities &optional (update-self t)* Method of **gwin:subpicture**
 Sets a list of the objects that are included in the subpicture. If *update-self* is non-**nil**, **:set-entities** updates the subpicture instance.
- :margin** *margin* Initialization Option of **gwin:subpicture**
Gettable, settable. Default: 3.
 Sets the margin width for the subpicture.
- :name** *name* Initialization Option of **gwin:subpicture**
Gettable, settable. Default: "", the empty string
 Sets a string that is the name for the subpicture.
- :weight** *weight* Initialization Option of **gwin:subpicture**
Gettable, settable. Default: 2.
 Sets the line thickness of the border around the subpicture.
- :x-origin** *x-origin* Initialization Option of **gwin:subpicture**
:y-origin *y-origin* Initialization Option of **gwin:subpicture**
 Sets the coordinates of the upper left corner of the raster object.
- :x-scale** *x-scale* Initialization Option of **gwin:subpicture**
:y-scale *y-scale* Initialization Option of **gwin:subpicture**
 Sets the scaling factor for the x or the y direction.
- :transform** *transform* Initialization Option of **gwin:subpicture**
Gettable, settable. Default: An array that performs no transformations
 Sets the initial transformation to be used. *transform* is a 3-by-3 matrix that contains scaling, translating, and rotational information for the translating points.
- Manipulating the Object* **12.15.1.2** The following standard methods are defined for subpictures. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.
- :distance** *from-x from-y* Method of **gwin:subpicture**
:draw *window* Method of **gwin:subpicture**
:edit-parameters Method of **gwin:subpicture**
:fasd-form Method of **gwin:subpicture**
:move *dx dy* Method of **gwin:subpicture**
:scale *sx &optional (sy sx) (scale-thickness? t)* Method of **gwin:subpicture**
:undraw *window* Method of **gwin:subpicture**
- :edge-point** *window &optional transform* Method of **gwin:subpicture**
 Returns the x and y coordinates for a point that is used for highlighting. The first edge point in the window is returned. If *transform* is non-**nil**, this subpicture is part of another subpicture, and *transform* holds the subpicture translations.

:copy Method of **gwin:subpicture**
 Returns a copy of the subpicture. This overrides the **:copy** method from **gwin:basic-graphics-mixin**.

Background Picture Flavor **12.15.2** The *background picture* entity is similar to the subpicture entity except that objects in background pictures cannot be selected or edited. Background pictures are typically used for reference or as template figures. You can create background pictures by using the **:create-and-add-entity-to-front** method of **gwin:world**.

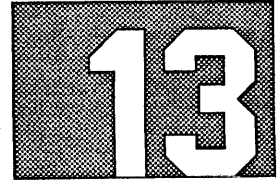
gwin:backgroundpic Flavor

The background picture graphics entity.

Background pictures are drawn according to the transformation for the current window. If this transformation is different from the one originally used to create the entities, they may appear to be a different size or to be shifted along some axis. The entities are actually the same, but the window used to view them has changed position.

The following methods are defined for background pictures. Each of these methods overrides the standard method of the same name. See paragraph 12.9, Standard Operations on Graphic Objects, for a detailed description of their operation.

:distance <i>&rest ignore</i>	Method of gwin:backgroundpic
:edit-parameters <i>&rest ignore</i>	Method of gwin:backgroundpic
:fasd-form	Method of gwin:backgroundpic
:highlight <i>ignore</i>	Method of gwin:backgroundpic
:unhighlight <i>ignore</i>	Method of gwin:backgroundpic



TYPEOUT WINDOWS

Using Typeout Windows

13.1 If a program normally displays a single updated picture, *typeout windows* provide an easy way to print a stream of unrelated output from time to time.

For example, Zmacs windows normally present a continuously updated display of an editor buffer. But some editor commands are designed to print output, such as a list of help commands (shown in the following figure). This output cannot be printed conveniently on the editor window itself, because that window is set up to maintain its standard display of an editor buffer and is not suitable for displaying anything else. Instead, a special kind of window, called a typeout window, receives the output. The typeout window exists as an inferior of the editor window. Other programs that maintain updating displays, such as the Inspector and Peek, also use typeout windows for this purpose.

```

Help:
Type one of the following characters to choose an option:
C Print out documentation for the command on a given key.
D Describes a command, specified by name.
O Print documentation of editor user option variable.
R List commands whose names contain a given string.
U List all editor options whose names contain a given substring.
U Undo the last undoable command done in the current buffer.
L List the last sixty keystrokes typed inside the editor.
S List the special characters available in this buffer.
W List all characters that invoke a given command.

You may continue choosing options until you exit.

Press the space bar to exit this command.
Any other non-option key will exit and be executed.

[
(check-type font font "a font object")
(LET* ((FONT-SIZE (IF (and (font-fill-pointer font) (NOT (ZEROP (FONT-FIL
(FONT-FILL-POINTER FONT)
;;ELSE
(LENGTH (FONT-CHAR-WIDTH-TABLE FONT))))

```

A typeout window is an inferior of another window, such as the editor or Peek display window, and grows over its superior as output is sent to it. The output starts at the top of the typeout window, which is also the top of its superior, and proceeds downward. The typeout window always keeps track of how far down output has proceeded so that the superior window can eventually find out how much of its permanent display has been overwritten by the typeout window and therefore needs to be redisplayed. A horizontal line or window shade appears just below the point of lowest output, enabling the user to separate the typeout from the remains of the permanent display. If output to the typeout window proceeds far enough, it wraps around to the top of the screen. The typeout window then records that the entire superior has been overwritten and no longer displays any horizontal line. You can suppress the horizontal line if desired.

w:basic-typeout-window

Flavor

Required flavor: w:essential-mouse

The base flavor for all kinds of typeout windows. This flavor is actually a mixin and is not instantiable by itself.

w:basic-typeout-window provides for methods and wrappers that cause the **:mouse-moves** and **:mouse-buttons** messages to be passed either to the typeout window or to its superior, depending on whether the typeout window has grown down to where the mouse is.

w:typeout-window Flavor

The flavor normally used for actual typeout windows.

w:typeout-window-with-mouse-sensitive-items Flavor

Provides the **:item** method for including mouse-sensitive rectangles among the typeout.

:bottom-reached Method of **w:basic-typeout-window**

Returns the greatest y position overwritten by the typeout window. The y position is a cursor position relative to the typeout window. The horizontal line (typeout window border), when enabled, appears at this position, provided the y position is not 0 or equal to the inside bottom of the window.

The value of this method is **nil** when the typeout window is not active.

w:*enable-typeout-window-borders* Variable

Default: **t**

Whether the bottom of the area used by the typeout window is marked by a horizontal line. When the variable is non-**nil**, a line appears if the typeout window has not used its entire area (if it has not wrapped around or executed a **:clear-screen**). When the variable is **nil**, the horizontal line does not appear.

Activation and Deactivation

13.2 A typeout window is deactivated when not in use. Any attempt to send output to the typeout window automatically activates and exposes the typeout window because this window uses the **:expose-for-typeout** method as its deexposed typeout action.

Exposing the typeout window automatically causes it to become the selection substitute of one of its ancestors. Which ancestor is determined by the situation; it is the nearest ancestor in the existing path of selection substitutes. This is the nearest ancestor that can be used for the purpose and actually make the typeout window be selected. The nearest ancestor is the typeout window's direct superior only if that superior is selected. For example, if you press **META-X** in Zmacs and then press **HELP**, the help message prints on the main editor window's typeout window, but that editor window is not selected (the minibuffer is). The typeout window substitutes for the editor frame rather than for the nonselected editor window immediately above it.

When the program wants to clear the typeout and put back its standard display, the program must first deactivate the typeout window using the **:deactivate** method.

When the typeout window is deactivated, it sends a **:remove-selection-substitute** message to whichever ancestor it decided to substitute for. As a result, if the typeout window is still that ancestor's selection substitute, the substitute is set back to what it was before the typeout window was exposed. If the ancestor's substitute has changed since then, the ancestor is left alone.

The primary purpose of making the typeout window a selection substitute is to make its cursor blinker blink. A typeout window by default shares the input buffer of its superior, so the window selected does not affect the reading of the keyboard input. A separate feature of typeout windows turns the superior's blinkers off completely while the typeout is exposed.

:expose-for-typeout Method of **w:basic-typeout-window**

Prepares the typeout window for typeout. The typeout window marks itself exposed while leaving the bits of its superior on the screen. The typeout window initializes itself as empty and its **:bottom-reached** as 0. The typeout window also finds a suitable ancestor and makes itself that ancestor's selection substitute. In normal use, this procedure selects the typeout window.

:active-p Method of **w:basic-typeout-window**

Returns non-*nil* if the typeout window is active, which is the case if and only if typeout is currently visible in the typeout window.

Windows With Inferior Typeout Windows

13.3 To make a window possess an inferior typeout window, include the **w:essential-window-with-typeout-mixin** flavor in it. The typeout window is not in the window superior's list of inferiors.

w:essential-window-with-typeout-mixin Flavor
Required flavor: **w:essential-mouse**

The basic mixin that gives a window the ability to manage a typeout window as its inferior. This flavor creates a typeout window and provides the methods to handle communication with the typeout window.

w>window-with-typeout-mixin Flavor

Prevents screen management of this window's inferiors from interfering with the operation of typeout windows.

:typeout-window Method of **w:essential-window-with-typeout-mixin**

Returns a pointer to the window's typeout window.

:typeout-window (*flavor-name options...*) Initialization Option of **w:essential-window-with-typeout-mixin**

Specifies what kind of typeout window to create. The car of the value, *flavor-name*, is the name of the flavor of typeout window to use; and the cdr, *options*, is a list of alternating options and values to pass to **make-instance**. If *options* is not specified or if it is *nil*, no typeout window is actually created.

:turn-on-blinkers-for-typeout Method of **w:essential-window-with-typeout-mixin**
:turn-off-blinkers-for-typeout Method of **w:essential-window-with-typeout-mixin**

Executes when the mouse moves into an area not used by the typeout window. These methods turn on or turn off, respectively, any blinkers associated with using the mouse. The definitions actually provided by the **w:essential-window-with-typeout-mixin** flavor do nothing; these methods exist so that you can add methods to them.

A typeout window does **MORE** processing if and only if **MORE** processing is enabled for its superior. The usual motivation for using a typeout window is that the superior is to be used for something other than sequential output; therefore, **MORE** processing is usually not desired on the superior. It is not desirable, however, to simply disable **MORE** processing for the superior because doing so disables it for the typeout window as well and because the user can reenable **MORE** processing for both windows with the TERM M keystroke sequence.

:more-p Method of **w:basic-typeout-window**

Returns **t** if **MORE** processing is enabled, or **nil** if the processing is disabled. The **:more-p** method is passed along to the superior so that the user who types the TERM M keystroke sequence need not be aware of the distinction between the typeout window and its superior.

:set-more-p *new-more-p* Method of **w:basic-typeout-window**

Changes **MORE** processing to the processing specified by *new-more-p*: **t** to enable **MORE** processing or **nil** to disable **MORE** processing.

w:intrinsic-no-more-mixin Flavor

Prevents **MORE** processing unconditionally without saying that it is disabled. Programs and the user think they enable and disable **MORE** processing for the window using the **:more-p** and **:set-more-p** methods and the TERM M command, but only the typeout window is affected. This flavor is intended for use in superiors of typeout windows.

The following code shows another way to disable **MORE** processing:

```
(defmethod (my-display-window-with-typeout-window :more-exception) ()
  (setf (w:sheet-more-flag) 0))
```

Delaying Redisplay After Typeout

13.4 Before redisplaying, the typeout window's superior must be able to determine whether part of its last display has been overwritten by the typeout window and therefore must be redisplayed. The **:bottom-reached** method returns the amount of the superior that the typeout window has overwritten. The typeout window must also be deactivated so that more typeout, occurring after the redisplay, works properly.

The following fragment of code shows how general scroll windows delay redisplay. Similar code can be used for any window that must redisplay itself after being covered by a typeout window, such as a Zmacs buffer.

```
;; Assumes w:typeout-window is an instance of a typeout window.
(defmethod (w:scroll-window-with-typeout-mixin :before :redisplay) (&rest ignore)
  (when (funcall w:typeout-window :active-p)
    (let ((btm-reached (min w:screen-lines
                          (1+ (truncate (send w:typeout-window :bottom-reached)
                                         w:line-height))))
          ;; btm-reached is the number of lines of the
          ;; display that were clobbered by typeout.
          (funcall w:typeout-window :deactivate)
          (dotimes (lines-clobbered btm-reached)
            ;; Mark lines as clobbered.
            (setf (aref w:screen-image lines-clobbered 0) nil)
            (setf (aref w:screen-image lines-clobbered 1) -1)
            (setf (aref w:screen-image lines-clobbered 2) -1))
          ;; Erase the clobbered area.
          (send self :draw-rectangle (w:sheet-inside-width) (* btm-reached w:line-height)
                  0 0 w:alu-andca))))
```


The editor normally updates its display after each command. After a command that prints typeout, however, it is important not to update the permanent display immediately; doing so makes the typeout disappear almost as soon as it appears. The same consideration applies to other programs that use typeout windows.

After a command has produced typeout, redisplay should be delayed until the user types another input character. If that character is a space, it is discarded. Otherwise, the character is interpreted as a command.

The program decides whether to wait before redisplaying by invoking the `:incomplete-p` method on the typeout window. This method reads a flag that is set whenever output is sent to the typeout window and that can be cleared by the program's command loop between commands. The flag, then, indicates whether the typeout window was used during the last command. The following code illustrates this technique:

```
(let ((*query-io* typeout-window))
  (do-forever
    ;; Clear the flag.
    (send *query-io* :make-complete)
    ;; Read and execute one command.
    (process-command (read-char *query-io*))
    (when (send *query-io* :incomplete-p)
      ;; If this command printed some typeout,
      ;; delay redisplay by waiting for next input char.
      (let ((ch (read-char *query-io*)))
        (unless (eq ch #\space)
          ;; Anything but Space, execute as a command.
          ;; Because Space is not unread, it allows immediate redisplay.
          (unread-char *query-io* ch))))
      ;; Redisplay after each command.
      (unless (send *query-io* :listen)
        ;; Normal redisplay must deactivate the typeout window;
        ;; see the previous example.
        (redisplay-normal-display))))
```

Note that this command loop follows the editor's practice of not redisplaying when there is input available. As a result, when the character read is not a `#\space`, input causes `unread-char` to prevent redisplay. Then the same character is read again at the top of the loop and processed as a command. If this command also prints typeout, its typeout adds to that already on the typeout window. If this command does not print typeout, the old typeout is erased after the command is executed.

:incomplete-p Method of `w:basic-typeout-window`

Returns the window's `incomplete-flag`. It is `t` if the command loop should wait for the next character before deactivating the typeout window.

:make-complete Method of `w:basic-typeout-window`

Sets the `w:incomplete-p` instance variable to `nil`. A command loop can use `:make-complete` to clear the flag after examining it.

Certain functions, such as `fquery`, perform this method on the I/O stream to tell the program not to wait before redisplaying, as it normally would do. The idea here is that the `fquery` question is not worth preserving on the screen once the user has answered it.

:make-incomplete Method of `w:basic-typeout-window`

Sets the `w:incomplete-p` instance variable to `t`. All the standard output stream methods also do this.

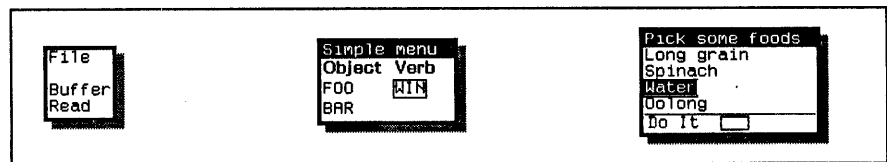
14

CHOICE FACILITIES

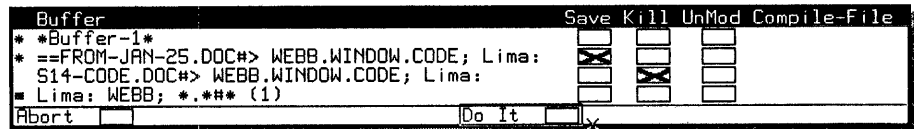
Introduction

14.1 The window system contains several facilities to allow a user to make choices. All these facilities work by displaying an arrangement of choices in a window. The user can select a choice by pointing to it with the mouse. The details (how the choices are specified, what the user interaction looks like, and what happens when a choice is selected) vary widely. The available choice facilities are the following:

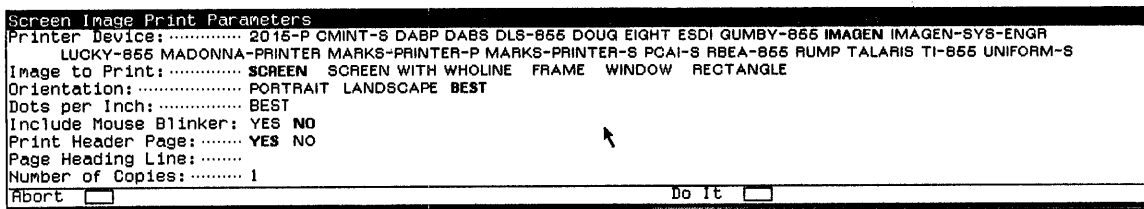
- Menus can be simple command menus, temporary or permanent, single column or multicolumn, scrolling or nonscrolling, highlighting or not highlighting, and so on. The items in the menu can be dynamically computed and put in different fonts, among other things.



- Multiple choice menus offer yes or no choices for several items. An example of a multiple choice menu is the Kill or Save Buffers menu from the Zmacs editor.



- Choose variable values menus enable a user to choose from several supplied values or to enter a value. An example of a choose variable values window is the Screen Image Print Parameters Window, invoked when you press TERM Q.



This section also discusses two other choice facilities: mouse-sensitive typeout and margin choices. This section does *not* discuss another facility that can be used as a choice facility: the universal command loop and suggestions. These are discussed in the *Explorer Tools and Utilities* manual.

Each choice facility is implemented by window flavors. For those who do not want to create their own window, each facility provides an easy-to-use function interface that creates a window of the appropriate flavor. The function interfaces are described first in each paragraph, followed by documentation on how to create and use a window that has the facility.

This section does not discuss how to modify these facilities to provide your own specialized versions, except in the simplest ways. If you want to customize your own facilities, read the code that implements the facility in question (for example, window instance variables and internal methods that you might want to add `:before` or `:after` methods to, or redefine).

Some portions of these facilities execute in the process that calls them, while other portions execute in the mouse process. All Lisp evaluation that concerns the user takes place in the user's process when the facilities described in this section are in use, with a very few exceptions that are noted when they occur. Thus, the user can freely use side effects (both special variables and the `throw` function) and need not worry that an error in the program will interfere with mouse tracking.

Menus Facility

14.2 A menu is an array of choices, each identified by a word or short phrase. You can select one of the choices by first moving the mouse near it, which causes it to be highlighted (a box appears around it), and then clicking any mouse button. The following code produces a simple menu that contains three mouse-sensitive items.



```
(w:menu-choose `(("FOO" :value foo :font fonts:tr12i
                 :documentation "Choose to FOO")
                ("BAR" :value bar
                 :documentation "Request a BAR")
                ("WIN" :value win
                 :documentation "Here's the brass ring")))
```

What happens when you select one of the choices depends on the particular type of menu. Typically, the choices in a menu are commands to a program or choices for the data that a command will operate on.

You can allow the system automatically to choose the arrangement of the choices and the size and shape of the window, or you can control these features explicitly.

To see an example of a menu, click the righthand mouse button twice to invoke the System menu.

Menu Items

14.2.1 A menu has a list of items, each representing one of the choices offered. An item tells the menu what to display and what to do if the user selects it (clicks on it). What to do specifies both what value to return and a possible side effect.

Response to selection of an item is implemented by the `:execute` method of the `w:menu` flavor, which is always sent in the user process rather than in the mouse process. Thus, side effects occur in the appropriate process. The returned value comes back to the user from `w:menu-choose`, `:choose`, or `:execute`, depending on how the menu is used.

An item can take one of the following forms:

- A string or symbol that is both what is displayed and what is returned. There are no side effects.
- A cons of the form *(name . atom)* where *name* (a symbol or a string) is what to display, and *atom* is what to return. There are no side effects.
- A list of the form *(name value)* where *name* is a string or a symbol to display, and *value* is any arbitrary object to return. There are no side effects.
- A list of the form *(name type arg option1 arg1 option2 arg2...)*. This is the most general form. *name* is a string or a symbol to display, *type* is a keyword symbol specifying what to do, and *arg* is an argument to this symbol. The *options* are keyword symbols specifying additional features desired, and the *args* following them are arguments to those options.
- An icon (a graphical object), which can appear in place of a string in any of the preceding forms.

Table 14-1 describes the keywords that can be the *type* value in the most general form of menu. If *nil* is supplied as a menu item, it is ignored completely. It takes up no space in the menu.

Each menu item can include one or more of the modifier keywords described in Table 14-2.

The first example of menu items is a list of three items that display as FOO, BAR, and WIN, and that return the symbols *foo*, *bar*, and *win* when chosen:



```
(w:menu-choose '(foo bar win))
```

The next example is another way of specifying the same thing, using more general syntax:



```
(w:menu-choose '(("FOO" :value foo)
                  ("BAR" :value bar)
                  ("WIN" :value win)))
```

A third way puts FOO in italics and adds strings for the mouse documentation window:

```
(w:menu-choose '(("FOO" :value foo :font fonts:tr12i
                  :documentation "Choose to FOO")
                  ("BAR" :value bar
                  :documentation "Request a BAR")
                  ("WIN" :value win
                  :documentation "Here's the brass ring.")))
```

With BAR selected, the documentation appears in the mouse documentation window.

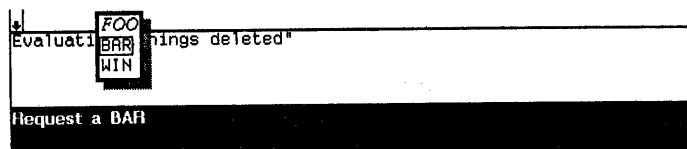


Table 14-1 Type Value Keywords for Menus


Keyword	Description of Argument
:buttons	A list of three menu items. The item actually chosen (that is, the item to be executed) is one of these three, depending on which mouse button was clicked. The order in the list is (<i>left middle right</i>). The three menu items in the list are used only for execution, not for display, so it does not matter what they have as the string to be displayed (it can be nil). You should put :font or :documentation keywords in the main menu item, the one that contains :buttons .
:eval	A form to be evaluated. Its value is returned.
:funcall	A function, with no arguments, to be called. Its value is returned.
:kbd	Usually either a character code, which is to be treated as if it were typed from the keyboard, or a list (a blip), which is a command to the program. In either case, the argument is sent to the selected window via the :force-kbd-input method. The use of :kbd produces an effect like using a command menu.
:menu	A menu instance to choose from; the menu is sent a :choose method, and the result is returned. Normally, the argument is a pop-up menu. If the argument is a symbol, it is evaluated.
:menu-choose	A list of the form (<i>label . menu-items</i>). The <i>label</i> and <i>menu-items</i> are passed as arguments to w:menu-choose , popping up another menu. The result of choosing from that menu is returned. <i>menu-items</i> is another list of menu items.
:no-select	Ignored but must be present to convert the item to the form that has a <i>type</i> keyword in it. This item cannot be selected. Moving the mouse near it does <i>not</i> cause it to be highlighted. This item is useful for putting comments, headings, and blank lines into menus.
:value	A form that is returned unevaluated. There are no side effects.
:window-op	A function with three arguments: the window the mouse was in before this menu popped up, and the x and y coordinates of the mouse at that time. This item type is handled by the :execute-window-op method of the w:menu flavor.

Table 14-2 Menu Item Modifier Keywords

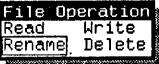

Keyword	Description
<code>:bindings</code>	A list of bindings to be made, suitable for passing to the function <code>progw</code> . These bindings are made before evaluating, before using <code>funcall</code> , before sending a message to a window, and so forth. If <code>:buttons</code> is used with <code>:bindings</code> , the <code>:bindings</code> must appear inside the menu item within the <code>:buttons</code> to have an effect on the final result.
<code>:documentation</code>	A string or list that briefly describes this menu item. When the mouse is pointing at this item, the documentation list or string is displayed in the mouse documentation window. The structure of the list is discussed in paragraph 11.6, How Windows Handle the Mouse, under the description of the <code>:who-line-documentation-string</code> method of <code>windows</code> .
<code>:font</code>	One of the following: a font object, a symbol whose value is the font, or a font purpose (such as <code>:menu-standout</code>). The item is displayed in that font instead of the menu's default font.
<code>:color</code>	A value or name of a color from the color map. An example of a name is <code>w:green</code> . The system-defined color names are listed in Table 19-2, Named Colors in the Default Color Map. Note that you must use a comma before a symbol name. The color is used when writing the item in the menu.

Other type value keywords and modifier keywords are used in the next example. The value returned is a symbol such as `read` or `write`, the value returned by the function `read`, or whatever the `buffer-op-menu` returns:

```
(setq buffer-op-menu (make-instance 'w:menu
  :pop-up t
  :label "The Buffer Op Menu"
  :item-list `(("this-buffer" :value here)
              ("that buffer" :value there))))
```



```
(w:menu-choose `(("File" :buttons
  ((nil :value read)
   (nil :value write)
   (nil :menu-choose
    ("File Operation" ;; Item list of menu obtained for click-right on File.
     ("Read" :value read :documentation "Read a file.")
     ("Write" :value write :documentation "Write a file.")
     ("Rename" :value rename :documentation "Rename a file.")
     ("Delete" :value delete :documentation "Delete a file."))))
  :documentation (:mouse-L-1 "Read file"
                 :mouse-M-1 "Write file"
                 :mouse-R-1 "Menu")))
  ;; The following makes a blank line.
  (" " :no-select nil)
  ("Buffer" :menu buffer-op-menu
   :documentation "Operate on this buffer")
  ("Read" :buttons
   ((nil :eval (read-char)) ; Reads one character
    (nil :eval (read-char) ; Reads one character
     :bindings ((base 10.))
     nil)
   :documentation
   (:mouse-L-1 "Read a single character"
    :mouse-M-1 "Read a single number, base 10")
  )))
```

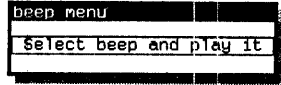



The following example shows the use of `:bindings`. The argument to the `:bindings` keyword temporarily binds the `beep-type` special variable to a two-column menu that contains all the beeping functions in a pop-up menu. When the user executes the outer `w:menu-choose`, a menu with a single item, `select beep and play it`, is displayed surrounded by white space. If the user clicks on this item, it invokes a second menu using the `:funcall beep-play` argument. This second menu uses the menu item list produced by `:bindings`.

```
(defun beep-play ()
  (beep beep-type))

(defvar beep-type nil)

(w:menu-choose `((" " :no-select)
  (" Select beep and play it " :funcall beep-play
    :bindings ((beep-type ` (w:menu-choose w:*beeping-functions*
      :columns 2 )))
    :documentation "Select a beep type from a menu and play it")
  (" " :no-select))
  :label "beep menu")
```



The following example illustrates how to make different choices appear in different colors.

```
(defun items-in-color ()
  (w:menu-choose `(("FOO" :value foo
    :font fonts:tr12i
    :color ,w:red ; note use of comma before
    ; symbol name
    :documentation "Choose to FOO")
  ("BAR" :value bar
    :color ,w:dark-green
    :documentation "Request to BAR")
  ("WIN" :value win
    :color ,w:blue
    :documentation "The winning choice"))))
```

Column Specification List

14.2.2 A column specification list, similar to a menu item list, specifies menu items. In addition to the data included in a menu item list, the column specification list details which column of a multicolumn menu the item appears in.

Each element of a column specification list specifies one column of the menu, and looks like this:

(heading item-list-form . options...)

where:

heading is a string to be displayed as a `:no-select` item at the top of the column.

item-list-form is a form to be evaluated to produce the list of items for the column. It should have no side effects and can be evaluated in any process.

options are modifier keywords and values. Possible values are `:font` and `:sort`.

- **:font** specifies the font for the column heading. It accepts as a value one of the following: a font object, a symbol whose value is the font, or a font purpose (such as **:menu-standout**).
- **:sort** applies to this column only. It accepts any value accepted by the **:sort** initialization option of the **w:menu** flavor, as described in paragraph 14.2.8, Menu Format.

Column specification lists are used when the **:multicolumn** keyword of **w:menu-choose** is **t** and with **w:multicolumn-menu-choose**.

Icons 14.2.3 An *icon* can be used in a menu item list or in a column specification list anywhere that a string can be used. An icon appears as a graphic image. Icons can also be used for other purposes. For example, the arrows that point up or down in the scroll bar are icons.

The icon object—the object manipulated by the Explorer system—is a structure similar to a font object; in fact, an icon object uses the font structure with only the name, icon width, icon height, and baseline defined. An icon object also has an associated function that is used to draw the icon. You create an icon object using the **w:make-simple-icon** function rather than explicitly using the **defstruct** function. The **w:draw-icon** function draws the icon on the window.

w:make-simple-icon *drawing-function icon-name width height &rest icon-args* Function
Creates an icon instance and initializes it according to the arguments passed in.

Arguments: *drawing-function* — The function that draws the icon. This function, which actually determines how the icon will appear, should have the following arguments:

icon-object window x y item &rest additional-args

where:

icon-object is an instance of the icon **defstruct**. This object is typically created by the **w:make-simple-icon** function that calls the drawing function.

window specifies the window on which to draw the icon.

x and *y* are the coordinates in *window* where the upper left corner of *icon-object* is to be drawn.

item is the menu item for this icon.

additional-args should be the same as *icon-args* for **w:make-simple-icon**.

icon-name — The user-defined name of the icon, used for sorting.

width, height — Width or height of the icon, respectively, in pixels. These values are used to specify the size of the area that is boxed when the user moves the mouse cursor over the icon.

icon-args — Any arguments to be passed to *drawing-function*.

`w:draw-icon` *icon-object* *window* *x* *y* &optional *item* *icon-color*

Function

Draws *icon-object* on *window* at coordinates *x,y*. *item* is the menu item for this icon; *icon-color* is the color of the icon, which is significant only on color systems. Note that the first five arguments of `w:draw-icon` are the same as the first five arguments of the *drawing-function* argument to `w:make-simple-icon`.

For example, suppose you create an icon on a window that is *not* a menu. First, you would define a drawing function for the icon that determines its shape, size, and position. Then, you would use the `w:draw-icon` function to actually draw the icon. Note that, in this case, you specify the window and coordinates where to draw the icon. To remove the icon, refresh the window by pressing the CLEAR SCREEN key.

```
(defun simple-box-icon-drawing-function (ignore window x y ignore)
  "Draws a simple box icon."
  (let ((inside-x (+ x -1 (w:sheet-inside-left window)))
        (inside-y (+ y (w:sheet-inside-top window))))
    (w:prepare-sheet (window)
      (sys:%draw-rectangle 50 25 inside-x inside-y w:alu-ior window))))

(w:draw-icon
 (w:make-simple-icon #'simple-box-icon-drawing-function 'box 50 25)
 w:selected-window 400 200)
```

```
;; -*- Mode:Common-Lisp; Package:W; Fonts:(CPTFONT HL12B HL12BI) -*-
(defun simple-box-icon-drawing-function (ignore window x y ignore)
  "Draws a simple box icon."
  (let ((inside-x (+ x -1 (w:sheet-inside-left window)))
        (inside-y (+ y (w:sheet-inside-top window))))
    (w:prepare-sheet (window)
      (sys:%draw-rectangle 50 25 inside-x inside-y w:alu-ior window))))

(w:draw-icon
 (w:make-simple-icon #'simple-box-icon-drawing-function 'box 50 25)
 w:selected-window 400 200)
```

However, if you invoke a menu of icons, the menu code itself calls `w:draw-icon` and supplies the window and coordinates. The following code creates a menu with a single box icon. This code uses the same drawing function as for the previous example. Note that, in this code, you do not specify the window or the coordinates; these are supplied automatically by the menu facility.

```
(w:menu-choose (list `((w:make-simple-icon #'simple-box-icon-drawing-function 'box 50 25)
                      :value :foo
                      :documentation "A box"))
              :item-alignment :center
              :label "A simple icon example")
```



You can create shapes other than boxes. The following code creates a menu that includes a box and a triangle. This code also uses the drawing function defined in the first example.

```
(defun simple-triangle-icon-drawing-function (ignore window x y ignore)
  "Draws a simple triangle icon."
  (let ((inside-x (+ x -1 (w:sheet-inside-left window)))
        (inside-y (+ y (w:sheet-inside-top window))))
    (w:prepare-sheet (window)
      (sys:%draw-shaded-triangle inside-x inside-y
                                 (+ 50 inside-x) (+ 50 inside-y)
                                 (+ 25 inside-x) (+ 50 inside-y)
                                 w:alu-ior nil nil t nil window))))
```

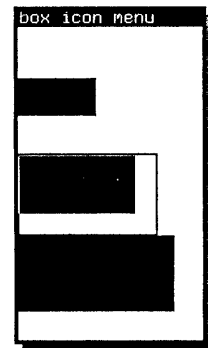
```
(w:menu-choose
  (list
    `,(w:make-simple-icon #'simple-box-icon-drawing-function 'box 50 25)
      :value :foo :documentation "A box")
    `,(w:make-simple-icon #'simple-triangle-icon-drawing-function 'triangle 50 50)
      :value :bar :documentation "A triangle")
  :label "icon menu")
```



You can also make more general drawing functions if you want to create a series of icons that are related geometrically. Note that this menu is a single column; drawing functions to enable multiple column menus with icons are more complex than those that assume the menu has only one column.

```
(defun general-box-icon-drawing-function (ignore window x y ignore width)
  "Draws a simple box icon."
  (let ((inside-x (+ x -1 (w:sheet-inside-left window)))
        (inside-y (+ y (w:sheet-inside-top window))))
    (w:prepare-sheet (window)
      (sys:%draw-rectangle (* 30 width) (* 15 width)
        inside-x inside-y w:alu-ior window))))
```

```
(w:menu-choose
  (append
    (loop for width from 2 to 4
      collect
        `,(w:make-simple-icon #'general-box-icon-drawing-function
          'box (* width 35) (* width 20) width)
          :documentation "Several graduated boxes"
        )))
  :columns 1
  :label "box icon menu")
```

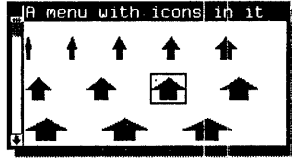


The following code presents a more complex example of using icons. This code creates a number of icons in a scrollable menu. The icons are similar to the arrows used to mark the scroll bar. The menu-with-icons function actually creates the menu with its icons; simple-icon-drawing-function invokes an internal function, w:scroll-bar-draw-icon, to draw each icon within the menu. This internal function handles multiple-column menus for this icon.

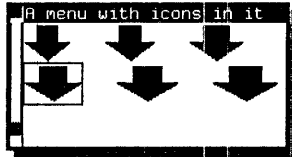
```
(defun menu-with-icons ()
  "Bring up a menu with some simple icons in it."
  (w:menu-choose
    (append
      ;; Upward pointing icons.
      (loop for width from 5 to 100 by 3
        collect `,(w:make-simple-icon #'simple-icon-drawing-function
          'arrow width 20. width 20. t)
          :value :foo :documentation
            ,(format nil "An arrow of width -D" width)))
      ;; Downward pointing icons.
      (loop for width from 5 to 50 by 3
        collect `,(w:make-simple-icon #'simple-icon-drawing-function
          'arrow width 30. width 30 nil)
          :value :foo :documentation
            ,(format nil "An arrow of width -D" width))))
    :geometry '(0 3 200.) ; Make the menu small so it must be scrolled to show all the items (icons).
    :label "A menu with icons in it"))
```

```
(defun simple-icon-drawing-function (ignore window x y ignore
  arrow-width arrow-height &optional (upward-pointing t))
  "Draws a simple arrow icon of specified dimensions and position which points in a
  specified direction."
  (let ((inside-x (+ x -1 (w:sheet-inside-left window)))
        (inside-y (+ y (w:sheet-inside-top window))))
    (w:prepare-sheet (window)
      (w:scroll-bar-draw-icon (w:sheet-screen-array window)
        inside-x inside-y
        arrow-width arrow-height upward-pointing))))
```

When you execute `menu-with-icons`, the initial display is similar to the following:



If you scroll to the bottom of the menu, the display is similar to the following:



The next example presents a menu of icons with each icon drawn in a different color using the default color map. Note that some choices seem to disappear because the contrast between the color of the icon and the background color is insufficient.

```
(defun circle-icon (ignore window x y ignore radius color)
  "Helper function to draw a circle as an icon"
  (let ((inside-x (+ x 15 (w:sheet-inside-left window)))
        (inside-y (+ y 12 (w:sheet-inside-top window)))
        )
    (tv:prepare-sheet (window)
      (send window
        :string-out-explicit (format nil "Color -d" color)
        (+ 15 inside-x) inside-y nil nil fonts:tr12i
        (w:sheet-char-aluf window) 0 nil nil color)
      (send window :draw-filled-circle inside-x inside-y radius color)
    ))
)

(defun display-color-icons ()
  (w:menu-choose
    (append
      (loop for color from 0 to 23 ; build a list of icons,
            ; each a different color
            collect
              `((w:make-simple-icon #'circle-icon 'circle
                90 26 12 color)
                :documentation ,(format nil "A circle of color -d" color)
              )))
      :columns 1
      :item-alignment :center
      :foreground-color w:white
      :background-color w:black
      :label `(:string " Color Selections: "
        :background ,w:dark-blue
        :color ,w:white)
    )
  )
)
```

If you want to define an icon with additional slots, then you define a new `defstruct` and use the `include defstruct` option to include the icon definition in the new `defstruct`.

You should also define a printer function for the new `defstruct` similar to the one defined for the icon `defstruct`. (A printer function defines how the icon name is printed.) The following is an example of a printer function:

```
(defun (:property new-icon named-structure-invoke)
  (op &optional slf arg1 &rest ignore)
  (case op
    (:which-operations `(:print-self))
    ((:print-self) (printing-random-object (slf arg1 :typep)
      ;; Print the new icon and its dimensions.
      (princ (icon-font-name slf) arg1)
      (princ "-" arg1)
      (princ (icon-font-char-width slf) arg1)
      (princ "-" arg1)
      (princ (icon-font-char-height slf) arg1)))
    (otherwise (ferror () "-S unknown message to -S" op slf))))
```

For example, suppose you create an icon instance with the following code:

```
(setq my-icon
  (w:make-simple-icon #'simple-box-icon-drawing-function 'box 50 25))
```

If you then evaluate `my-icon`, the system returns

```
#<w::icon box-50-25 23207100>
```

where:

`box` is the `icon-font-name`.

`50` is the `icon-font-char-width`.

`25` is the `icon-font-char-height`.

`23207100` is an arbitrary memory location that changes for each icon instance.

Functions That Create Menus

14.2.4 The following functions create menus. The most general function, `w:menu-choose`, is discussed first, followed by two special cases of this function (`w:multicolumn-menu-choose` and `w:multiple-menu-choose`). Finally, two special functions, `w:mouse-y-or-n-p` and `w:mouse-confirm`, are discussed. These functions allow the user to answer yes or no to a specific question by either clicking on the window or moving the mouse outside of the window.

The Most General Function

14.2.4.1 The `w:menu-choose` command can be used to create all general-purpose menus.

```
w:menu-choose item-list &key (:abort-on-deexpose nil) (:columns nil)      Function
                (:command-menu nil) (:default-item nil) (:dynamic nil)
                (:geometry nil) (:highlighted-items nil) (:highlighting nil)
                (:io-buffer nil) (:item-alignment :left) (:label nil)
                (:menu-margin-choices nil) (:multicolumn nil) (:near-mode '(mouse))
                (:permanent nil) (:pop-up t) (:scrolling-p t) (:sort nil)
                (:superior w:mouse-sheet)
                (:foreground-color w:*default-menu-foreground*)
                (:background-color w:*default-menu-background*)
                (:label-color w:*default-menu-label-foreground*)
                (:label-background w:*default-menu-label-background*)
```

Creates a menu by using the `w:menu` flavor. You can specify various keywords to determine the most frequently used features of menus. `w:menu-choose` returns either `nil` (if the user moved the mouse out of the

menu or pressed the ABORT key) or two values (if the user chose an item): the value computed from the chosen item, and the item itself (an element of *item-list*).

If you do not specify any keywords, the function creates a pop-up menu and allows the user to make a choice with the mouse or the keyboard. When the choice is made, the menu disappears and the chosen item is executed. Note that if the user moves the mouse blinker just outside the menu, the menu does not disappear. This facility prevents inadvertent deexposure of a menu.

By default, the menu does not include a label, but it does include a scroll bar and scroll icons if all the choices do not fit within the menu (That is, the menu's scroll bar mode is **:maximum**. See paragraph 11.8.1, Scroll Bars, for more information about scroll bars and scroll bar modes.) If you specify a label but do not specify its position, the label appears at the top of the menu in a box.

w:menu-choose performs some validity checking to catch some keyword combinations that do not make sense. For example, a highlighting menu either must have margin choices, or be either a command menu or a permanent menu. If none of these cases apply, an error is signaled. **w:menu-choose** also verifies that the number of columns specified by **:columns** is the same as the number of columns listed in *item-list*.

Arguments: *item-list* — A list of items as described at the beginning of paragraph 14.2.1, Menu Items. For multicolumn menus this is the column specification list instead as described in paragraph 14.2.2.

:abort-on-deexpose — Executes an abort item when the menu becomes deexposed without selecting an item.

- If **:abort-on-deexpose** is **nil** (the default), this search is not made. Instead, **nil** is returned.
- If **:abort-on-deexpose** is **t**, an attempt is made to find an abort item to execute when the menu becomes deexposed.
 1. The attempt to find an abort item starts by looking for a margin choice that has "Abort" as its value (case is not significant).
 2. If no abort margin choice is found, then an abort menu item is searched for.
 3. If no abort menu item is found, then no menu item is executed.

:columns — The number of columns in the menu. The default is **nil**, which means that the number of columns is determined from the geometry. If a value is specified for **:columns**, the value must be consistent with the number of columns used in the column specification list in *item-list*.

:command-menu — When specified as **t**, **:command-menu** causes menu selections to be placed into an I/O buffer as a blip. The default is **nil**, which means that this is not automatically done. If you specify **:command-menu** as **t**, you must also supply an I/O buffer. This keyword is explained in more detail in paragraph 14.2.6.1.

:default-item — The item over which the mouse is initially positioned. The value is either **nil** or is an item that is **eq** to an element of *item-list*. The default is **nil**, which means that the mouse is initially positioned in the center of the menu. This argument allows the user to select that item without moving the mouse.

Recall that the second value returned by `w:menu-choose` is the item that was selected. If you save this value and use it as the `:default-item` in a subsequent call to `w:menu-choose`, the mouse is automatically positioned over the user's previous selection.

- :dynamic** — Determines whether *item-list* can be dynamically updated. A value of `t` allows dynamic updating. The default is `nil`. If you specify **:dynamic** as `t`, *item-list* is a symbol whose value can change. After the change, the **:update-item-list** method must be sent to the menu to cause it to reflect the new item list.
- :geometry** — A geometry list that specifies the layout and dimensions of the menu. The default is `nil`, which means that the layout is determined either from the number of items present or from the dimensions of the superior if there are too many items. See the description of the **:set-geometry** method of `w:menu` (paragraph 14.2.7.2, Elements of a Geometry List) for details about geometry lists.
- :highlighted-items** — A list of menu items that are initially highlighted. The elements of this list must be either `eq` to items in *item-list*, or `nil`. The default is `nil`, which means that no items are initially highlighted.
- :highlighting** — Determines whether to allow the user to select more than one choice before clicking on the margin choices. A value of `t` allows more than one choice; a value of `nil` (the default) does not. Menus that allow more than one choice are usually called multiple-choose menus. If you specify **:highlighting** as `t`, you should also specify margin choices. If you do not, the user can exit the menu only by moving the mouse outside the menu and clicking there, which always returns a value of `nil`.
- :io-buffer** — The I/O buffer for command menus. This must be specified if you specify **:command-menu** as `t`. The default is `nil`.
- :item-alignment** — The alignment of items within a column. Possible values are **:left** (the default), **:center**, or **:right**. See paragraph 14.2.8, Menu Format, for examples of the different alignments.
- :label** — Either the label of the menu, or `nil` if the menu is not to have a label. The default is `nil`. The exact appearance of the label depends on whether the menu is a permanent or a pop-up menu. Labels on pop-up menus appear in reverse video; labels on permanent menus are in standard video.
- :menu-margin-choices** — A list of margin choices that allow a user to complete the selection of menu items and return from `w:menu-choose`. The default is `'(nil)`. Possible elements of the list are `nil` and the keywords **:abort**, **:doit**, and **:abort-and-doit**. You should include margin choices if you specify **:highlighting** as `t`. In addition, a user can specify a special-purpose margin choice. For menus, a margin choice has the same format as a menu item. A commonly used case is one in which the `:eval` type value keyword is used. See Table 14-1 for information about `:eval`.
- :multicolumn** — Whether the menu is a multicolumn menu. When the value is `t`, *item-list* is interpreted as a column specification list to generate a multicolumn menu. The default is `nil`. See paragraph 14.2.2, Column Specification Lists, for the required format of *item-list*.
- :near-mode** — Where to put the menu. It must be an acceptable argument to **:expose-near** method. The default is `'(:mouse)`, the current position of the mouse cursor.
- :permanent** — Whether the menu is a permanent menu (specified by `t`) or a pop-up menu (specified by `nil`, the default). If both **:permanent** and

:pop-up are specified as **t**, an error is signaled. This keyword is described in more detail in paragraph 14.2.6.4, Permanent Versus Pop-Up Menus.

:pop-up — Whether the menu becomes deexposed when the mouse moves out of it. A value of **t**, the default, causes the menu to be deexposed; a value of **nil** causes it not to be deexposed. If both **:permanent** and **:pop-up** are specified as **t**, an error is signaled.

:scrolling-p — Whether the menu has scrolling. A value of **t**, the default, enables scrolling; a value of **nil** disables scrolling. If scrolling is disabled, the user cannot move to the items that are not currently displayed by using the mouse; the user can reach these items by using keystroke sequences such as CTRL-V, META-V, and so on. Even if scrolling is enabled, the scroll bar only appears when the menu cannot display all the items.

:sort — The manner in which the menu items should be ordered. Possible values are the following:

:ascending , or t for short	(:ascending key)
:descending	(:descending key)
<i>predicate</i>	(<i>predicate key</i>)
nil	

where *predicate* and *key* are as required by the **sort** function. The argument passed to *key* is a string displayed for a menu item; in the case of an icon item, it is a string containing the name of the icon. Similarly, the arguments passed to *predicate* are either menu item strings or values returned by *key*.

The default for **:sort**, **nil**, does not change the ordering of the menu items. Note that using **:sort** destructively modifies the item list.

:superior — The sheet of which the menu should be an inferior. The default is **w:mouse-sheet**.

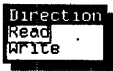
:foreground-color — The foreground (text) color of the menu.

:background-color — The background color of the menu.

:label-color — The label foreground (text) color for the menu.

:label-background — The label background color for the menu.

The following example returns **foo** or **bar** if a user clicks on an item, or **nil** if the user moves the mouse out of the menu.



```
(w:menu-choose `(("Read" :value foo)
                 ("Write" :value bar))
               :label "Direction")
```

Special Functions for Compatibility

14.2.4.2 The **w:multicolumn-menu-choose** and **w:multiple-menu-choose** functions are special cases of the **w:menu-choose** function that are included for compatibility. If you are writing new code, you should use **w:menu-choose**.

w:multicolumn-menu-choose *column-spec-list* Function
 &optional (:label nil) (:near-mode '(:mouse))
 (:default-item nil) (:superior w:mouse-sheet)

Passes the arguments to the **w:menu-choose** function to create a multicolumn menu. **w:menu-choose** allows the user to choose one of the items using the mouse. When the choice is made, the menu disappears and the chosen item

is executed. The value of that item is returned as the first value of `w:multicolumn-menu-choose`, and the item itself (an element of *column-spec-list*) is returned as the second value.

If the user moves the mouse blinker well outside of the menu, the menu disappears and `w:multicolumn-menu-choose` returns `nil`.

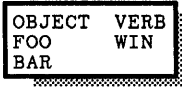
column-spec-list is a list of column specifications as described in paragraph 14.2.2, Column Specification List. The keywords are the same as those used for `w:menu-choose`.

For example, consider the previous example:



```
(w:menu-choose `(("FOO" :value foo)
                 ("BAR" :value bar)
                 ("WIN" :value win)))
```

Suppose you wanted to put `FOO` and `BAR` in a column labeled `OBJECT`, with `WIN` in a second column labeled `VERB`. The following code does this.



```
(setq object-list `(("FOO" :value foo)
                   ("BAR" :value bar))
      verb-list    `(("WIN" :value win))
(w:menu-choose   `(("OBJECT" object-list)
                   ("VERB" verb-list)
                   :multicolumn t)
```

However, the following code returns the same results but has a more effective display:

- The column headings (which are not selectable) are in a different font.
- The menu as a whole has a title.



```
(w:multicolumn-menu-choose `(("Object" object-list :font fonts:hl12b)
                              ("Verb" verb-list :font fonts:hl12b))
                           :label "Simple menu")
```

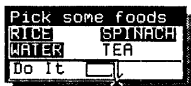
`w:multiple-menu-choose` *item-list* &optional (:label nil) Function
 (:near-mode '(:mouse)) (:highlighted-items nil)
 (:menu-margin-choices '(:doit)) (:superior w:mouse-sheet)

Passes the arguments to the `w:menu-choose` function to create a multiple choose menu. `w:menu-choose` allows the user to choose any subset of the available items. The user finalizes the choice by clicking on the Do It box at the bottom of the menu or pressing the END key. At this time, `w:multiple-menu-choose` returns as its first value a list of the results of executing all the chosen menu items. The second value of `w:multiple-menu-choose` is non-`nil` in this case.

If the user moves the mouse well outside of the menu, the menu disappears and the `w:multiple-menu-choose` function returns `nil` for both values. The second value enables the caller to distinguish between a refusal to choose and choosing the empty set of items.

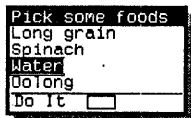
item-list is a list of items as described at the beginning of paragraph 14.2.1, Menu Items. The keywords are the same as the keywords used for `w:menu-choose`.

The following code returns the list (rice spinach water) if the user clicks on the entries for rice and spinach and does not turn off water:



```
(w:multiple-menu-choose '(rice spinach water tea)
  :label "Pick some foods"
  :highlighted-items '(water))
```

The following code returns the same possible values but enables you to display a string that is not necessarily the same as the value returned.



```
(defvar items `(("Long grain" :value rice)
  ("Spinach" :value spinach)
  ("Water" :value water)
  ("Oolong" :value tea))
(w:multiple-menu-choose items
  :label "Pick some foods"
  :highlighted-items
  (list (assoc-if #'(lambda (foo) (equal foo "Water"))
    items)))
```

Other Special Functions

14.2.4.3 The `w:mouse-y-or-n-p` and `w:mouse-confirm` functions are special menus that allow the user to answer yes or no to a specific question by either clicking on the window or moving the mouse outside of the window.

`w:mouse-y-or-n-p` *string*

Function

Asks the user to answer yes or no. The user can answer yes by clicking on the Yes item or by pressing the END key. The user can answer no by clicking on the No item or by moving the mouse out of the window. The window is a menu that displays a single item: *string*. If the user clicks on the Yes item, the function returns `t`; otherwise, the function returns `nil`.

For example, suppose you execute the following function:

```
(w:mouse-y-or-n-p "Do you want to continue?")
```

The system pops up a window similar to the following near the mouse:



with the following documentation in the mouse documentation window:

```
Click mouse or press <END> to select YES. Pressing <ABORT> or moving off menu will also select NO.
```

`w:mouse-confirm` *message* & optional *what-to-do*

Function

```
(message-font fonts:hl12b) (what-to-do-font fonts:hl12)
(window-max-width 400.)
```

An improved version of the `w:mouse-y-or-n-p` function. `w:mouse-confirm` allows you to specify multiple fonts for the message or the action description. If the message for the *what-to-do* string is too long, it is formatted to fit the window using the `w:adjust-by-interval` function. Formatting involves moving words that do not fit on one line to the next line.

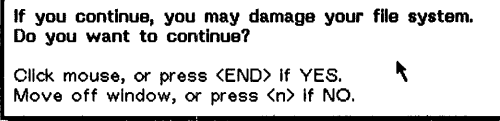
The user can answer yes by clicking on the Yes item or by pressing the END key or the space bar. The user can answer no by clicking on the No item or by moving the mouse out of the window. If the user answers yes, the value returned is `t`. If the user answers no, the value returned is `nil`.

- Arguments:**
- message* — The string to display at the top of the window.
 - what-to-do* — A string to display at the bottom of the window. The string tells the user what to do.
 - message-font* — The font used to display *message*.
 - what-to-do-font* — The font used to display *what-to-do*.
 - window-max-width* — The maximum number of pixels for the window width.

For example, suppose you execute the following function:

```
(w:mouse-confirm "If you continue, you may damage your file system. Do you want to continue?")
```

The system pops up a window similar to the following near the mouse:

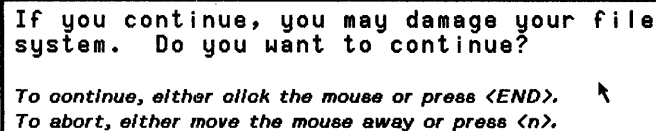


```
If you continue, you may damage your file system.
Do you want to continue?

Click mouse, or press <END> if YES.
Move off window, or press <n> if NO.
```

You can customize the instructions, as the following code does:

```
(w:mouse-confirm "If you continue, you may damage your file system. Do you want to continue?"
  "To continue, either click the mouse or press <END>."
  "To abort, either move the mouse away or press <n>."
  fonts:bigfnt fonts:h112bi 600)
```



```
If you continue, you may damage your file
system. Do you want to continue?

To continue, either click the mouse or press <END>.
To abort, either move the mouse away or press <n>.
```

However, you should be aware that the formatting function can only display within the *window-max-width* constraint. If you specify a small value for that argument, as in the following code, you could get a very tall, thin window, which is probably not what you want.

```
(w:mouse-confirm "If you continue, you may damage your file system. Do you want to continue?"
  "To continue, either click the mouse or press <END>."
  "To abort, either move the mouse away or press <n>."
  fonts:bigfnt fonts:h112bi 60)
```



```
If
you
cont i
nue,
you
may
danag
e
your
file
syste
n.
-
```

w:adjust-by-interval *maximum-number-of-characters string*

Function

Returns *string* formatted so that each line contains at most *maximum-number-of-characters*. If possible, the lines are broken at spaces. This function is used by **w:mouse-confirm**.

**Keyboard
Interface for Menus**

14.2.5 A user can move the mouse cursor within a menu by using keystroke sequences as well as by using the mouse. These keystroke sequences are similar to the ones used to move around in a choose variable values window.

w:menu-default-command-characters

Variable

An association list that defines the keystroke sequences used to move about the menu. Each element of the association list has the form:

(character (method . argument-list) documentation-string)

where *character* is either a character or a string. If it is a character and *method* is non-`nil`, then *method* is called with *argument-list*. If *character* is a string, it is displayed for documentation purposes.

All the methods called by the default values either return `nil` (which indicates that nothing else needs to be done) or return a value (which indicates that an item has been selected and that it needs to be processed). By default, the only keystroke that causes a non-`nil` value to be returned is `END`.

documentation-string is displayed when the `HELP` key is pressed. If no documentation string is to be displayed, this value should be `nil`.

The following keys are defined by default:

Keystroke Sequence	Description
<code>CLEAR SCREEN</code>	Refreshes the menu.
<code>RETURN</code>	Moves the mouse cursor to the next line.
<code>META-<</code>	Moves the cursor to the top item.
<code>META-></code>	Moves the cursor to the bottom item.
<code>CTRL-P</code>	Moves the mouse cursor up.
<code>CTRL-N</code>	Moves the mouse cursor down.
<code>CTRL-F</code>	Moves the mouse cursor right.
<code>CTRL-B</code>	Moves the mouse cursor left.
<code>CTRL-A</code>	Moves the mouse cursor to the leftmost column.
<code>CTRL-E</code>	Moves the mouse cursor to the rightmost column.
<code>CTRL-V</code>	Moves the mouse cursor to the next page.
<code>META-V</code>	Moves the mouse cursor to the previous page.
<code>↑</code>	Moves the mouse cursor up.
<code>↓</code>	Moves the mouse cursor down.
<code>→</code>	Moves the mouse cursor right.
<code>←</code>	Moves the mouse cursor left.
<code>CTRL-S</code>	Searches forward for the string using an incremental search. The user can press the <code>HELP</code> key to see a description of the keys that are defined within the search mode.
<code>CTRL-R</code>	Searches backward for the string using an incremental search. The user can press the <code>HELP</code> key to see a description of the keys that are defined within the search mode.
<code>SPACE</code>	In a multiple choose menu, selects an item
<code>ABORT</code>	Exits abnormally.
<code>CTRL-G</code>	Exits abnormally

Continued

(Continued)

Keystroke Sequence	Description
END	Exits normally with item(s) selected
HELP	Displays a help screen that includes these keystroke sequences.

If you want to supply additional keystrokes, you should supply the command-characters initialization option when you create the menu. The argument to this initialization option is a list similar to `w:menu-default-command-characters`.

Flavor and Initialization Options That Define Menus

14.2.6 If you prefer, you can use a flavor and its initialization options and methods to implement menus rather than using the functions. Using the `w:menu` flavor and its associated initialization options and methods gives you greater flexibility in customizing the action of a menu; the code is correspondingly more complex and difficult to use. Before attempting to design your own menus, you should scan the following paragraphs to see what initialization options and methods are available.

`w:menu`
`w:menu`

Flavor
Resource

Used directly by the `w:menu-choose` function. The methods and defaults for the flavor correspond closely to the keywords and defaults for the function. However, the `w:menu-choose` function creates a pop-up menu by default, while making an instance of the `w:menu` flavor defaults to creating a permanent menu.

Some initialization options are exclusive choices; others can be specified for any kind of menu, as follows:

- A menu can be either a command menu (a `:menu` blip is placed in the I/O buffer as soon as the user clicks a mouse button) or a highlighting menu (the user can select several menu items and must place the values in the I/O buffer by selecting a margin choice). Both command menus and highlighting menus must specify I/O buffers; a highlighting menu should also include margin choices.
- A menu can be either a permanent menu (the menu remains exposed even when the user has moved the mouse well outside the menu area) or a pop-up menu (the menu deexposes itself when the user moves the mouse well outside the menu area).
- Any of these kinds of menus can be dynamic or static, single column or multicolumn, or scrolling or nonscrolling.

Command Menus

14.2.6.1 The menus described so far are driven by the `:choose` method; that is, the program decides when it is time for the user to choose something in the menu. In some applications, the user should decide when to choose something from a menu. For example, in Peek, the user can select a new mode with the menu at any time, but Peek cannot spend all its time waiting for the user to do this. The command menu is designed for such applications. When an item in a command menu is chosen, the menu puts a blip into the application's I/O buffer.

Usually, a command menu pane is part of a team of windows managed by a single process and sharing a single input buffer. Menu clicks generate input that is typically read by a single stream together with mouse clicks on the other windows and keyboard input. For example, Peek and the Inspector both use command menus in this way. Once the controlling process reads the blip, the process can execute the form (`send menu :execute item`) if the process wishes the item to be processed in the usual way for menu items.

:command-menu *t-or-nil* Initialization Option of **w:menu**
Gettable. Default: **nil**

If this option is **t** and the user clicks on a mouse button, **:command-menu** puts a **:menu** blip into the I/O buffer. The blip is a list like the following:

`(:menu item button-mask menu)`

where:

item is the menu item that was clicked on.

button-mask indicates which mouse button was used (as in **w:mouse-last-buttons**).

menu is the menu instance that was clicked on, in case you are using more than one.

This list can be read as an input character with the **read-any-char** function from any other window sharing the same input buffer.

Multiple Menus **14.2.6.2** Specifying **:highlighting** as **t** for a menu creates a *multiple menu*. A multiple menu asks the user to select any combination of menu items rather than a single item. The menu has a choice box (usually named Do It) at the bottom in addition to its menu items. Clicking or pressing the space bar when the mouse is on a menu item selects it or deselects it; the selected items are displayed in reverse video. Clicking on the Do It box or pressing the END key indicates that the user has finished selecting items from the menu.

:highlighting *t-or-nil* Initialization Option of **w:menu**
Gettable. Default: **nil**

If this option is **t**, the menu can have more than one choice highlighted (displayed in reverse video). This type of menu requires margin choices to execute. The list of items currently selected is contained in **w:highlighted-items**.

:highlighted-items *items* Initialization Option of **w:menu**
Gettable, settable. Default: **nil**

The list of items to be highlighted when the flavor is instantiated.

:add-highlighted-item *item* Method of **w:menu**
:remove-highlighted-item *item* Method of **w:menu**

Enables or disables, respectively, the highlighting of the menu item identified by *item* when the menu is displayed.

:highlighted-values	Method of w:menu
:set-highlighted-values <i>list</i>	Method of w:menu
:add-highlighted-value <i>value</i>	Method of w:menu
:remove-highlighted-value <i>value</i>	Method of w:menu

Analogous to the **:highlighted-items** methods of similar names, these methods refer to the items' values (that is, the result of executing them), instead of referring to items directly. For example, if the item list is an association list with the elements (*string . symbol*), **:highlighted-values** uses *symbol*. These methods work only for menu items that can be executed without side effects, not for item types like **:eval**, **:funcall**, and so on.

The argument is the list of menu items. For example, consider the following code:

```
(defvar simple-menu (make-instance 'w:menu
  :pop-up t
  :item-list '(("Hungry" :value food)
               ("Thirsty" :value water)
               ("Cold" :value sunshine)
               ("Hot" :value air-conditioner)
               ("Bored" :value read))
  :highlighting t
  :menu-margin-choices :doit-and-abort))

(defun highlighting ()
  (send simple-menu :choose)
  (send simple-menu :highlighted-values))

(highlighting)
```

The menu appears similar to the following after the user selects **Hot**, **Thirsty**, and **Bored**:



Clicking (or pressing the space bar) on some items and then clicking on **Do It** (or pressing **END**) returns a list of the values associated with the items selected. Thus, if the user now clicks on **Do It**, the **highlighting** function returns a value of `'(read water air-conditioner)`.

Margin Choices **14.2.6.3** The following variables are the defaults for margin choices. If the user specifies a keyword for a margin choice, one of the strings contained in the following variables is substituted.

:menu-margin-choices <i>choices</i>	Initialization Option of w:menu
<i>Settable</i> . Default: nil	

Accepts a list containing the keywords **:abort**, **:doit**, and **:abort-and-doit** or other margin choices. A margin choice has the same format as a menu item (described in paragraph 14.2.1). This initialization option defines the default margin choices for a particular menu. If keywords are used, the string displayed by the margin choice is supplied by either or both the **w:margin-choice-completion-string** and the **w:margin-choice-abort-string** variables.

w:margin-choice-completion-string Variable
Default: "Do It"

The label by the margin choice box that, when selected, completes the selection. The margin choice is searched for when the user presses the END key.

w:margin-choice-abort-string Variable
Default: "Abort"

The label by the margin choice box that, when selected, aborts the selection. The margin choice is searched for during the processing of the abort-on-deexpose option.

*Permanent Versus
Pop-Up Menus*

14.2.6.4 Menus can be either permanent or pop-up menus. Permanent menus are always displayed; pop-up menus are displayed until the user makes a choice or moves the mouse outside the menu. Permanent menus appear slightly different from pop-up menus. Pop-up menus include a shadow border and, if a label exists, the label is in reverse video. Permanent menus do not include a shadow border and the label, if one exists, is in standard video.

:permanent *t-or-nil* Initialization Option of **w:menu**
Default: **t**

If this option is **t**, the menu is a permanent menu, that is, the menu does not disappear after the user selects an item or moves the mouse out of the menu. If this option is **nil**, the menu is temporary but not necessarily pop-up. A temporary menu remains exposed even when the user moves the mouse far outside the menu.

:pop-up *t-or-nil* Initialization Option of **w:menu**
Default: **nil**

If this option is **t**, the menu is a pop-up menu that disappears when the user moves the mouse well outside the menu. A pop-up menu exhibits a slight hysteresis, or margin of safety, to prevent the menu from disappearing when the user inadvertently moves the mouse to just outside the menu. The user must move the mouse outside this small margin of safety before the pop-up menu disappears.

Pop-up menus used to be called momentary menus. The name was changed to be more meaningful.

*Dynamic Item
List Menus*

14.2.6.5 *Dynamic item list menus* dynamically recompute the item list at various times. Whenever the program makes an explicit request to use the menu, the menu checks automatically to see whether its item list has changed. Any menu can be either dynamic or static.

:dynamic *t-or-nil* Initialization Option of **w:menu**
Default: **nil**

If this option is **t**, the menu can have an item list that is dynamically changed; that is, the menu item list is updated when necessary.

:add-item *item* Method of **w:menu**

Adds *item* to the end of the item list.

:item-list-pointer *function* Initialization Option of **w:menu**

Gettable, settable. Default: **nil**

Specifies the Lisp expression evaluated to recompute the current item list. This value is used when the menu is dynamic.

For example, consider the following code where `my-list` is not a simple list of menu items. The `compute-menu-list` function pulls the menu items from the list each time the menu is created.

```
(defparameter my-list (list `("a" able) `("b" baker) `("c" charlie)
                            `("d" dog) `("e" easy) `("f" fox)
                            `("g" giant) t))

(defun compute-menu-list ()
  (setq menu-list (list (cdr (first my-list))
                       (cdr (third my-list))
                       (cdr (nth (- (length my-list) 2) my-list))))
  'menu-list)
```



```
(progn
  (setq pop-up-menu (make-instance 'w:menu
                                  :item-list-pointer (compute-menu-list)
                                  :dynamic t
                                  :pop-up t
                                  :label 'computed-menu-list))
  (send pop-up-menu :choose)
)
```

Other Kinds of Menus **14.2.6.6** Menus can be single or multicolumn and scrolling or nonscrolling.

:multicolumn *t-or-nil* Initialization Option of **w:menu**
Default: **nil**

If this option is **t**, the menu has several dynamic columns. Each column comes from a separate menu item list that is recomputed as needed.

:scrolling-p *t-or-nil* Initialization Option of **w:menu**
Default: **t**

If this option is **t**, the menu can include a scroll bar if all the choices cannot be displayed at once in the menu. If this is specified as **nil**, scrolling is disabled. If scrolling is disabled, the user cannot use the mouse to move to the items that are not currently displayed by using the mouse; the user can reach them by using keystroke sequences such as **CTRL-V**, **META-V**, and so on.

Even if **:scrolling-p** is specified as **t**, the menu does not include a scroll bar if all items can be displayed without scrolling.

General Options **14.2.6.7** These are general-purpose initialization options to be used with any menu flavor.

:abort-on-deexpose *t-or-nil* Initialization Option of **w:menu**

If this option is **t** and the menu is deexposed without selecting an item, the command menu automatically executes an abort item.

- If **:abort-on-deexpose** is **nil**, this search is not made. Instead, **nil** is returned.
- If **:abort-on-deexpose** is **t**, an attempt is made to find an abort item to execute when the menu becomes deexposed.

1. The attempt to find an abort item starts by looking for a margin choice that has "Abort" as its value (case is not significant).
2. If no abort margin choice is found, then an abort menu item is searched for.
3. If no abort menu item is found, then no menu item is executed.

`:item-list` *item-list*

Initialization Option of `w:menu`

Gettable, settable. Default: nil

Sets the list of items (choices) for a menu. Setting the item list with the method recomputes the geometry. *item-list* specifies the list of items for the menu. (See paragraph 14.2.1, Menu Items, for a discussion of how to specify menu items.)

The following example shows how to add colors to the label of a pop-up menu, as well as individual elements of the menu. The menu has a dark-blue background with menu selections in white, unless overridden by a color directive. The label is cyan on a dark-gray background.

```
(defun items-in-color ()
  (let ((pop-up-menu (make-instance 'w:menu
    :pop-up t
    :label `(:string "Map Formats" :color ,w:cyan :background ,w:88%-gray-color)
    :foreground-color w:white
    :background-color w:dark-blue
    :item-list `(("Standard Projection" :value 1
      :documentation "N level cartographic")
      ("Topographic" :value 2
      :documentation "4 color standard statistical")
      ("Relief" :value 3
      :color ,w:yellow
      :documentation "Standard relief symbology")
      ("Hexagonal" :value 4
      :color ,w:green
      :documentation "Standard game board layout")
      ("3D-isoplanar" :value 4
      :documentation "Classified military format")))))
    (send pop-up-menu :choose)))
```

`:column-spec-list` *list*

Initialization Option of `w:menu`

Gettable, settable. Default: nil

Sets the column specification list for multicolumn menus. Each element of *list* specifies one column of the menu and has the format described in paragraph 14.2.2, Column Specification List.

The following example shows how to make a multicolumn menu with column headings in different colors. This menu uses a number of colors to illustrate how to use colors in various parts of a multicolumn menu. This example uses too many colors; normally, you should use color in a menu to draw attention to important areas.

```
(defvar column-1-list `(("Line" :value :line
  :documentation "Draw a line from A to B")
  ("Circle" :value :circle
  :documentation "Draw a circle centered at A")
  ("Rectangle" :value :rect
  :color ,w:red
  :documentation "Draw a rectangle with upper left at A")
  ("Ellipse" :value :ellipse
  :documentation "Draw an ellipse centered at A")))
```

```

(defvar column-2-list `(("Translate" :value :trans
  :color ,w:green
  :documentation "Translate selected image from A to B")
  ("Rotate" :value :rotate
  :documentation "Rotate selected image around A")
  ("Scale" :value :scale
  :documentation "Change the size of the selected image"))

(defvar column-3-list `(("Near Plane" :value :near
  :color ,w:cyan
  :documentation "Toggle near clipping plane on or off")
  ("Far Plane" :value :far
  :documentation "Toggle far clipping plane on or off")
  ("World Window" :value :world
  :documentation "Set the window into the world space")
  ("Display Viewport" :value :Viewport
  :color ,w:orange
  :documentation "Set the viewport onto the display screen"))

(defun color-multicolumn-menu ()
  (let* ((pop-up-menu (make-instance 'w:menu
    :multicolumn t
    :item-alignment :center
    :column-spec-list `(("Operations:" column-1-list
      :font fonts:mets
      :color ,w:yellow)
      ("Transforms:" column-2-list
      :font fonts:mets
      :color ,w:12%-gray-color)
      ("Viewing Parameters:" column-3-list
      :font fonts:mets
      :color ,w:yellow))
    :pop-up t
    :save-bits nil
    :foreground-color w:white ; default for writing
    ; menu entries
    :background-color w:66%-gray-color
    :label `(:centered :font fonts:mets
      :string "Graphics Operations Menu"
      :color ,w:cyan
      :background ,w:dark-blue)))
  )
  (send pop-up-menu :choose)))

```

:last-item Method of **w:menu**

Settable.

Returns the last item selected with a mouse click in this menu, or *nil* if none has been selected yet. The **w:last-item** instance variable is used for positioning the mouse for pop-up menus. (See the **:default-item** keyword argument of the **w:menu-choose** function, described in paragraph 14.2.4.1.)

:chosen-item Method of **w:menu**

Settable. Default: *nil*

Returns the selected item. The value returned is updated each time an item is selected. A program can be put into a wait state by setting the **w:chosen-item** instance variable to *nil* and waiting for it to become non-*nil*.

:process-character Method of **w:menu**

Implements the keyboard interface for menus. The keystrokes that are defined by default are contained in the **w:menu-default-command-characters** variable. You can supply additional commands for a menu by using the **:command-characters** initialization option. If the user presses a keystroke that is not defined, the system sounds a beep.

`:command-characters` *item-list*

Initialization Option of `w:menu`

Settable. Default: `nil`

Sets the association list of keyboard commands specialized for a particular menu. When a character from the keyboard is processed, the `command-characters` alist is examined first. If the character is not handled by that list, then the `w:menu-default-command-characters` list is examined. The `w:menu-default-command-characters` variable contains a list of the default movement commands.

Geometry

14.2.7 The way a menu is displayed can be described by the `:rows` and `:columns` initialization options. In addition, a menu can be described by six parameters that are collectively called its *geometry*. Each of these parameters can be specified as a constraint or can be allowed to default based on the item list and the parameters that are constrained.

Filled Versus Columnar Format

14.2.7.1 There are two styles of arranging the choices in the menu. They can be in an array of rows and columns, or they can be filled (that is, each line has as many choices as fit with a reasonable amount of whitespace in between). In columnar format, each line has about the same number of choices: the same as the number of columns. This is not true in filled format.

For example, suppose you have a menu item list as follows:

```
(setq menu-list '(able baker charlie dog easy fox giant house island jack kite lion))
```

Consider the following examples of menus created using this menu item.



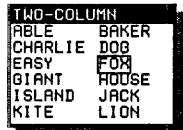
```
(setq filled-nil-menu
  (make-instance 'w:menu
                :item-list menu-list
                :pop-up t
                :label 'filled-nil))
(send filled-nil-menu :choose)
```

The menu is as wide as needed to display the widest item (in this case, the label) and is tall enough to display all the items without scrolling.



```
(setq filled-t-menu
  (make-instance 'w:menu
                :item-list menu-list
                :pop-up t :fill-p t
                :label 'filled-t))
(send filled-t-menu :choose)
```

The menu fills so the ratio of the menu's height to its width is approximately the golden ratio.



```
(setq two-column-menu
  (make-instance 'w:menu
                :item-list menu-list
                :pop-up t
                :columns 2
                :label 'two-column))
(send two-column-menu :choose)
```

THREE-COLUMN		
ABLE	BAKER	CHARLIE
DOG	EASY	FOX
GIANT	HOUSE	ISLAND
JACK	KITE	LION

```
(setq three-column-menu
      (make-instance 'w:menu
                    :item-list menu-list
                    :pop-up t
                    :columns 3
                    :label 'three-column))
(send three-column-menu :choose)
```

TWO-ROW					
ABLE	BAKER	CHARLIE	DOG	EASY	FOX
GIANT	HOUSE	ISLAND	JACK	KITE	LION

```
(setq two-row-menu (make-instance 'w:menu
                                  :item-list menu-list
                                  :pop-up t
                                  :rows 2
                                  :label 'two-row))
(send two-row-menu :choose)
```

You can explicitly specify which menu item is to go into a specific column by using a column specification list.

Elements of a Geometry List

14.2.7.2 The geometry of a menu is represented as a list of six elements, one for each parameter:

- The number of columns, or 0 for filled format
- The number of rows
- The inside width and height of the window, in pixels
- The maximum width and height of the window, in pixels. These parameters are meaningful only as constraints, because the way the menu is displayed is sufficiently described by its actual width or height. If the maximum width is constrained, the system chooses a tall, thin shape rather than exceeding this constraint. If the maximum height is constrained, the system chooses a short, fat shape rather than exceeding this constraint.

For the first four parameters, you must distinguish between the current value and the imposed constraint. The constraint values can be `nil`, meaning that this parameter is not to be constrained.

The last two parameters exist only as constraints and can be `nil`.

The actual display of a menu is based on four parameters: the number of rows, the number of columns (or whether to use fill mode), the height, and the width. Some of these parameters can be specified by constraints; others can be specified on a one-time basis when the menu is displayed. The rest of the parameters are chosen on the basis of constraints already known and the item list.

The default geometry constraints are all `nil`, meaning that the system can choose the size and shape freely, based on the specified item list. The default shape is an upright rectangle using columnar format with as many columns as fit in the width. Most small menus have only one column.

If both the height and width are specified (either precisely or indirectly) in such a way that not all the items can fit, the menu has a scroll bar that allows the user to see all the items. The user can also use the keyboard to move to items that are not currently visible.

When the item list of a menu is changed, the display of the menu is recomputed on the basis of the new item list and the geometry.

Initialization Options and Methods 14.2.7.3 The following initialization options and methods manipulate the geometry of a menu.

:geometry *list-of-specs* Initialization Option of **w:menu**
Gettable. Default: '(nil nil nil nil nil nil)'

:current-geometry Method of **w:menu**

A list of six elements representing the geometry (constraints) of the menu: the number of columns (0 for filled menu), number of rows, inside width, inside height, maximum width, and maximum height. The default for **:geometry** specifies nil for all six elements, which allows any aspect of the menu to change.

For **:current-geometry**, these items correspond to the actual current state of the menu. In the current geometry, the first four elements—which are sufficient to describe the current state—are never nil. The last two elements are the constraint values for the maximum width and height and can be nil.

:set-geometry &optional *columns rows inside-width inside-height max-width max-height* Method of **w:menu**

Sets the geometry (the constraints) from the arguments. **:set-geometry** uses six separate arguments rather than a list of six elements, as the **:geometry** method and initialization option do; this arrangement allows you to omit some of the arguments. The menu may change its shape and redisplay when its geometry changes.

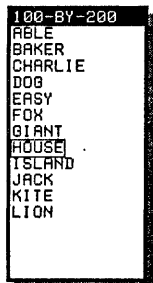
For **:set-geometry**, an explicit argument of **nil** makes the corresponding aspect of the geometry unconstrained. An omitted argument or an argument of **t** leaves the corresponding aspect of the geometry the way it is (if unconstrained, it remains so).

For example, consider the following menus:



```
(setq geo-spec1
      (make-instance 'w:menu
                    :item-list menu-list
                    :pop-up t
                    :label 'no-geo
                    :geometry '(nil nil nil nil nil nil)))
(send geo-spec1 :choose)
```

This menu is the same as one created with **:filled-p nil**; that is, the menu is as wide as needed to display the widest item and as high as needed to display all the items without scrolling.



```
(setq geo-spec2
      (make-instance 'w:menu
                    :item-list menu-list
                    :pop-up t
                    :label '100-by-200
                    :geometry '(nil nil 100 200 100 200)))
(send geo-spec2 :choose)
```

This menu is 100 pixels high by 200 pixels wide, regardless of the amount of space needed to display the items.



```
(setq geo-spec3
      (make-instance 'w:menu
                    :item-list menu-list
                    :pop-up t
                    :label 'at-most
                    :geometry '(nil nil nil nil 100 200)))
(send geo-spec3 :choose)
```

This menu can be up to 100 pixels high by 200 pixels wide, depending on the amount of space needed to display the items. If less space is needed, less space is used. Note that the menu appears similar to the menu stored in `geo-spec1`, where the geometry list is all `nil`.



```
(setq geo-spec4
      (make-instance 'w:menu
                    :item-list menu-list
                    :pop-up t
                    :label 'too-small
                    :geometry '(nil nil nil nil 50 100)))
(send geo-spec4 :choose)
```

This menu has a geometry specified that is too small to display the longest items, which are the label and the item called charlie. The label is truncated as well as part of the e in charlie. In addition, scrolling is enabled.



```
(setq geo-spec5
      (make-instance 'w:menu
                    :item-list menu-list
                    :pop-up t
                    :label 'way-too-small
                    :geometry '(nil nil nil nil 25 100)))
(send geo-spec5 :choose)
```

This menu has a geometry specified that is too small to display many of the items. The items are truncated on the right, and the user can still choose these items by boxing the portion of the item that is visible. Scrolling is enabled.

- :fill-p** *t-or-nil* Initialization Option of `w:menu`
Gettable, settable.
 Specifies whether to use a filled format (`t`) rather than a columnar format (`nil`). See the examples in paragraph 14.2.7.1, Filled Versus Columnar Format.
- :rows** *n-rows* Initialization Option of `w:menu`
 Default: `nil`
- :columns** *n-columns* Initialization Option of `w:menu`
 Default: `nil`
 Set the number of rows or columns, respectively, to the value specified by the argument. If `:columns` is specified as 0 or as `nil`, the menu is in fill mode.
- :column-row-size** Method of `w:menu`
 Returns two values: the width of a column in bits and the height of a row in pixels.
- :set-position** *new-x new-y &optional option* Method of `w:menu`
 Passes *new-x* and *new-y* to the `:set-edges` method to redraw the menu, thereby setting the current position of the menu. The size of the menu remains the

same. *new-x* and *new-y* are in pixels relative to the upper left corner of the menu's superior.

option makes the specified size a permanent constraint for the menu. *option* can be `:temporary` or `nil`. If *option* is `:temporary`, the menu is redisplayed with the specified edges for now, but once it is redisplayed for any reason, the permanent constraints (or lack of them) reemerge.

w:menu-compute-geometry *draw-p* &optional *inside-width inside-height* Function

Computes the current display parameters from the constraints, the item list, and the default font. This function is a subroutine of various menu methods and, when called, it assumes that `self` is bound to the menu being manipulated.

Arguments: *draw-p* — Whether the menu is redrawn. If *draw-p* is non-`nil`, the menu is redrawn.

inside-width, inside-height — Constraints for this invocation only, overriding any permanent constraints for those parameters. These arguments are used if the function is recomputing the row layout but is using a particular shape, which is the case if the menu has been reshaped by the user.

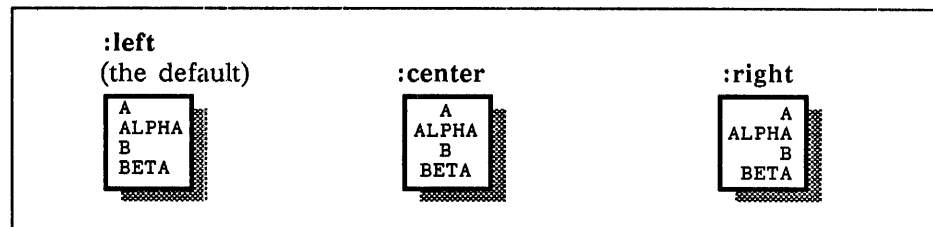
Menu Format 14.2.8 In addition to the geometry of a menu, several other initialization options, methods, and variables determine how a menu is formatted.

:default-font *font* Initialization Option of **w:menu**
Settable. Default: `w:*menu-item-standard-font*`, initially `cpfont`
 Sets the default font, used to display items that do not specify a font.

:item-alignment *alignment* Initialization Option of **w:menu**
 Default: `:left`

How the items are aligned in the columns of the menu. *alignment* can be one of `:left`, `:center`, or `:right`. The following table shows examples of each of these alignments, obtained by substituting the appropriate keyword for alignment in the following code.

```
(w:menu-choose '(a alpha b beta) :item-alignment :left)
```



w:menu-fill-breakage Variable
 Default: 60.

The estimated spacing in pixels needed for each item in a filled menu.

w:menu-golden-ratio Variable
 Default: 1.6s0

The aspect ratio used for menus when there are no constraints.

- w:menu-intercolumn-spacing** Variable
 Default: 10.
 The minimum spacing in pixels between columns in non-filled menus.
- w:menu-interword-spacing** Variable
 Default: 27.
 The minimum spacing in pixels between items used in filled menus.
- w:*default-menu-item-who-line-documentation-function*** Variable
 Default: **w:default-menu-item-documenter**
 The function to call to document menu items when the menu item is not documented. If you create a special function, this function should accept one argument, a menu item, and return either the documentation for that item or **nil**.
- :mouse-standard-blinker** Method of **w:menu**
 Default: A small cross
 Returns the default mouse blinker for menus. If you want to use a different default mouse blinker, you should redefine this method. When the mouse is over an item, the mouse blinker changes into a very small dot—the **w:mouse-glyph-small-dot** character (·)—so the mouse blinker is visible but does not obscure the item. If the mouse blinker was not changed, the menu item would be more difficult to read.
- :sort *predicate-and-key*** Initialization Option of **w:menu**
Settable. Default: **nil**
 Specifies how the items in the menu item list are sorted before displaying. **:sort** can be one of the following:
- | | |
|-------------------------------------------|---------------------------------|
| :ascending , or t for short | (:ascending <i>key</i>) |
| :descending | (:descending <i>key</i>) |
| <i>predicate</i> | (<i>predicate key</i>) |
| nil | |
- where *predicate* and *key* are as required by the **sort** function. The argument passed to *key* is a string displayed for a menu item; in the case of an icon item, it is a string containing the name of the icon. Similarly, the arguments passed to *predicate* are either menu item strings or values returned by *key*.
- The default for **:sort**, **nil**, does not change the ordering of the menu items. Note that using **:sort** destructively modifies the item list.

Methods That Implement Type Value Item Types

14.2.9 The following methods implement some of the type value item types discussed in Table 14-1.

- :choose** Method of **w:menu**
 Moves the menu window to the current position of the mouse and exposes the menu window. The **:choose** method then waits for the user to make a finishing choice and returns the menu window to the same activate/expose status it had before the **:choose** method was invoked. Keyboard input is also handled by the **:choose** method. This method does much of the work involved with menus and is called by the **w:menu-choose** function.

:execute *item* Method of **w:menu**

Processes *item*, computing and returning the value according to the rules described in Table 14-1. For most kinds of menus, this method is invoked automatically as part of the **:choose** method. An exception to this is a command menu, which must invoke the **:execute** method explicitly.

:execute-no-side-effects *item* Method of **w:menu**

Processes *item*, computing and returning the value, provided that this can be done without side effects. If computing the value might possibly have side effects (such as for item types **:eval**, **:funcall**, **:kbd**, **:window-op**, **:menu**, and **:menu-choose**), the value is not computed and **nil** is returned. The **:execute-no-side-effects** method is used by the highlighting option.

:execute-window-op *function* Method of **w:menu**

Implements the **:window-op** type value as described in Table 14-1.

:mouse-buttons-on-item *buttons-down-mask* Method of **w:menu**

Is invoked by the mouse process when the mouse blinker is positioned on an item and a mouse button is clicked. **:mouse-buttons-on-item** does everything that should be done by the mouse process when a mouse button is clicked. Specifically, the method records the chosen item and processes the **:buttons** item type when it is used.

Methods That Operate on Menus

14.2.10 The following methods deal with the size and position of the chosen item and where the menu is displayed.

:current-item Method of **w:menu**

Returns the item at which the mouse is pointing, or **nil**.

:item-cursorpos *item* Method of **w:menu**

Returns two values, as **:read-cursorpos** does, giving the x and y coordinates of the center of the displayed representation of *item* relative to the inside coordinates of the menu. The result is **nil** if the item is scrolled off the display.

:item-rectangle *item* Method of **w:menu**

Returns four values: the left, top, right, and bottom coordinates—inside the margins—of the rectangle enclosing the displayed representation of the specified item. The coordinates include one pixel of margin all around. The result is **nil** if the item is scrolled off the display. Note that, because of the one pixel margin, the returned values can be outside the window.

:menu-draw Method of **w:menu**

Draws the menu's display. The system automatically invokes **:menu-draw** when required, and **:menu-draw** should not be used in application programs. However, user-defined menu flavors can redefine the **:menu-draw** method or add methods to it.

:move-near-window *window* Method of **w:menu**

Exposes the menu below *window*, giving the menu the same width as the window. If the menu will not fit below *window*, **:move-near-window** then

tries to expose the menu above *window*. If the menu is too large to fit within *window*, scrolling can be enabled.

:center-around *x y*

Method of **w:menu**

Displays the menu so that, if possible, the last item chosen appears centered on the given coordinates. *x* and *y* are relative to the inside upper left corner of the window's superior. If displaying the menu as specified would cause part of the menu to be outside of its superior, the menu is offset slightly to keep it inside its superior. The actual coordinates of the center of the last item chosen are returned. (You might want to put the mouse there.) Pop-up menus use this method to put the menu in such a place that the mouse blinker appears over the last item chosen.

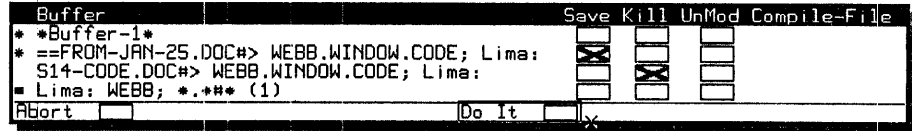
**Methods That
Reposition the
Menu Current Item**

14.2.11 The following methods of **w:menu** deal with positioning the mouse cursor onto a different item. These methods are invoked when the user presses a keystroke sequence to move to a new item on the menu.

Method	Invoked by Pressing	Description
:move-cursor-next-line	RETURN	Moves the mouse cursor to the first item in the next line.
:move-cursor-top	META-<	Moves the cursor to the first item.
:move-cursor-bottom	META->	Moves the cursor to the last item.
:move-cursor-up	CTRL-P or ↑	Moves the mouse cursor up to the previous line.
:move-cursor-down	CTRL-N or ↓	Moves the mouse cursor down to the next line.
:move-cursor-forward	CTRL-F or →	Moves the mouse cursor right.
:move-cursor-backward	CTRL-B or ←	Moves the mouse cursor left.
:move-cursor-first	CTRL-A	Moves the mouse cursor to the first item in this row.
:move-cursor-last	CTRL-E	Moves the mouse cursor to the last item in this row.
:move-cursor-page-down	CTRL-V	Moves the mouse cursor to the first item on the next page.
:move-cursor-page-up	META-V	Moves the mouse cursor to the first item on the previous page.
:search <i>character</i>	CTRL-S or CTRL-R	Searches forward or backward for an item. If <i>character</i> is CTRL-S then the search is in the forward direction. If <i>character</i> is CTRL-R then the search is in the backward direction. The user is prompted for the characters being searched for.

Multiple-Choice Facility

14.3 The *multiple-choice facility* provides a window containing several items, one per text line, and several choices for each item. To see an example of a multiple-choice window, invoke the Kill or Save Buffers menu from the Zmacs editor.



For each item, there can be several yes/no choices for the user to make. The same set of choices is given for each item (though some items may omit some choices). For example, in the Kill or Save Buffers command, there is an item (a line) for each buffer, and each line offers the choices Save, Kill, and Unmod. Choices of the same kind for different items form a column, with a heading at the top describing that choice. The leftmost column contains the text naming each item. The remaining columns contain small boxes, called *choice boxes*. A no box has a blank center, while a yes box contains an X inside the box. Moving the mouse blinker to a choice box and clicking the left button turns the choice on or off. Each choice can be initialized by the program to yes or no as appropriate for a default.

Each item choice can have constraints. For example, if you want the choices to be mutually exclusive, you set up constraints so that clicking one choice box to yes automatically sets the other choice boxes on the same line to no.

A multiple-choice window can have more lines of choices to offer than the window has lines. In this case, the user can scroll to reveal more choices, because the multiple-choice window is a type of text scroll window.

Multiple-Choice Functional Interface

14.3.1 The `w:multiple-choose` function is the interface to the multiple-choice facility.

`w:multiple-choose` *item-name item-list keyword-alist* Function
 &optional (*near-mode* '(:mouse))(*maxlines* 20.) *superior*

Pops up a multiple-choice window and allows the user to make choices with the mouse. The dimensions of the window are automatically chosen for the best presentation of the specified choices. If there are more choices available than can be shown within the maximum number of lines, scrolling is enabled.

When the user positions the mouse blinker on either Do It or Abort in the bottom margin and presses the appropriate mouse button, the multiple-choice window disappears and `w:multiple-choose` returns two values:

1. A list of selected choices of the form (*item selected-choices* ...), or nil if the user aborted.
2. The reason the function was exited — nil for a normal exit, `:abort` if the user aborted.

Arguments: *item-name* — A string that is the column heading for items. In the editor example, it is "Buffer".

item-list — A list of representations of items. Each element is a list of the following form:

(*item name choices*)

where:

item is any arbitrary object, such as an editor buffer. *item* is supplied so that it appears in the returned value and therefore allows you to identify what the returned answers apply to.

name is a string that names that object; it is displayed to the left on the line of the display devoted to this item.

choices is a list of keywords representing the choices the user can make for this item. Each element of *choices* is one of **t**, **nil**, a keyword, or a list of the following form:

(*keyword default*)

where:

keyword is a keyword mentioned in *keyword-alist*.

If *default* is present and non-**nil**, the choice is initially yes; otherwise, it is initially no. This is how the editor initializes the Save choice to be yes for a modified buffer in the Kill or Save Buffers editor command.

keyword-alist — A list defining all the choice keywords allowed. Each element takes the form:

(*keyword name &optional implications*)

where:

keyword is one of **t**, **nil**, a keyword, or a list, as described in the choices field of an *item-list* element.

name is a string used to name that keyword. This value is used as the column heading for the associated column of choice boxes.

implications are up to four additional list elements for each element. These implications control what happens to other choices for the same item when this choice is selected by the user. Each implication can be **nil** (meaning no implication), a list of choice keywords, or **t** (meaning all other choices).

The first implication is *on-positive*; it specifies which other choices are also set to yes when the user sets this one to yes.

The second implication is *on-negative*; it specifies which other choices are set to no when the user sets this implication to yes.

The third and fourth implications are *off-positive* and *off-negative*; they take effect when the user sets this choice to no. The default implications are **nil**, **t**, **nil**, and **nil**, respectively. In other words, the default is for the choices that are to be mutually exclusive.

The editor Kill or Save Buffers command specifies the implications as shown in the following example so that Unmod cannot be chosen when either Save or Kill is chosen.

```
((:save "Save" nil (:not-modified) nil nil)
 (:kill "Kill" nil (:not-modified) nil nil)
 (:not-modified "UnMod" nil (:save :kill) nil nil)
 (:compile "QC-FILE" nil nil nil nil))
```

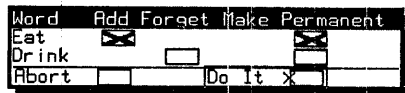
near-mode — Where the window is displayed. See the description of the `:expose-near` method in paragraph 5.7.3, Symbols That Manipulate Screen Arrays and Exposure, for values to specify for the *near-mode* argument.

maxlines — The maximum number of lines that the `w:multiple-choose` function displays without enabling scrolling. The default is 20.

superior — The superior of this menu. If *superior* is not specified, *near-mode* is examined. If *near-mode* specifies a `:window` option, then the superior of that window is used as the superior of the multiple-choose window. Otherwise, `w:mouse-sheet` is used as the superior of the multiple-choose window.

The code in this example offers the possibilities of `:add` or `:make-permanent` for `:eat` and the possibilities of `:forget` or `:make-permanent` for `:drink`. Presumably, these possibilities would be used because `:drink` has already been added and `:eat` has not.

```
(w:multiple-choose "Word"
 '(:eat "Eat" (:add :make-permanent))
 (:drink "Drink" (:forget :make-permanent)))
 '(:add "Add" nil nil nil (:make-permanent))
 (:forget "Forget" nil (:make-permanent) nil nil)
 (:make-permanent
 "Make Permanent" (:add) (:forget) nil nil)))
```



The implications indicate that making permanent is incompatible with forgetting when forgetting is possible, and require adding when adding is possible.

The following value might be returned:

```
((:eat :make-permanent :add) (:drink))
```

In this example, the items are keywords (symbols), but that is not significant. The system never looks inside the items; it simply compares the items with `eq` and puts the items in the returned value.

Making Your Own Multiple-Choice Windows

14.3.2 The following flavors and the related instance variables, resources, initialization options, and methods provide multiple-choice facilities.

`w:basic-multiple-choice`

Flavor

The basic flavor that implements the multiple-choice facility. Like most basic mixins, `w:basic-multiple-choice` is not itself instantiable but does commit any window that incorporates it to being a `w:basic-multiple-choice` multiple-choice window rather than a different sort of window.

:item-name *name* Initialization Option of **w:basic-multiple-choice**
Gettable, settable. Default: nil
 Sets a string used as the column heading for items.

:item-list *item-list* Initialization Option of **w:basic-multiple-choice**
Gettable, settable. Default: nil
 Initializes the window's item list to *item-list*, a list of representations of the rows of the multiple-choice window. Each element is a list of the following form:

(*item name choices*)

where:

item is any arbitrary object, such as an editor buffer.

name is a string that names that object; it is displayed to the left on the line of the display devoted to this item.

choices is a list of keywords representing the choices the user can make for this item. Each element of *choices* is either a symbol, a keyword, or a list of the following form:

(*keyword default*)

where:

keyword is a keyword mentioned in the *keyword-alist* argument of the **:setup** method.

If *default* is present and non-nil, the choice is initially yes; otherwise, it is initially no. This is how the editor initializes the Save choice to be yes for a modified buffer in the Kill or Save Buffers editor command.

:choice-types *alist* Initialization Option of **w:basic-multiple-choice**
Gettable, settable.
 Sets the window's keyword association list for the columns of the multiple-choice window.

:setup *item-name keyword-alist finishing-choices item-list* Method of
 &optional (*maxlines* 20.) **w:basic-multiple-choice**
 Sets up all the various parameters of the window. Usually it is used while the window is deexposed. The window first decides what size it should be and whether all the items will fit or scrolling is required, then draws the display into its bit-array. Thus, when the window is exposed, the display will appear instantaneously.

Arguments: *item-name* — A string that is the column heading for items. In the case of the editor example, it is "Buffer".

keyword-alist — A list defining all the choice keywords allowed, as explained previously in the discussion of the *keyword-alist* argument for the **w:multiple-choose** function.

finishing-choices — The margin choices that go in the bottom margin. When the user clicks on one of these, he or she is finished with a particular

menu. The `w:default-finishing-choices` variable contains a reasonable default for this, providing Do It and Abort choices.

item-list — A list of representations of items, as explained previously in the discussion of the `:item-list` initialization option of `w:basic-multiple-choice`.

maxlines — The maximum number of lines that the `:setup` method displays without enabling scrolling.

`w:choice-value` Instance Variable of `w:basic-multiple-choice`

When the mouse process sets the `w:choice-value` instance variable to non-nil, the `:choose` method completes execution.

`:choose` &optional *near-mode* Method of `w:basic-multiple-choice`

Moves the window to the place specified by the *near-mode* argument and exposes the window. The `:choose` method then waits for the user to make a finishing choice and returns the window to the same activate/expose status the window had before the `:choose` method was invoked. The `:choose` method returns the same value as the function `w:multiple-choose`.

The *near-mode* argument is where the window is exposed. The default is the `(:mouse)` list. (See the `:expose-near` method in paragraph 5.7.3, Symbols That Manipulate Screen Arrays and Exposure, for values to specify for the *near-mode* argument.)

`w:margin-choices` Instance Variable of `w:basic-multiple-choice`

Default: `w:default-finishing-choices`, initially "Abort" and "Do it"

The choices displayed at the bottom of the multiple choice menu.

`w:multiple-choice` Flavor

An instantiable window that includes the `w>window` flavor with the multiple-choice facility built in. The window has borders and a label area on top used for column headings.

`w:temporary-multiple-choice-window` Flavor

A multiple-choice window that pops up.

`w:temporary-multiple-choice-window` Resource

&optional (*superior* `w:mouse-sheet`)

A resource of temporary multiple-choice windows. The `w:multiple-choose` function uses the `w:temporary-multiple-choice-window` resource. *superior* specifies the superior window for this window.

Choose-Variable-Values Facility

14.4 The choose-variable-values facility presents the user with a display of several Lisp variables and their values. The user can change the value of the variables by using the mouse and the keyboard. When the user has changed the variables to the values desired, he or she can then exit from the choose-variable-values window.

The choose-variable-values window is a kind of text scroll window, so each line of the display corresponds to one variable. The name of the variable, a colon, and the value of the variable are displayed. Pointing the mouse at the

value causes a box to appear around it. Clicking the left mouse button or pressing the LEFT key at that point allows the value to be changed. The arrow keys can be used to position the box around a value.

For example, the Edit Attributes option in the System menu, shown below, is a choose-variable-values window.

```

Edit window attributes of Edit: TEST-FRAME.LISP#> WEBB; Lima:
Current font: ..... HL12 HL12B HL12BI CPTFONT
More processing enabled:.. Yes No
Reverse video: ..... Yes No
Vertical spacing: ..... 2
Deexposed typein action:.. Wait until exposed Notify user
Deexposed typeout action: Wait until exposed Notify user Let it happen Signal error Other
("Other" value of above): NIL
ALU function for drawing: IOR ANDCA XOR
ALU function for erasing: IOR ANDCA XOR
Screen manager priority:.. NIL
Save bits: ..... Yes No
Label: ..... NIL
Width of borders: ..... 1
Width of border margins:.. 1
Abort  Do It 

```

Specifying the Variables

14.4.1 When you use a choose-variable-values window, you must specify one or more variables with a list of specifiers. You pass the list as an argument to `w:choose-variable-values`. Each variable must be a special variable, that is, a variable that is declared special, defined in a `defvar` or a `defparameter`, and so on. (See the *Explorer Lisp Reference* manual for more information about special variables.)

Each variable has a *type* that controls the values it can take on, the way the values are displayed, and the way the user enters new values. The type mechanism is extensible and is described in detail later. The types fall into two categories: those with a small number of legal values and those with a large number of legal values. The first type displays all the choices, with the one that is the current value of the variable in boldface. Pointing at a choice and clicking the mouse sets the variable to that value. Types with a large number of legal values display the current value. Pointing at the value and clicking the mouse allows a new value to be entered from the keyboard. Rubbing out more characters than were typed restores the original value instead of changing it.

The variables themselves can be either symbols, which are effectively examined and set as special variables in the calling program's process, or locatives, whose contents are examined and set.

The variables can be specified by describing each line of the display as a separate item, or you can display related variables in a table.

Variables in Separate Items

14.4.1.1 Each line of the display is specified by an *item*, which can be one of—or a combination of—a string, a symbol, or a list:

- A string is simply displayed. This item is used to put headings and blank separating lines into the display.

NOTE: A string *cannot* be used by itself as the only item(s) in a variables list. A string must be used in conjunction with variables and other value types.

- A symbol whose value can be any Lisp object. The print name of the item symbol is displayed, and the input value is stored as the value of the symbol.
- A list of the general form:
(*variable name modifiers type args...*)

where:

variable is the variable whose value is being chosen. It is either a symbol or a locative.

name is optional if *variable* is a symbol; if it is omitted, it defaults to the print name of *variable*. If *variable* is a locative, you must supply *name*. *name* can be either a string, which is displayed as the name of the variable, or nil, meaning that this line should have no variable name but only a value.

modifiers are optional. If *modifiers* are supplied, they are either keywords or keyword-value pairs as described in Table 14-4. You can supply more than one modifier for an item.

type is an optional keyword giving the type of variable; if omitted, it defaults to `:sexp`.

args are possible additional specifications dependent on *type*. It is possible to omit *name* and supply *type* because *name* is always a string or nil, while *type* is always a non-nil symbol.

The following are some examples of items that can be included in an item list:

```

; The simplest case: a symbol
my-variable                               ; The label is MY-VARIABLE; it
                                           ; returns my-variable as its value

; A list with a variable and its name
(my-variable "Variable Name")           ; The label is Variable Name; it
                                           ; returns my-variable as its value

; A list with a variable, its name, and a type
(my-variable "Variable Name" :string)    ; The label is Variable Name; the value
                                           ; is stored as a string.

; A list with a variable, its name, a type, and an item modifier
(my-variable "Variable Name"            ; The name of the variable and its label
  :documentation "Specify a string. Do not use quotes" ; The modifier and modifier argument
  :string)                               ; The variable type

; A list with a variable, its name, a type with an argument, and two item modifiers
(my-variable "Variable Name"           ; The name of the variable and its label
  :documentation "Specify a string surrounded by quotes." ; One modifier and modifier argument
  :edit                               ; The second modifier
  :typep (string))                    ; The variable type

```

Of course, other combinations within an item in a list are possible: a variable without a name but with a type, a variable with a modifier but no name or type, and so on.

Variables in Tables

14.4.1.2 You can also specify variables in table format. The table specification item presents selected columns of a variable in a conventional tabular format and allows the user to edit individual table entries.

The format of the table is specified by an element in the specification list of the general form:

```
(variable (column-list-0)
          (column-list-1)
          (column-list-2)
          . . .
          (column-list-n))
```

where:

variable is a list treated like a table of data. *variable* can be either a symbol or a locative. Specifically, *variable* is a list of lists, where each sublist contains *n* or more elements and there are *m* sublists. The table contains a row for each sublist. The first element of each of the sublists makes up one column, the second element another column, and so on. Each column of the table must contain data of the same type.

each *column-list* specifies a column in the table. Columns are displayed left to right in the same order as the list of column-list specifications. *column-list* is a list of the form:

```
(column-index column-heading item-modifiers item-type args)
```

where:

column-index is an integer that is the zero-based index that points to a specific element in the sublist in *variable*. Thus, the columns need not be in the same order as the elements of the variable sublists, and not all the elements need to be displayed. In addition, columns can be repeated.

column-heading is a string that is used as the column heading. The length of the string determines the width of the column. You may need to pad the string with enough blanks to accommodate the widest value displayed in the column because displaying a value that is wider than the column width causes the remaining data in that row to shift to the right, ruining the alignment of elements in the columns.

item-modifiers are any of the item modifiers shown in Table 4-4. These modifiers are optional. If specified, a modifier applies to all elements of the column.

item-type controls how the data for that column is entered and displayed and is one of the item types described in Table 4-3.

args are possible additional specifications dependent on *item-type*.

A table specification item does not itself accept item modifiers. Instead, you can specify item modifiers within any *column-list*.

For example, the following variable contains a list with 6 sublists (that is, 6 unique rows). Each sublist contains 5 or more elements, which gives a possibility of 5 unique columns. Note that all the strings are padded with blanks so they are all the same length. This is produce a table with columns that line up.

```
(defvar table-variable `(("abc" 1 car "john " 3 1.314 2.1417)
                        ("def" 2 cdr "mary " 5)
                        ("ghi" 3 princ "albert" 7)
                        ("jkl" 4 read "jane " 11)
                        ("mno" 5 write "bob " 13)
                        ("pqr" 6 prin1 "esther" 17 .25)))
```

Next, consider the following code that specifies the columns of the display:

```
(w:choose-variable-values `((table-variable ; Refers to the defvar previously defined
  ; index number
  ; col heading optional item modifier and argument item type and optional type arguments
  (3 "name " :documentation "Who this is" :string)
  (4 "prime #" :documentation "Their favorite number" :number)
  (1 "ID #" :documentation "Their locker number" :number)
  (2 "function" :documentation " " :test functionp)
  (0 "letters" :documentation "Various letters" :string)
  (3 "name (again)" :documentation " " :string)))
```

Together, the variable and the specification produce a display similar to the following:

name	prime #	ID #	function	letters	name (again)
john	3	1	CAR	abc	john
mary	5	2	CDR	def	mary
albert	7	3	PRINC	ghi	albert
jane	11	4	READ	jkl	jane
bob	13	5	WRITE	mno	bob
esther	17	6	PRIN1	pqr	esther

Choose-Variable-Values Functional Interface

14.4.2 The `w:choose-variable-values` function is a functional interface for the choose-variable-values facility. Table 14-3 shows the predefined variable types that can be used by `w:choose-variable-values`. In general, any value input must be of the type specified for that variable. If the user inputs an unacceptable value, the system displays an error message in the choose-variable-values window and the user can then input a different value. Note that some keywords require one or more arguments; other keywords do not require arguments.

Table 14-3 Predefined Variable Types for w:choose-variable-values

Name	Description of Input Value
<code>:assoc</code> <i>values-list print-function</i>	One of the displayed choices. Each displayed choice is the car of an element of <i>values-list</i> ; the cdr is the value that is put in the variable. All the choices are displayed, with the current value in boldface. The user chooses a new value by pointing to it with the mouse and clicking. Comparison is done by equal rather than eq . <i>print-function</i> prints a value; <i>print-function</i> is optional and defaults to princ .
<code>:boolean</code>	Either t or nil . The choices are displayed as yes and no .
<code>:character</code>	A character object. It is printed as a character name using the <code>~:@C</code> format operator, and read as a single keystroke. If the key you want to identify is a special editing key, such as RUBOUT or CLEAR INPUT, you must precede this keystroke with CTRL-Q. Editing a value (clicking right) of type <code>:character</code> is not meaningful because the printed representation of every character is already at least one character/keystroke long. The edit operation simply takes the first character of the current printed representation and returns.
<code>:character-or-nil</code>	Like <code>:character</code> but nil is also allowed as the value. A value of nil displays as none and can be input using the CLEAR INPUT key.
<code>:choose</code> <i>values-list print-function</i>	One of the displayed choices. Each displayed choice is an element of <i>values-list</i> . All the choices are displayed, with the current value in boldface. The user chooses a new value by pointing to it with the mouse and clicking. Comparison is done by equal rather than eq . <i>print-function</i> prints a value; <i>print-function</i> is optional and defaults to princ .
<code>:date</code>	A universal date-time. It is printed with time:print-universal-time and read with the readline-trim and time:parse-universal-time functions.
<code>:date-or-never</code>	Either a universal date-time or nil . A value of nil is printed as never , and a number is printed using the time:print-universal-time function. Input is read with the readline-trim function. If the user enters a non- nil value, it is passed to the time:parse-universal-time function.
<code>:directory-pathname</code>	A directory portion of a pathname. The value is printed with princ and read with read-line , fs:parse-pathname , and fs:merge-pathname-defaults . The specified value is merged with the tv:*default-directory-pathname* global variable to produce the value stored in the variable. The tv:*default-directory-pathname* variable is initially bound to nil . See the <i>Explorer Input/Output Reference</i> manual for details about pathnames and defaults.
<code>:fixnum</code>	A fixnum that is printed by prin1 and read by read .
<code>:fixnum-or-nil</code>	Either a fixnum or nil .

Continued

Table 14-3 Predefined Variable Types for `w:choose-variable-values` (Continued)

Name	Description of Input Value
<code>:interval-or-never</code>	The actual value of the variable is either <code>nil</code> or the length of the interval in seconds. However, new values are read in using the <code>time:read-interval-or-never</code> function. This function does not accept a simple fixnum as a second specification; see the <i>Explorer Lisp Reference</i> manual for details. The value is displayed using <code>time:print-interval-or-never</code> .
<code>:list-of type-keyword args</code>	A list of values accepted by <i>type-keyword</i> , with the values separated by commas. The values must satisfy the type specified by <i>type-keyword</i> . In the printed representation, the values are separated by commas. For example, <code>:list-of :typep (integer 0 20)</code> would require the user to enter a list of integers in the range from 0 through 20, inclusive, with the integers separated by commas.
<code>:menu item-list</code>	One of the values in <i>item-list</i> . <i>item-list</i> is a list of menu items suitable for <code>w:menu-choose</code> . When a user clicks on a current value of variable type <code>:menu</code> , <code>w:choose-variable-values</code> invokes a pop-up menu that contains <i>item-list</i> . The value of the item chosen is stored as the variable value.
<code>:menu-alist item-list</code>	One of the displayed choices. All the choices are displayed, with the current value in boldface. The user chooses a new value by pointing to it with the mouse and clicking. <i>item-list</i> is a list of menu items that can contain the usual menu mechanisms for specifying the string to display, the value to return, and documentation. Each element of <i>item-list</i> must be a list.
<code>:multiple-menu arg</code>	One or more of the values available from <i>arg</i> . <i>arg</i> is evaluated and should return an item list suitable for the multiple menu option of <code>w:menu-choose</code> . When a user clicks on a current value of variable type <code>:multiple-menu</code> , <code>w:choose-variable-values</code> invokes a pop-up multiple (highlighting) menu that contains <i>item-list</i> . The value stored is a list of the menu item values chosen.
<code>:non-negative-fixnum</code>	A fixnum that is greater than or equal to zero.
<code>:number</code>	Any type of number. It is printed with <code>prin1</code> and read with <code>read</code> , but only a number is accepted as input.
<code>:number-or-nil</code>	Either a number or <code>nil</code> .
<code>:pathname</code>	A pathname. The value is printed with <code>princ</code> and read with <code>read-line</code> , <code>fs:parse-pathname</code> , and <code>fs:merge-pathname-defaults</code> . See the <i>Explorer Input/Output Reference</i> manual for details about pathnames and defaults.
<code>:pathname defaults</code>	A list of the form <code>(:pathname defaults)</code> where <i>defaults</i> is a pathname or a defaults association list to pass to <code>fs:merge-pathname-defaults</code> . <i>defaults</i> can also be a symbol whose value should be used. If this is the same variable this item is setting, then each typed-in value is merged with the previous setting.
<code>:pathname-list</code>	A list of pathnames. In the printed representation, they are separated by commas.

Table 14-3 Predefined Variable Types for `w:choose-variable-values` (Continued)

Name	Description of Input Value
<code>:pathname-or-nil</code>	Like <code>:pathname</code> but <code>nil</code> is also allowed. A value of <code>nil</code> is printed as a blank line.
<code>:princ</code>	The same as <code>:sexp</code> , except that the value is printed with <code>princ</code> rather than <code>prin1</code> .
<code>:set</code>	Like <code>:choose</code> , except that <code>:set</code> allows you to select more than one element of a set. The value of <code>:set</code> is the list of elements selected.
<code>:sexp</code> , <code>:list</code> , and <code>:any</code>	Any Lisp expression printed with <code>prin1</code> or read with <code>read</code> .
<code>:small-fraction</code>	Any number greater than 0 and less than or equal to 1.
<code>:string</code>	A string that is printed with <code>princ</code> or read with <code>read-line</code> . With this type, the user does not need to include quotation marks around the string. With the variable type <code>:typep string</code> the user must include the delimiting quotation marks.
<code>:string-list</code>	A list of strings, whose printed representation for input and output consists of the strings separated by commas and spaces.
<code>:string-or-nil</code>	Like <code>:string</code> but <code>nil</code> is also allowed as a value.
<code>:test function</code>	A value validated by <i>function</i> . The <i>function</i> argument is a function with one parameter as its value. For example, <code>:test functionp</code> would require that the user enter a function.
<code>:typep type-specifier</code>	A value validated by <i>type-specifier</i> . The <i>type-specifier</i> argument is a Common Lisp type specifier suitable for the <code>typep</code> function. For example, <code>:typep (integer 2 8)</code> would force the input value to be an integer in the range from 2 through 8, inclusive.

Table 14-4 lists the item modifiers that can be used with `w:choose-variable-values`. Each modifier is associated with one of the variables in the window. In general, a modifier is placed between the variable name (if one is present) and the variable type. A variable can have more than one modifier.

Table 14-4 Item Modifiers for `w:choose-variable-values`

Item	Description
<code>:constraint</code> <i>user-defined-function</i>	<p>Calls a user-defined function that checks the value entered to determine if the value is within the limits specified by <i>user-defined-function</i>. If the value is within the specified limits, <i>user-defined-function</i> returns <code>nil</code>. If the value is not within the specified limits, <i>user-defined-function</i> returns a string that is an error message.</p> <p><i>user-defined-function</i> takes four arguments:</p> <p><i>window</i> — The choose-variable-values window</p> <p><i>variable</i> — The variable part for the item</p> <p><i>old-value</i> — The old value of <i>variable</i></p> <p><i>new-value</i> — The new value of <i>variable</i></p>
<code>:documentation</code> <i>doc</i>	<p>Displays <i>doc</i> in the mouse documentation window when the mouse is pointing at this item. <i>doc</i> is a string or list. (The structure of this list is discussed in paragraph 11.6, How Windows Handle the Mouse, under the description of the <code>:who-line-documentation-string</code> method of <i>windows</i>.) If no documentation is supplied in this way, the default depends on the type, and generally is something like <code>click left to input a new value from the keyboard</code>. Note that the ordering of this modifier with the variable type is different from the syntax of similar elements of an item list used with the <code>w:menu-choose</code> function.</p>
<code>:edit</code>	<p>Puts the old value clicked on in the input editor buffer. This prevents the value from disappearing and allows the user to edit the value.</p>
<code>:quote</code>	<p>Puts a quote symbol in front of the value the user enters. This item modifier should only be used with a variable type.</p>
<code>:side-effect</code> <i>list-or-function</i>	<p>Stores the modified value and then handles <i>list-or-function</i>.</p> <ul style="list-style-type: none"> ■ If <i>list-or-function</i> is a list, the list is evaluated. ■ If <i>list-or-function</i> is a function, the function is called with four arguments: <i>window</i>, <i>variable</i>, <i>old-value</i>, and <i>new-value</i>. <p>The side effect produced by <i>list-or-function</i> may further modify (or even reset) the value that was originally modified.</p>

The following code illustrates the use of several of the item modifiers for a choose-variable-values window, including the function called from a :constraint modifier.

```
(defun check-amount (ignore variable item-value-for-set &optional ignore)
  "Validate the amount relative to the specified currency."
  variable
  (let* ((item-value item-value-for-set))
    (if (not (integerp item-value))
        (string-append
         "amount"."")
        nil)))

(defvar currency nil)
(defvar item-to-quote nil)
(defvar amount 0)

(w:choose-variable-values
  `((currency
     "Monetary unit"
     :menu-alist
     (("Dollar" :value dollar :documentation "Currency of the United States.")
      ("Pound" :value pound :documentation "Currency of Britain.")
      ("Yen" :value yen :documentation "Currency of Japan.")
      ("Ruble" :value ruble :documentation "Currency of the USSR.")))
     (amount "Numerical amount" :constraint check-amount)
     (item-to-quote "Item-to-quote" :quote
                    :documentation "This item will be quoted within a list." :fixnum)))
```

w:choose-variable-values *variables &rest options*

Function

Interfaces with the choose-variable-values facility. **w:choose-variable-values** pops up a window displaying the values of the specified variables and permits the user to alter them. One or more choice boxes (as in the multiple-choice facility) appear in the bottom margin of the window. When the user clicks on the Do It choice box, the window disappears and this function returns. The value returned is not meaningful; the result is expressed in the values of the variables.

There are several ways for the user to exit from the **w:choose-variable-values** function.

- Select the Do It margin choice box. **w:choose-variable-values** returns normally for this case.
- Press the END key. This has the same effect as selecting the Do It margin choice box.
- Press the ABORT key.
 - If an Abort margin choice exist, this has the same effect as selecting the margin choice box.
 - If an Abort margin choice *does not* exist, this signals a condition for **sys:abort** that can be trapped with an **unwind-protect** or a **condition-case**.

The system chooses the dimensions of the window and enables scrolling if there are too many variables to fit in the chosen height of the window.

Arguments: *variables* — A list of elements that are special variables of the types listed in Table 14-3, modified by the item modifiers listed in Table 14-4.

options — The usual list of alternating option keywords and argument values. Table 14-5 discusses the keywords that can be used as the *options* argument. These keyword-value pairs affect the `w:choose-variable-values` window as a whole rather than modifying an individual variable.

Table 14-5 The `w:choose-variable-values` Options Keywords

Name	Description
:extra-width	The amount of extra space to leave after all the items that have values when <code>:width</code> is not specified. This extra space allows for changing the value to something bigger. The extra space is specified as either a number of characters or a character string. The default is ten characters. If <code>:width</code> is specified, <code>:extra-width</code> is ignored.
:function	<p>A function to be called if the user changes the value of a variable. The default is <code>nil</code> (no function).</p> <p>This function can implement constraints among the variables. It is called with arguments <i>window</i>, <i>variable</i>, <i>old-value</i>, and <i>new-value</i>. The function should return <code>nil</code> only if the original variable needs to be redisplayed, or <code>t</code> if no redisplay is required; if <code>t</code>, this function would usually <code>setq</code> several of the variables, then perform a <code>:refresh</code> method on the window.</p>
:height	The height of the window in pixels. By default, the height of the window is large enough to accommodate the items up to a certain maximum. After that maximum, scrolling is activated. The <code>:height</code> keyword enables you to specify the height large enough to prevent the need to scroll.
:label	The argument to the <code>:label</code> keyword is a string that is the label to be displayed at the top of the window. The default is "Choose Variable Values".
:margin-choices	<p>A list of specifications for choice boxes to appear in the bottom margin. Each element in the list can be either a string or a list.</p> <ul style="list-style-type: none"> ■ If the element is a string, that string is used as a label for a margin choice. If any of these labels are selected, the <code>w:choose-variable-values</code> function returns. If no string appears in the list, by default the label <code>Do It</code> is added to the margin choices. ■ If the element is a list, then the first element in the sublist should be a string to use as a label, and the second element should be a form. If the form is the symbol <code>nil</code>, then the <code>w:choose-variable-values</code> function returns when this label is selected; if the form is not the symbol <code>nil</code>, that form is evaluated. If the return value of the form is true, then the <code>w:choose-variable-values</code> function returns; otherwise, <code>w:choose-variable-values</code> continues processing user input. Because this form is evaluated in the caller's process, it can do such things as alter the values of the special variables or exit using the <code>throw</code> function. <p>The default for <code>:margin-choices</code> is <code>Do It</code>.</p>
:near-mode	Where to position the window. This is a suitable argument for the <code>:expose-near</code> method. The default is the list <code>(:mouse)</code> . See paragraph 5.7.3, Symbols That Manipulate Screen Arrays and Exposure, for other suitable arguments.

Table 14-5 The `w:choose-variable-values` Options Keywords (Continued)

Name	Description								
<code>:reverse-video-p</code>	Whether the window displays white-on-black or black-on-white. <code>:reverse-video-p</code> is used as the argument of the <code>:set-reverse-video-p</code> method. The default is <code>nil</code> , which causes the display to be in regular video. A value of <code>t</code> causes the window to be displayed in reverse video.								
<code>:superior</code>	The window to which the pop-up choose-variable-values window is inferior. The default is one of the following, in decreasing order of precedence: <ul style="list-style-type: none"> ■ The superior of <code>w</code> if <code>:near-mode</code> is <code>(:window w)</code> ■ <code>w:selected-window</code> if it is running the <code>current-process</code> ■ <code>*terminal-io*</code> if it is running the <code>current-process</code> and <code>*terminal-io*</code> is a window ■ The value of <code>w:mouse-sheet</code> 								
<code>:value-tab</code>	Whether to tab the values past the choice labels. <code>:value-tab</code> can have one of the following values: <table border="1" data-bbox="500 890 1427 1276"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>t</code> (the default)</td> <td>Aligns the values in a column about two spaces to the right of the longest choice label.</td> </tr> <tr> <td><code>nil</code> or <code>0</code></td> <td>Prints each value about two spaces to the right of its choice label.</td> </tr> <tr> <td>A fixnum n</td> <td>n specifies the number of pixels from the inside left margin to the beginning of the column of values. If n is large enough to space past all the choice labels, the values are aligned in a column at n. If n is not as large as the widest of the choice labels, the values are printed as though <code>:value-tab</code> was <code>nil</code>.</td> </tr> </tbody> </table>	Value	Description	<code>t</code> (the default)	Aligns the values in a column about two spaces to the right of the longest choice label.	<code>nil</code> or <code>0</code>	Prints each value about two spaces to the right of its choice label.	A fixnum n	n specifies the number of pixels from the inside left margin to the beginning of the column of values. If n is large enough to space past all the choice labels, the values are aligned in a column at n . If n is not as large as the widest of the choice labels, the values are printed as though <code>:value-tab</code> was <code>nil</code> .
Value	Description								
<code>t</code> (the default)	Aligns the values in a column about two spaces to the right of the longest choice label.								
<code>nil</code> or <code>0</code>	Prints each value about two spaces to the right of its choice label.								
A fixnum n	n specifies the number of pixels from the inside left margin to the beginning of the column of values. If n is large enough to space past all the choice labels, the values are aligned in a column at n . If n is not as large as the widest of the choice labels, the values are printed as though <code>:value-tab</code> was <code>nil</code> .								
<code>:width</code>	How wide to make the window. This width can be a number of characters or a string (the window is made just wide enough to display that string). The default is to make the window wide enough to display the current values of all the variables, provided that the window is not too wide to fit in its superior.								
<code>:foreground-color</code>	The foreground (text) color of the menu.								
<code>:background-color</code>	The background color of the menu.								
<code>:label-color</code>	The label foreground (text) color for the menu.								
<code>:label-background</code>	The label background color for the menu.								

The following examples show how to call `w:choose-variable-values`, which displays the three variables' names and values and lets the user change them. `*print-base*`, `*read-base*`, and `*print-radix*` are all globally-defined special variables.

```
(w:choose-variable-values `(*print-base* *read-base* *print-radix*)
  :label "Number format parameters")
```

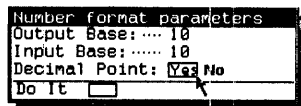


The following is the default mouse documentation for the margin choices:

```
L: input a new value from the keyboard, R: edit this value
```

The same example can be executed with better formatting by using the following code:

```
(w:choose-variable-values
  `((*print-base* "Output Base" :number)
    (*read-base* "Input Base" :number)
    (*print-radix* "Decimal Point" :boolean))
  :label "Number format parameters")
```



The following code adds several special margin choices. Note that the default margin choice, Do It, appears in addition to the choices you specify.

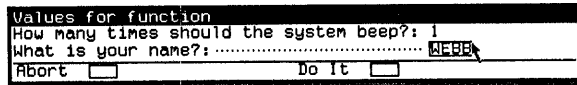
```
(w:choose-variable-values `(*print-base* *read-base* *print-radix*)
  :label "Number format parameters"
  :margin-choices `(("Abort" (signal-condition
    eh:*abort-object*))
    ("Help" (my-display-help-function))))
```



The following code defines a function that prompts the user for the number of times the system should beep and for the user's name. The function then beeps the required number of times and displays a message using the user's name. Note that the variables are declared special.

```
(defun call-cvv ()
  (let ((number-of-beeps 1)
        (your-name user-id))
    (declare (special number-of-beeps your-name))
    (w:choose-variable-values
      '((number-of-beeps "How many times should the system beep?" :typep (integer 0 10))
        (your-name "What is your name?" :string))
      )
    :label "Values for function"
    :margin-choices `(("Abort" (signal-condition eh:*abort-object*)))
  )
  (dotimes `number-of-beeps
    (progn
      (beep)
      (sleep .5 "Wait for the beep")))
    (format *query-io* "-% -A, the beeps have sounded. You may proceed." your-name)
  ))
```

```
(call-cvv)
```



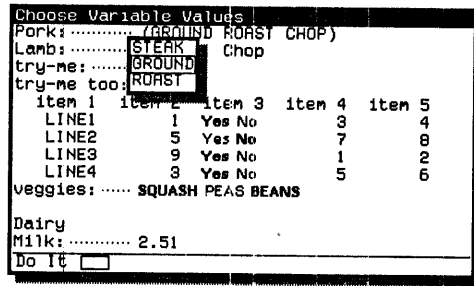
The following example shows how to construct a more complicated choose variable values menu.

```
; Builds a list of values that corresponds to the rows and columns of a table.
(defvar cuts-of-pork `(ground roast chop))
(defvar cuts-of-lamb `("Leg of" "Chop"))
(defvar cuts-of-beef-menu `(steak ground roast))
(defvar number-menu `(("one" . 1)
                     ("two" . 2)
                     ("three" . 3)))

(defvar milk-price 2.51)
(defvar my-variable nil "General use variable in this example")
(defvar veggie-set `(squash beans))
; Determines the default values for the table in the cvv window.
(defvar table `((line1 1 t 3 4)
                (line2 5 nil 7 8)
                (line3 9 t 1 2)
                (line4 3 t 5 6)))

(w:choose-variable-values
  '((cuts-of-pork "Pork" :princ)
    ; When chosen, beeps as well as changes the value.
    (cuts-of-lamb "Lamb" :edit :side-effect (beep) :string-list)
    ; Invokes a menu whose items are symbol names.
    (my-variable "try-me" :menu cuts-of-beef-menu)
    ; Invokes a menu whose items are strings.
    (my-variable "try-me too" :menu number-menu)
    (table (0 "item 1" :sexp) ; Produces a table of values.
           (1 "item 2" :number)
           (2 "item 3" :boolean)
           (3 "item 4" :number)
           ; The :typep variable type with its argument requires that an entered
           ; value must be an integer between 2 and 8 inclusive.
           (4 "item 5" :typep (integer 2 8)))
    ; Produces a set of values that the user can choose one or more from.
    (veggie-set "veggies" :set (squash peas beans))
    "" ; Produces a blank line.
    "Dairy" (milk-price "Milk"
              :documentation "Click left to input a new price."
              :number)))
```

The following figure shows the choose variable values window after the user clicks on the cuts-of-beef-menu variable, which is labeled try-me.



Defining Your Own Variable Type 14.4.3 Some variable types must be decoded to be used by the system. The Explorer system uses the `:decode-variable-type` method for this.

`:decode-variable-type` *kwd-and-args* Method of `w:basic-choose-variable-values`

The system uses the `:decode-variable-type` method on a choose-variable-values window when the system needs to understand an item. The *kwd-and-args* argument is a list whose car is the item's type keyword and whose remaining elements, if any, are the arguments to that keyword. `:decode-variable-type` returns the following six values:

1. *print-function* is a function of two arguments—object and stream—used to print the value. Specifying `prin1` is acceptable.
2. *read-function* is a function of one argument—the stream—used to read a new value. Specifying `read` is acceptable. If `nil` is specified, there is no *read-function*. Instead, new values are specified by pointing at one choice from a list. If the *read-function* is a symbol, it is called inside an input editor, and an over-robout condition automatically leaves the variable with its original value. If *read-function* is a list, its car is the function, and it is called directly rather than inside an input editor.
3. *choices* consist of a list of the choices to be printed, or `nil` if only the current value is to be printed. The choices are printed using the *print-function*, just as the current value is.
4. If there are choices and *print-translate* is non-`nil`, *print-translate* is an element of the choice list and must return the value to be printed using the *print-function*.
5. If there are choices and *value-translate* is non-`nil`, *value-translate* is an element of the choice list and must return the value to be stored in the variable.
6. *documentation* is a string or list to display in the mouse documentation window when the mouse is pointing at this item. This string or list should tell the user that clicking the mouse changes the value of this variable and should give any special information (for example, that the value must be a number). The structure of this list is discussed in paragraph 11.6, How Windows Handle the Mouse, under the description of the `:who-line-documentation-string` method of *windows*.

Alternatively, *documentation* can be a symbol that is the name of a function. The function is called with one argument, which is the current element of

choices or the current value of the variable if *choices* is `nil`. It is a good idea to have the function return a documentation string or list or `nil` if the default documentation is desired. Using *documentation* can be helpful when you want to document the meaning of a particular choice rather than simply saying that clicking the mouse on this choice selects the choice. The function returns a constant string or list rather than building one because it is called repeatedly as long as the mouse is pointing at an item of this type. The function is called by the status line updating in the scheduler, not in the user process.

The default method for `:decode-variable-type` looks for two properties on the keyword's property list:

- The value of the `w:choose-variable-values-keyword` property is a list of the six values discussed previously. Unnecessary values of `nil` can be omitted at the end.
- The value of the `w:choose-variable-values-keyword-function` property is a function that is called with one argument, *kwd-and-args*. The function must return the same six values as returned by the `:decode-variable-type` method (discussed previously).

You can add a new variable type to the standard set by putting one of the `w:choose-variable-values-keyword` or `w:choose-variable-values-keyword-function` properties on the keyword. You can also define your own flavor of choose-variable-values window and give it a `:decode-variable-type` method to make your window not use the standard variable types. Your method must take care of implementing the `:documentation` keyword, which can appear in an item where a variable type would normally appear.

The following example shows how to define `:boolean`:

```
(setf (get :boolean 'w:choose-variable-values-keyword)
      '(choose-variable-values-boolean-print nil (t nil)))
```

The type `:any` is defined as follows:

```
(setf (get :any 'w:choose-variable-values-keyword)
      '(print1 read-sexp))
```

Making Your Own Window

14.4.4 The `w:choose-variable-values` function may not be adequate if you wish to keep a window permanently exposed or if you wish to alter its behavior. You must then create a window yourself. The following flavors create choose-variable-values windows with different attributes.

w:basic-choose-variable-values Flavor
 Required flavor: `w:stream-mixin`

The basic flavor that makes a window implement the choose-variable-values facility. The `w:basic-choose-variable-values` flavor is not instantiable.

w:choose-variable-values-window Flavor
w:choose-variable-values-pane Flavor

A choose-variable-values window or pane of a constraint frame, respectively, with a set of features that includes borders, a label at the top, stream I/O, the ability to be scrolled if there are too many variables to fit in the window, and the ability to have choice boxes in the bottom margin. In addition, the `w:choose-variable-values-pane` flavor redefines the `:adjustable-size-p`

method to return `nil` on the assumption that the window's size has been specified by the frame and cannot be changed except by the frame.

w:temporary-choose-variable-values-window Flavor

A **w:choose-variable-values-window** that can pop up temporarily.

w:temporary-choose-variable-values-window Resource
&optional (*superior w:mouse-sheet*)

Contains windows of the **w:temporary-choose-variable-values-window** flavor. The **w:choose-variable-values** function uses this resource. *superior* specifies this window's superior.

The two main ways of using the previous flavors to create a window are to either specify all the parameters in the `init-plist`, or not specify the parameters but use the `:setup` method to set the parameters before using the window. In any case, you must specify the list of variable specifiers and the stack group in which variables are to be evaluated before you can use the window.

The following initialization options are available.

:variables *specifier-list* Initialization Option of **w:basic-choose-variable-values**
Settable.

Initializes the list of variable specifiers, telling the window which variables to display and how to read and print the values. The *specifier-list* argument is the list of variable specifiers, the same as in the first argument to the **w:choose-variable-values** function.

:function *fun* Initialization Option of **w:basic-choose-variable-values**
Gettable, settable. Default: `nil`, indicating no function

The window's associated function, which is the function called when the window changes the value of one of the variables it displays.

:stack-group *sg* Initialization Option of **w:basic-choose-variable-values**
Gettable, settable. Default: `nil`

Initializes the stack group containing binding for the variables whose values are to be chosen. The window needs these bindings so that it can get the values while running in another process (for instance the mouse process) to update the window display when the window is refreshed or scrolled. If you do not specify the stack group at this time, you must specify it with the `:setup` method before you can use the window.

:name-font *font* Initialization Option of **w:basic-choose-variable-values**

:value-font *font* Initialization Option of **w:basic-choose-variable-values**

:string-font *font* Initialization Option of **w:basic-choose-variable-values**

Default for all three: The system default font

Initializes the font used to display the names or values of variables, or strings (typically heading lines), respectively. *font* specifies the font to use.

:unselected-choice-font *font* Initialization Option of **w:basic-choose-variable-values**
Default: A small, distinctive font

Initializes the font used to display the choices for a value other than the current value. *font* specifies the font to use.

:selected-choice-font *font* Initialization Option of **w:basic-choose-variable-values**
 Default: The boldface version of the unselected choice font

Initializes the font used to display the current value of a variable when there is a finite set of choices. *font* specifies the font to use. This should be a boldface version of the preceding font.

:margin-choices *choice-list* Initialization Option of **w:choose-variable-values-window**
 Default: A single choice box, labeled "Do It"

Initializes a list of specifications for choice boxes to appear in the bottom margin. If this list contains an item with "Do It" as the string, the list is used. If the list does not contain such an item, the default Do It item is appended to the list. Each element can be a string, which is the label for the box that means the user is finished, or a cons of a label string and a form to be evaluated if that choice box is selected. Because this form is evaluated in the user process, it can do such things as alter the values of variables or exit using the **throw** function. Specifying **nil** for this initialization option suppresses the margin choices entirely.

If no dimensions are specified in the **init-plist**, the width and height are automatically chosen according to the other **init-plist** parameters. The number of variables to be displayed dictates the height. Specifying a height in the **init-plist** by using any of the standard dimension-specifying options overrides the automatic choice of height. If the specified height is not large enough to display all the choices, scrolling is enabled.

The following fragment of code demonstrates how to create a nonstandard margin choice for a choose variable values window. This example is taken from the **Frame** option of the **Split Screen** operation, which in turn was invoked from the **System** menu. This menu has only one margin choice: a **Cancel**, analogous to an **Abort** option. The **Do It** option that is typically one of the margin choices is, for this case, part of the menu that invokes this choose-variable-values window.

```
:margin-choices (list
  (list "Cancel the Frame" nil 'split-screen-punt-frame nil nil
    :documentation
      '(:documentation "Abort the frame. Mouse the DO IT item to create the frame.")))
```

Note that the format to specify margin choices is different from the format used to specify margin choices for the **choose-variable-values** function. The following function translates a list of margin choices from the **choose-variable-values** format to the **:margin-choices** format.

```
(defun process-margin-choices (margin-choices)
  (mapcar #'(lambda (x)
    (list (if (atom x) x (car x)) nil
      'tv:choose-variable-values-choice-box-handler
      nil nil (if (atom x) nil (cadr x))))
    margin-choices))
```

Choose-variable-values windows provide the following methods.

:setup *items label function margin-choices* Method of
w:basic-choose-variable-values

Changes the list of items (variables), the window label, the constraint function, and the choices in the bottom margin, and sets up the display. The **:setup** method also remembers the current stack group as the stack group

where the variables are bound. If the window is not exposed (more specifically, if the `:adjustable-size-p` method on the window returns `non-nil`), the `:setup` method reshapes the window to a size based on the specified items.

Arguments: *items* — The list of menu items to change.
label — The window label to change.
function — The constraint function to change.
margin-choices — The new list of margin choices. The format is the same as the `:margin-choices` initialization option.

:set-variables *item-list* &optional *dont-set-height* *width* *extra-width* *margin-choice-width* Method of **w:basic-choose-variable-values**

Sets the list of variable specifiers that controls the variables displayed in the window, then redisplay the window.

If you want the window to be reshaped, the following conditions must exist:

1. The window must be `:adjustable-p` (that is, `deexposed`).
2. The *dont-set-height* argument must be `nil`.
3. The *width* argument must be `non-nil`.

If these conditions are met and the window is resized, you should use the `:expose-near` method to reexpose the window to avoid problems with exposing the `choose-variable-values` window outside its superior.

Arguments: *item-list* — The list of variable specifiers.
dont-set-height — When true, inhibits the resizing of the window. Otherwise, allows resizing according to the number of lines needed by the new set of variables. The window can include up to 25 lines; if the window needs to display more lines, it switches to scrolling mode. The default value is `nil`.
width — The width of the window in characters. If *width* is `nil`, no resizing is done. If *width* is the symbol `t`, then the new width is calculated according to the needs of the new set of variables. The default value is the inverse of *dont-set-height*, thus allowing resizing in the default case.
extra-width — The amount of extra space each item must have so that the window is wide enough to display the longest item.
margin-choice-width — The required width, in pixels, to make the window wide enough so all the margin choices fit. If *margin-choice-width* is `nil` (the default), an appropriate width is calculated relative to the current set of margin choices, if any.

:adjustable-size-p Method of **w:basic-choose-variable-values**

If `:adjustable-size-p` returns `non-nil`, `:setup` reshapes the window. By default, this method returns `non-nil` when the window is `deexposed`.

:appropriate-width &optional *extra-space* Method of **w:basic-choose-variable-values**

Returns the inside width appropriate for this window to accommodate the current set of variables and their current values. You should use the `:appropriate-width` method after a `:setup` method and before an `:expose`

method, and then use the result to execute the `:set-inside-size` method. The returned width is less than the maximum that fits inside the superior.

A choose-variable-values window handles mouse clicks by putting blips (lists) in its input buffer. These blips are generated by the mouse process and are supposed to be read in the controlling process. There are two types of blips, both used for specific purposes, and your program must be able to take the appropriate actions when it reads them. The easy way for you to do this is to call the function `w:choose-variable-values-process-message`, which is provided precisely for this purpose.

`:io-buffer` *io-buffer* Initialization Option of `w:basic-choose-variable-values`
Default: `nil`

Sets the I/O buffer to be used for blips and for ordinary input from the window.

The following forms of lists are inserted as blips into the input buffer:

- `(:variable-choice window item value line-no start-x end-x)` — The user clicked on the value of a variable to change it. The controlling process should read keyboard input as necessary and set the variable.

window — The choose-variable-values window

item — An item in the window which was selected

value — The value of the variable

line-no — The line number of the item

start-x — The starting x-coordinate of the value

end-x — The ending x-coordinate of the value

- `(:choice-box window box)` — The user clicked on one of the choice boxes in the bottom margin. The controlling process may wish to deexpose the window if the box was the Do It box.

window — The choose-variable-values window

box — The name of the box

`w:choose-variable-values-process-message` *window blip* Function

Implements the proper response to the `:variable-choice` and `:choice-box` blips. `w:choose-variable-values-process-message` should be called in the process and stack group in which the variables being chosen are bound. *window* should be the choose-variable-values window, and *blip* should be the object read as input from the I/O buffer.

The `w:choose-variable-values-process-message` function returns `nil` except when *blip* indicates a click on a Do It choice box.

If *blip* is a `:variable-choice` blip (that is, the blip indicates that the user clicked on a variable), this function reads user input from the window, as necessary, and sets the variable.

If *blip* is a `:choice-box` blip (that is, the blip indicates that the user clicked on a finishing choice), the action depends on the *box* in it. If the sixth element of *box* is `nil`, which is normally the case for the Do It box, this function returns `t`. Otherwise, the sixth element of *box* is evaluated, but this function returns `nil`.

If *blip* is actually a character rather than a blip, the `:process-character` method is invoked. It is therefore reasonable to use this function with a loop like the following:

```
(do ()
  ((w:choose-variable-values-process-message
    c-v-v-window
    (progn
      (process-wait "Choose" #'listen c-v-v-window)
      (w:read-any c-v-v-window))))))
```

`:process-character` *ch*

Method of `w:basic-choose-variable-values`

Processes command characters entered into the choose-variable-values window. The user can see the available command characters by pressing the HELP key. *ch* is the character entered.

Mouse-Sensitive Items

14.5 The mouse-sensitive items facility is another kind of choice feature, although it is substantially different from the menu features. The mouse-sensitive items facility is a way of making selected rectangular areas of a window *mouse-sensitive*. When a user moves the mouse cursor over such a mouse-sensitive area, the rectangular outline of the area is displayed. A mouse-sensitive item associates with this rectangular area an arbitrary Lisp object. Thus, when a user clicks a mouse button on the highlighted mouse-sensitive area under the mouse, input is sent to the window informing the program which associated Lisp object was chosen. The mouse-sensitive items facility also allows a mouse-sensitive item to present a menu when clicked on, giving the user several options for the type of input that the mouse-sensitive item generates.

Mouse-sensitive items differ from other choice facilities, such as menu items, in that they are not permanently associated with a window. Mouse-sensitive items disappear when the window is cleared or when other output covers them up.

The Zmacs List Buffers command (CTRL-X CTRL-B) shows an example of the use of mouse-sensitive items. The line describing each buffer is surrounded by a mouse-sensitive area that is highlighted as you move the mouse cursor over it. If you click right on one of these areas, the system presents a menu that allows you to choose one of the possible operations on the buffer. These mouse-sensitive items generate input to the Zmacs editor that tells it which buffer and which operation was chosen.

How Mouse-Sensitive Items Work

14.5.1 Mouse-sensitive items are implemented by the `w:basic-mouse-sensitive-items` flavor. This flavor is not itself instantiable; instead, you mix it into the flavor of the window that will use mouse-sensitive-items, thus changing the window's mouse handling methods appropriately.

Programs receive input from mouse-sensitive items in the form of mouse blips that can be read from the window's input stream. The content of these mouse blips is defined and interpreted solely by the program. A program using mouse-sensitive item input must follow five basic steps.

1. Define and instantiate a window flavor, mixing in `w:basic-mouse-sensitive-items`.

2. Create a mouse-sensitive item. A mouse-sensitive item is defined by three attributes: a *type*, which is a keyword that controls the generation of mouse blips; an item object, which is an arbitrary Lisp object; and a rectangular area. A mouse-sensitive item is created by using methods such as `:item` or `:primitive-items`, or by using the `-M` directive of the `format` function.
3. Add an entry to the association list in the `:item-type-alist` instance variable of the window. The `:item-type-alist` entry for each type of mouse-sensitive item used defines how to generate a mouse blip for items of that type. This entry also defines any menu displayed by items of that type.
4. Draw output on the window the text or graphics displayed inside the item's area. This step can be combined with step 2 if you use the `-M` directive of the `format` function or the `:item` method.
5. Read the mouse blips generated by a mouse-sensitive items and perform the program operations that they specify.

For each type of mouse-sensitive item, the window's `:item-type-alist` instance variable must contain an entry of the form

```
(type left-button-alternative documentation menu-item-1
 menu-item-2 ...)
```

where:

type is a keyword for the mouse-sensitive item type.

left-button-alternative is the value that is returned in a mouse blip when the left mouse button is clicked over any item of this type.

documentation is used to define the string that appears in the mouse documentation window when the mouse is over any item of this type. *documentation* can be a string or a list of the form *(doc-function label-function)*, where *doc-function* is a function with a single argument (the mouse-sensitive item's item object) that returns the item's mouse documentation string, and *label-function* is a function with a single argument (the mouse-sensitive item's item object) that returns the string which is used as the label of the item's right-button menu.

menu-item-1, *menu-item-2*, and so on may be any menu item type described in Table 14-1, Type Value Keywords for Menus. At least one menu item is required. Menu items define a menu displayed when the righthand mouse button is clicked over any item of this type. In this case, the value of the menu item chosen by the user is included in the mouse blip generated.

The mouse blip generated by a mouse-sensitive item has the form `(:typeout-execute alternative item-object)`, where *item-object* is the Lisp object belonging to the item chosen and where *alternative* depends on the user's action, as follows:

- If the user clicked on the item with the left mouse button, *alternative* is the *left-button-alternative* from the `:item-type-alist` entry for the item's type.

- If the user clicked on the item with the righthand mouse button (thus displaying the item's menu), *alternative* is the value of chosen menu item from the `:item-type-alist` entry.

The `-M` format
Directive

14.5.2 The `-M` format directive is a simple way to create and display mouse-sensitive items containing text. This method uses the `format` function to send output to a window stream whose flavor includes the `w:basic-mouse-sensitive-items` flavor. The options for the `-M` directive are as follows:

Directive	Description
<code>-M, -nM</code>	Prints the argument with <code>princ</code> and creates a mouse-sensitive item. The item object is the argument itself. The item area is the bounding rectangle for the printed argument. The item type defaults to the first item type in the window's <code>:item-type-alist</code> . <code>-nM</code> can be used to set the item type to the <i>n</i> th item type in <code>:item-type-alist</code> .
<code>-VM</code>	Like <code>-M</code> , but used to specify the item type with a symbol, rather than by position. Two arguments are used. The first argument gives the item type; the second argument, which is printed, gives the item object.
<code> -:M, -:nM</code>	A recursive directive, where the argument is a list containing the item object, followed by a format string and format arguments. A mouse-sensitive item is created with an area that encloses all of the output specified by the format string and format arguments. The format string may include additional <code>-M</code> directives, thus creating nested items. The type of the item created by <code> -:M</code> is given by the first item type in the window's <code>:item-type-alist</code> . <code> -:nM</code> can be used to set the item type to the <i>n</i> th item type in <code>:item-type-alist</code> .
<code> -:VM</code>	Like <code> -:M</code> , but used to specify the item type with a symbol, rather than by position. Two arguments are used. The first argument gives the item type; the second argument is a list that contains the item object, followed by a format string and format arguments, as described for <code> -:M</code> .

NOTE: Items created by `-M` cannot extend over more than one line.

For example, the following code creates an instance of a mouse-sensitive window that includes three item types: a word item type, a phrase item type, and a sentence item type.

```
(defflavor mouse-sensitive-window ()
  (w:basic-mouse-sensitive-items w>window))

(setq my-window (make-instance 'mouse-sensitive-window
  :item-type-alist
  `((word-item left-click-word "a word type item.")
    (phrase-item left-click-phrase "a phrase type item.")
    (sentence-item left-click-sentence "a sentence type item."))))
```

Next, the window is selected, several mouse-sensitive items are output on it, and then the system waits for the user to select one of the items.

```
(w:window-call (my-window :deactivate)
  (format my-window "This ~M contains ~VM mouse-sensitive ~1M."
    "sentence"
    ^word-item "some"
    "words and phrases")
  (send my-window :any-tyi)
  )
```

When the user boxes words and phrases, the display appears similar to the following:

```
This sentence contains some mouse-sensitive words and phrases.
```

If the user clicks on sentence, the system returns (:typeout-execute left-click-word "sentence").

If the user clicks on words and phrases, the system returns (:typeout-execute left-click-phrase "words and phrases").

Similarly, the following code selects the same window again and creates and displays some mouse-sensitive items. The system then waits for the user to select one of the items.

```
(w:window-call (my-window :deactivate)
  (format my-window "-:2M" ^the-whole-sentence
    "This whole ~M, as well as ~VM ~1M, is mouse-sensitive."
    "sentence"
    word-item "some"
    "words and phrases"))
  (send my-window :any-tyi)
  )
```

When the user boxes the entire sentence, the display appears similar to the following:

```
This whole sentence, as well as some words and phrases, is mouse-sensitive.
```

If the user clicks on the entire sentence, the system returns (:typeout-execute left-click-sentence the-whole-sentence).

Using w:basic-mouse-sensitive-items 14.5.3

w:basic-mouse-sensitive-items

Flavor

Required flavors: w:essential-mouse, w:stream-mixin

Supplies the capability to output mouse-sensitive items on a window. The w:basic-mouse-sensitive-items flavor is called *basic* because it redefines the handling of the mouse by the window; mixing w:basic-mouse-sensitive-items into another flavor that defines a different type of mouse handling does not work. However, unlike many basic flavors, w:basic-mouse-sensitive-items does not do anything special with the displayed image of the window.

:item-type-alist *item-type* Initialization Option of **w:basic-mouse-sensitive-items**
Gettable, settable. Default: nil

Associated with each item type is a set of operations that are legal to perform on items of that type. One of these operations is selected as the default. The format of *item-type* is described in detail in paragraph 14.5.2, How Mouse-Sensitive Items Work.

The following code shows part of the item-type association list used in typeout windows of editor windows.

```
((zwei:directory zwei:directory-edit-1
 :mouse-L-1 "Run Dired on this directory." :mouse-R-1 "menu of View, Edit."
 ("View" :value zwei:view-directory
 :documentation "View this directory")
 ("Edit" :value zwei:directory-edit-1
 :documentation "Run Dired on this directory."))
(zwei:file zwei:find-defaulted-file
 :mouse-L-1 "Find file this file." :mouse-R-1 "menu of Load, Find, Compare."
 ("Load" :value zwei:load-defaulted-file
 :documentation "LOAD this file.")
 ("Find" :value zwei:find-defaulted-file
 :documentation "Find file this file.")
 ("Compare" :value zwei:srccom-file
 :documentation "Compare this file with the newest version."))
(zwei:flavor-name zwei:edit-definition-for-mouse
 :mouse-L-1 "Edit definition." :mouse-R-1 "menu of Describe, Edit."
 ("Describe" :value zwei:describe-flavor-internal
 :documentation "Describe this flavor.")
 ("Edit" :value zwei:edit-definition-for-mouse
 :documentation "Edit definition."))))
```

For the Load alternative on a file item in the editor, the blip might be as follows:

```
(:typeout-execute zwei:load-defaulted-file
 #Cfs:logical-pathname "SYS: SYS; QFCTNS LISP"␣)
```

w:add-typeout-item-type *alist type name function* Macro
&optional default-p documentation

Adds an entry to an association list kept in a special variable; this action declares information about a mouse-sensitive item type. This association list can then be put into the item-type association list of a mouse-sensitive window, for example, using the **:item-type-alist** *init-plist* option. Note that each possible alternative for a particular mouse-sensitive item type is defined with a separate **w:add-typeout-item-type** form. This convention allows each alternative to be defined at the place in the program where it is implemented rather than collecting all the alternatives into a separate table. The **w:add-typeout-item-type** macro also allows new alternatives to be added in a modular fashion.

alist, *type*, and *function* are not evaluated when the macro is expanded. *name*, *default-p*, and *documentation* are evaluated.

In the editor, *function* is interpreted (when a **:typeout-execute** blip is read) as a function to be called, and the **w:add-typeout-item-type** form is typically placed immediately before the function definition of *function*.

Arguments:

- alist* — The special variable containing the association list. You should set *alist* to `nil` (for example, by using `defvar`) before defining the first item type. Each program that uses mouse-sensitive items has its own association list of item types so that there is no conflict in the names of the types.
- type* — The keyword symbol for the type being defined.
- name* — The name of the operation performed on typeout items of this type.
- function* — The function performed if the user clicks on a mouse-sensitive item of this type. The function(s) that executes is listed in the mouse documentation window.
- default-p* — If non-`nil`, this method is the default performed when you click the left button on an item of this type.
- documentation* — Optional but highly recommended; a string or list that documents what *function* does. When the user points the mouse at an item of this type, the mouse documentation window at the bottom of the screen gives the documentation for the default function (reachable by the left button) and a list of the functions in the menu (reachable by the right button). If the user clicks right, calling for a menu, then the documentation for whichever function in the menu he or she points the mouse at is displayed.

The following methods are used to print items on a window.

`:item` *type item &rest format-args* Method of `w:basic-mouse-sensitive-items`

Prints a new item *item* of type *type*, either by calling `format` with *format-args*, or by calling `princ` on *item* if *format-args* is `nil`.

The mouse-sensitive area of the item is whatever space is used by printing it, as judged by the motion of the cursor.

The arguments *item* and *type* are not necessarily used in printing the item, but they are used in handling a click on the item. *type* is used to look up a function in the item-type association list, and *item* is placed directly into the `:typeout-execute` blip.

The following example shows an editor window where `standard-output` is a window that supports mouse-sensitive items, executes `princ` on the value of *pathname*, and makes an item of type `zwei:file` whose data is that *pathname*.

```
(send standard-output :item 'zwei:file pathname)
```

`:primitive-item` *type item left top right bottom* Method of `w:basic-mouse-sensitive-items`

`:primitive-item-outside` *type item left top right bottom* Method of `w:basic-mouse-sensitive-items`

Defines the position of a mouse-sensitive item without printing it. (Presumably, you print it yourself, either before or after calling this method.) These methods differ in the way the coordinate arguments are defined.

As in the `:item` method, *item* and *type* are not necessarily used in printing the item, but they are used in handling a click on the item. The *type* argument is used to look up a function in the item-type association list, and *item* is placed directly into the `:typeout-execute` blip.

In **:primitive-item**, the coordinates are relative to the inside top-left corner of the window (that is, they are cursor positions such as the **:read-cursorpos** method would return).

In **:primitive-item-outside**, the coordinates are relative to the outside corner of the window (like values of the **w:cursor-x** and **w:cursor-y** instance variables).

:item-list *type list* Method of **w:basic-mouse-sensitive-items**

Prints an item for each element of *list*. An element of *list* can be either a string or a list of the form (*name . item*). In the latter case, *name* (typically a string) is printed with **princ**, and *item* is used as the data for the item. If the element is an atom, that atom serves both as data and an argument to **princ**. All the items are of type *type*.

The elements of *list* are displayed using the geometry specified in initialization options or with the default layout used by the **w:menu** flavor.

:mouse-sensitive-item *x y* Method of **w:basic-mouse-sensitive-items**

Returns either a list describing the mouse-sensitive item found at cursor position *x*, *y* in the window, or **nil** if there is no item there.

The list has the following format:

(*type item left top right bottom*)

where:

type and *item* are not necessarily used in printing the item, but they are used in handling a click on the item. The *type* argument is used to look up a function in the item-type association list, and *item* is placed directly into the **:typeout-execute** blip.

left, *top*, *right*, and *bottom* are cursor position coordinates relative to the outside top-left corner of the window.

The **:mouse-click** method of **w:basic-mouse-sensitive-items** ignores all clicks that are not over a mouse-sensitive item except for **#\MOUSE-R-2**. If you want access to the clicks that occur over non-sensitive areas of the screen, you can define a method called **:non-sensitive-mouse-click**, which is called automatically by the **:mouse-click** method in the appropriate circumstance.

:non-sensitive-mouse-click *button x y* *User-supplied* method of
w:basic-mouse-sensitive-items

The arguments of **:non-sensitive-mouse-click** are the same as the arguments passed to **:mouse-click**, the current button pressed, and *x* and *y*, the current mouse coordinates. The **:mouse-click** method automatically calls the **:non-sensitive-mouse-click** method if a click occurs while the cursor is not over a

mouse-sensitive item. The return value of the `:non-sensitive-mouse-click` method is returned as `:mouse-click`'s return value.

NOTE: Because `:mouse-click` is an `:or` method combination, a true value inhibits subsequent handlers of `:mouse-click`, including *essential-mouse*. If you want the rest of the mouse processing to continue after you return from `:non-sensitive-mouse-click`, you return `nil` from `:non-sensitive-mouse-click`. If you do not want the rest of `:mouse-click` processing to continue, you return `true` from `:non-sensitive-mouse-click`.

Margin Choices

14.6 A window can be augmented with choice boxes in its bottom margin using the flavor `w:margin-choice-mixin`. These choice boxes give the user a few labeled mouse-sensitive points that are independent of anything else in the window.

Margin choices are not a complete, standalone choice facility and consequently do not have an easy-to-use functional interface.

For an example of a window with margin choices (as well as choice boxes in its interior), try the editor command META-X Kill or Save Buffers.

`w:margin-choice-mixin`

Flavor

Required flavor: `w:essential-window`

Puts choice boxes in the bottom margin, according to a list of choice-box descriptors that can be specified with the `:margin-choices` initialization option or the `:set-margin-choices` method. A choice box descriptor has the following form:

```
(name state function x1 x2 :documentation documentation-list)
```

where:

name is a string that labels the box.

state is `t` if the box has an X in it, or `nil` if it is empty.

function is a function called in a separate process if the user clicks on the choice box. This *function* receives three arguments: the choice-box descriptor for the choice box, the margin region that contains the choice boxes, and the y-position of the mouse relative to this window. You probably want to ignore the last two arguments. When *function* is called, `self` is bound to the window, so *function* can use `(declare (:self-flavor flavor))` to access the window's instance variables. The structure access functions `w:choice-box-name` and `w:choice-box-state` can be of use inside *function*. If *function* changes the state of the choice box, it needs to refresh the choice boxes by executing the following:

```
(funcall
 (w:margin-region-function region) :refresh region)
```

where `region` is the second argument of *function*, which is why that argument is passed.

x1 and *x2* are used internally to store the location of a choice box. *x1* is the left edge of the box; *x2* is the right edge of the box. Both *x1* and *x2* are pixel positions measured from the outside left edge of the window. Choices boxes are always spread out evenly in the available width of the window.

:documentation *documentation-list* is either a string or a list that provides mouse documentation. (The structure of this list is discussed in paragraph 11.6, How Windows Handle the Mouse, under the description of the **:who-line-documentation-string** method of *windows*.)

The **w:margin-choice-mixin** flavor contains the **w:margin-region-mixin** flavor as an included flavor; thus, **w:margin-region-mixin** appears in any combination immediately after **w:margin-choice-mixin** if it is not explicitly specified to appear somewhere else. The position of the **w:margin-region-mixin** flavor controls where the choice boxes appear in relation to the other margin items (borders, labels, and so forth).

To create a menu with margin choices, the **w:menu** flavor interfaces the margin choices to the menu.

:margin-choices *list* Initialization Option of **w:margin-choice-mixin**
Settable. Default: nil

Sets either a list of margin choices, or nil. If nil, no choice boxes appear and there is no space for them in the bottom margin; however, the window is still capable of using the **:set-margin-choices** method to create a line of choice boxes later. If non-nil, a line of choice boxes appears in the bottom margin of the window.

:margin-choice-default-font *font* Initialization Option of **w:margin-choice-mixin**
 Default: 0.

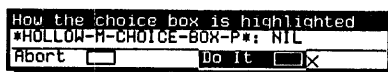
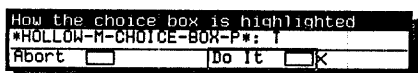
Sets an index of a font in the font map of the window to be the default font for the margin choice. The default, 0, specifies the first font in the window's font map.

w:*hollow-m-choice-box-p* Variable
 Default: t

Whether the box that surrounds the margin choice is a hollow box (that is, appears as standard video) or is a solid box (that is, appears as reverse video). The box is actually a blinker. A value of t produces a hollow blinker; a value of nil produces a solid rectangular blinker. This variable can be changed in the Profile utility.

For example, the following code produces a choose-variable-values window in which the user can change the value of **w:*hollow-m-choice-box-p*** and then move to the margin choice to see what the effect of the value is.

```
(w:choose-variable-values '(w:*hollow-m-choice-box-p*)
  :label "How the choice box is highlighted"
  :margin-choices `(("Abort" (signal-condition eh:*abort-object*))))
```



Using Frames

15.1 A *frame* is a window that is divided into subwindows, using the hierarchical structure of the window system. The subwindows, called *panes*, are the inferiors of the frame, and the frame is the superior of each pane.

Frames provide a convenient way to organize both information to be displayed and responses to be received from the user. Several frequently used system programs—such as the Inspector, the window-based debugger, the Glossary, and the Zmacs editor—use frames. In Zmacs, each new editor window is a pane of the Zmacs frame.

From the number of system programs that use frames, you can see some of the operations for which frames are designed. Using a frame as a user interface to an interactive subsystem is a convenient way to put many different items on the video display, each in its own place. You can split the frame into areas in which you can display text or graphics, areas where you can put menus or other mouse-sensitive input areas, and areas to interact with, where the keyboard input is echoed or otherwise acknowledged.

The Select menu and the TERM and SYSTEM keys should treat a frame and its panes as a unit. The mixins `w:inferiors-not-in-select-menu-mixin` and `w:alias-for-inferiors-mixin` in the frame's flavor are responsible for this treatment. As a result, selection of panes within the frame is done by making the chosen pane the selection substitute of the frame. The program managing the frame can maintain a selected pane within the frame in this way, while letting the user decide when to select the frame as a whole.

It is also common for all of the panes to use the same input buffer so that the program can always receive its input in the same fashion and collect keyboard and mouse input from all the panes.

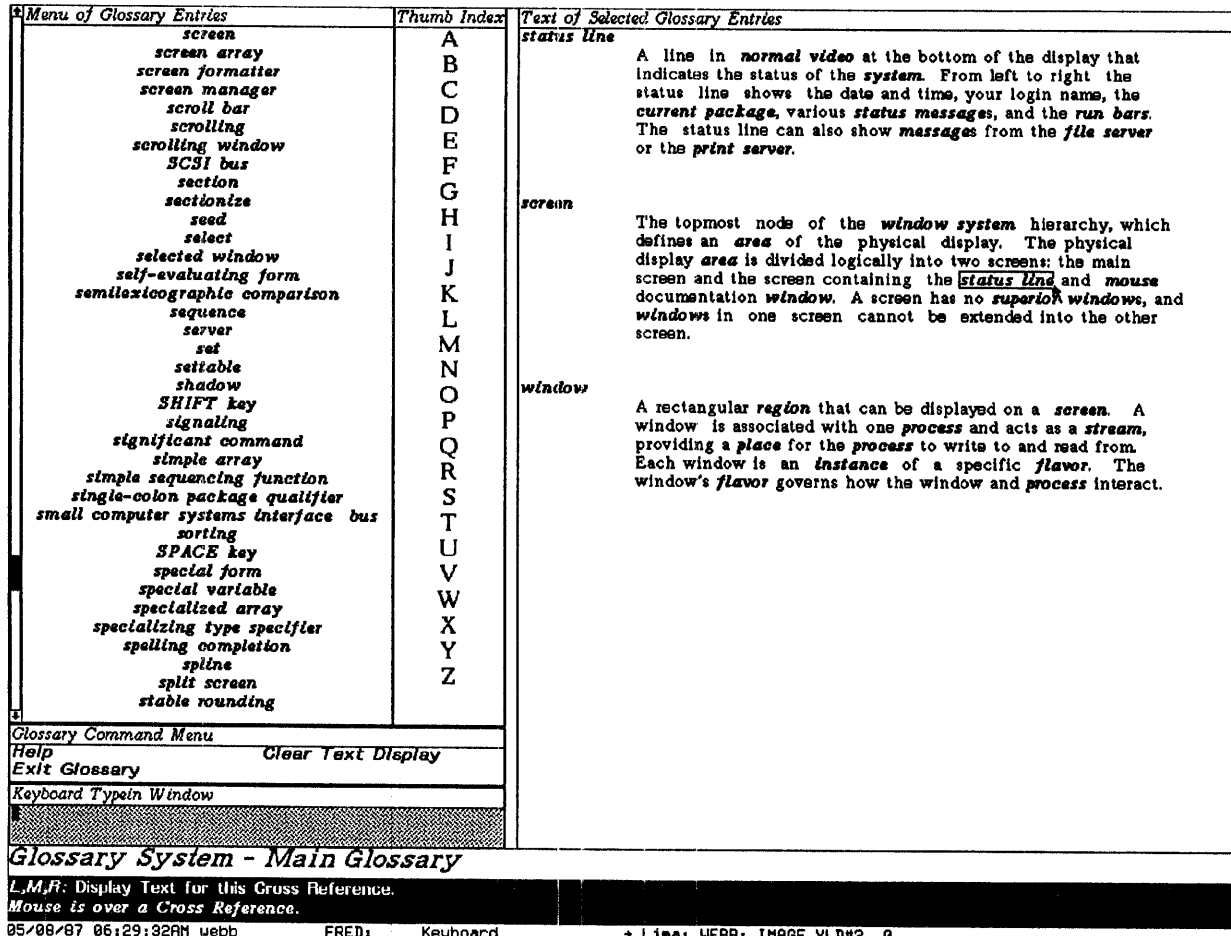
In addition to frames that serve a single application, you can also create frames that include two or more unrelated frames or windows. For example, suppose you create a frame with the System menu Split Screen option that includes both an Inspector and a Zmacs editor, each of equal size. The frame as a whole makes its panes share an input buffer and allows them to be individually represented both in the Select menu and for the TERM and SYSTEM keys. It also lets the panes—in this case, a half-size Inspector and a half-size editor, which are in turn frames—be selected independently rather than as substitutes for the frame because each window in the split screen frame is managed by its own process.

One kind of frame is the *constraint frame*, which adjusts the shapes of its panes automatically as its own shape is changed. Because constraint frames are a ready-to-use facility, they are described first along with WINIFRED, the window-based frame editor that helps you create code to generate constraint frames. There are more basic frame flavors that can be used to create frames that manage their panes' exposure and shapes in other ways. Zmacs, for example, manages its frames in a special way.

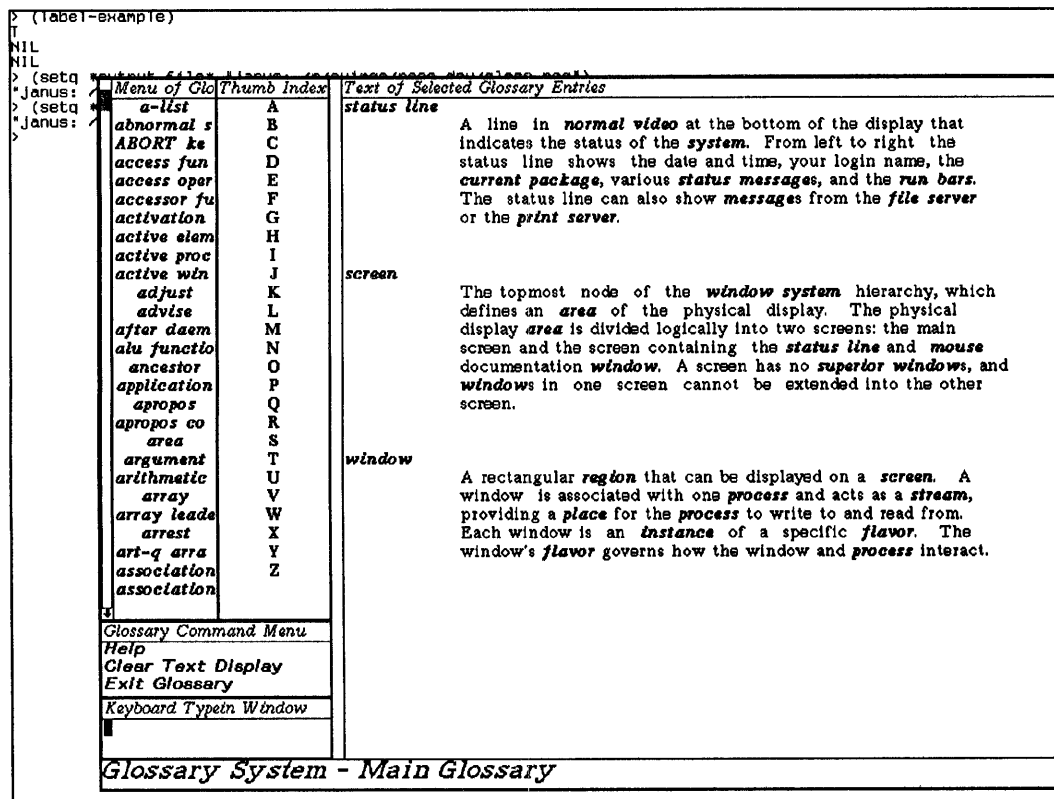
This section describes what frames are and explains how to use them on the Explorer system. Examples of frequently used or difficult features are given. These examples are written so that they can be executed directly, without additional coding.

Constraint Frames 15.2 Constraint frames are the most useful frame mechanism because of their ability to change shape upon request. One way to see this change is by using the Edit Screen feature of the System menu.

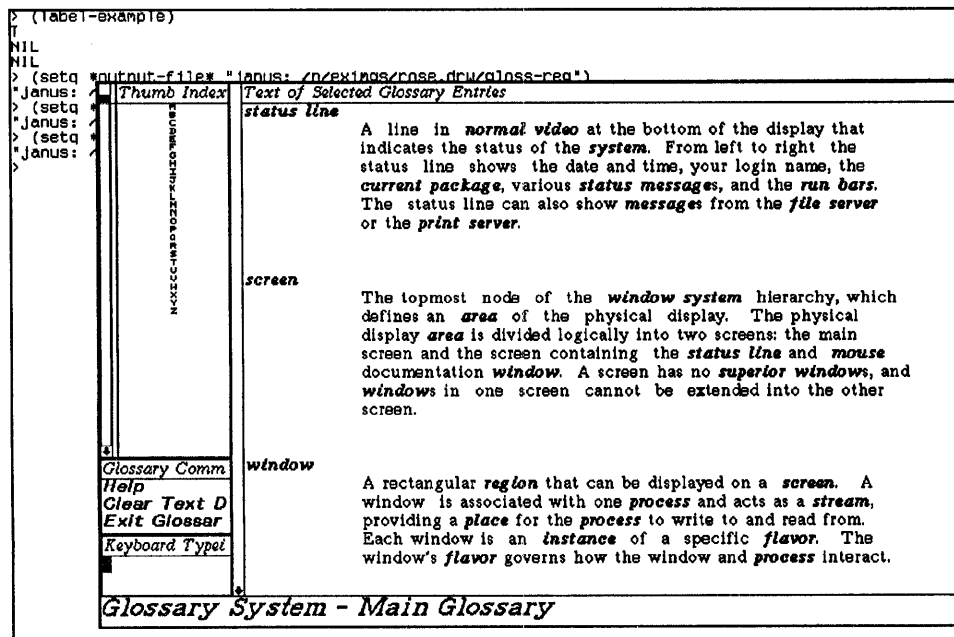
Consider the following three figures. The first figure shows the Glossary window, a typical constraint frame, as it normally appears when first invoked.



The next figure shows the Glossary when it has been made smaller using the Edit Screen option of the System menu. A Lisp Listener appears behind the Glossary. Note the differences in the heights of the individual panes. Not only are the individual panes smaller, the font in the *Thumb Index* pane changes. This font changes because the flavor that produces the Glossary window is defined to use a smaller font when the *Thumb Index* pane becomes smaller. In addition, the *Glossary Command Menu* pane has changed from two columns to one column.



The third shows a much smaller version of the Glossary window. In this figure note that one pane (the *Menu of Glossary Entries* pane) has disappeared altogether, and that the names of the commands in the *Glossary Command Menu* pane are truncated.



The constraint frame mechanism works so well because the positions and shapes are specified symbolically. For other windows, the positions and shapes are specified explicitly in terms of pixel coordinates and pixel dimensions. For constraint frames, you specify the positions and shapes of the panes as a set of constraints. The following three items are specified by the set of constraints:

- Which window flavors are used for each frame
- What layout should be used to organize the frames
- How the space for the window should be allocated among the various panes

Another feature of constraint frames is the ability to include several configurations. A *configuration* is a particular layout of a frame. Changing from one configuration to another allows you to present the frame in different ways, as described in paragraph 15.5.3, Multiple-Configuration Constraint Frame. For example, the Inspector has several configurations.

Constraint Frame Editor

15.3 The constraint frame editor (WINIFRED) is a menu-driven utility that generates code for constraint frames. Using WINIFRED enables a programmer to:

- Specify the position and size of panes in a constraint frame by using the mouse to drag one border of the pane
- Optionally specify the flavor type and name of a pane
- Write the constraint frame code generated by WINIFRED to a Zmacs buffer or to a file for later use
- Generate a sample constraint frame specified with WINIFRED
- Change the defaults for creating constraint frames

In addition, if the programmer edits the constraint frame code in the Zmacs buffer, he or she can then evaluate the buffer and use WINIFRED to test the edited code.

WINIFRED includes help screens that are invoked either by pressing the HELP key or by selecting the Help item in the WINIFRED command menu.

NOTE: The details of how the Explorer window system implements constraint frames are discussed later in this section. The following procedure describes how to use WINIFRED to create constraint frames with enough detail for you to follow the discussion.

Invoking WINIFRED 15.3.1 To invoke WINIFRED, do the following:

1. If you are unsure whether WINIFRED is currently available on your Explorer system, invoke the System menu.
 - If an item called Frame Editor is listed in the Programs column of the System menu, WINIFRED is already available on your Explorer system. Skip to step 2.
 - If an item called Frame Editor is *not* listed in the Programs column of the System menu, you must first make the frame editor system before you can use it.

Execute the form `(make-system 'winifred :noconfirm)`. The Explorer system requires several minutes to load the required files.

2. Invoke the System menu, if necessary, and select the Frame Editor item in the Programs column.

WINIFRED displays a command menu similar to the following:

```

CONSTRAINT FRAME EDITOR MENU
INITIALIZE OUTSIDE EDGES OF FRAME
DIVIDE FRAME INTO PANES
EDIT NAMES AND FLAVORS OF CONSTRAINT PANES
BUILD A FRAME IN A ZMACS BUFFER
WRITE EDGES, PANES AND CONSTRAINTS LISTS TO A FILE
WRITE EDGES, PANES AND CONSTRAINTS LISTS TO A ZMACS BUFFER
CHANGE SYSTEM DEFAULTS
REFRESH CURRENT FRAME
RESTART CURRENT FRAME
HELP
EXIT
  
```

Using WINIFRED 15.3.2 The process of creating a constraint frame using WINIFRED includes these general steps. Each of these steps is described in detail later.

1. Initialize the size and position of the constraint frame.
2. If desired, change the system defaults for the constraint lists and variable names.
3. Divide the frame into panes. (You specify pane size and position by subdividing a current pane and dragging a border of the new pane with the mouse.)
4. Supply the names and flavor types of the panes. If you do not specify either of these for a pane, WINIFRED specifies either a default or a dummy value for them.
5. Either write the code generated by WINIFRED to a file or write it to a Zmacs buffer and then save the buffer to a file.
6. Optionally, write the code to a buffer and then evaluate the buffer to generate a sample frame with this constraint. Using this mechanism, you can make changes in the code and evaluate the changed code to see how the frame is affected.

7. Exit WINIFRED by selecting the Exit item or by moving to another part of the Explorer system.

In practice, the division between steps 2, 3, and 4 and their order of execution is not rigid; you can create a pane, then change its name and flavor type, reassign the default values, create another pane or two, change their names and flavor types, and so on.

Initializing the Size and Position of the Constraint Frame

15.3.2.1 To initialize the size and position of the constraint frame, select the INITIALIZE CONSTRAINT FRAME SIZE AND POSITION item. This command invokes a pop-up menu that offers four choices:

- Whole WINIFRED Window — Creates a constraint frame that covers the entire main screen.
- Keyboard Set Edges — Pops up a choose-variables-value window that prompts you for the pixel location of the four edges of the constraint frame. WINIFRED creates a frame of this size and at this position. Thus, to create a frame that is 900 x 500 pixels wide with the frame origin offset from the origin of the screen by 50,50, you would specify Top as 50, Left as 50, Bottom as 550, and Right as 950.
- Centered Height and Width — Pops up a choose-variables-value window that prompts you for the height and width of the constraint frame. WINIFRED creates a frame of this size centered within the main screen. Thus, to create a frame that is 900 x 500 pixels wide, you would specify width as 900 and Height as 500.
- Exit — Returns to the main WINIFRED menu.

When you initialize the size of the frame, WINIFRED displays the borders around the frame as well as the name of the pane used in the code and the flavor type of the pane. The pane name is a dummy name followed by a unique number; the pane flavor is a system value, which you can change.

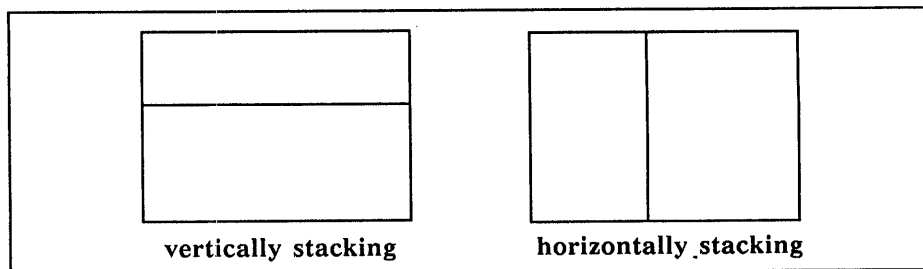
Changing the Default Values

15.3.2.2 Invoking the CHANGE SYSTEM DEFAULTS item pops up a choose-variables-value window that prompts you for various default values:

- The name of the default flavor type for each pane. At any time while you are creating panes within a constraint frame, you can change the default value for the flavor type of the panes you create after you change the default. This default value does not change for the panes you have already created.
- The name of the entire configuration. (This is *not* used for the label of the constraint frame.)
- The variable that contains the panes list.
- The variable that contains the constraints list.
- The variable that contains the constraint frame edges.

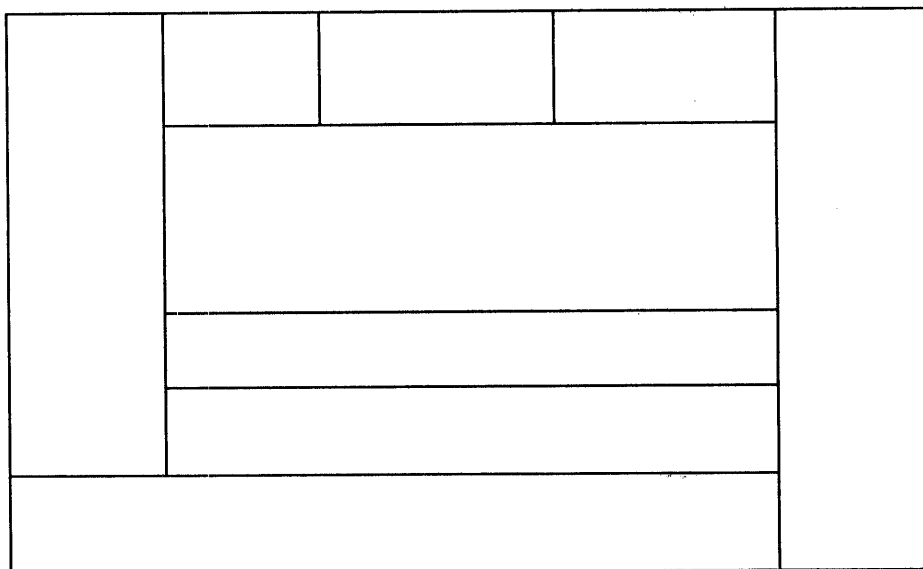
Specifying Pane Size and Position

15.3.2.3 In WINIFRED, you specify the size and position of panes by dividing one of the current panes or the frame itself into smaller and smaller panes. You can divide a pane into vertically stacking panes by clicking left, or into horizontally stacking panes by clicking right. Clicking middle returns to the main WINIFRED menu.

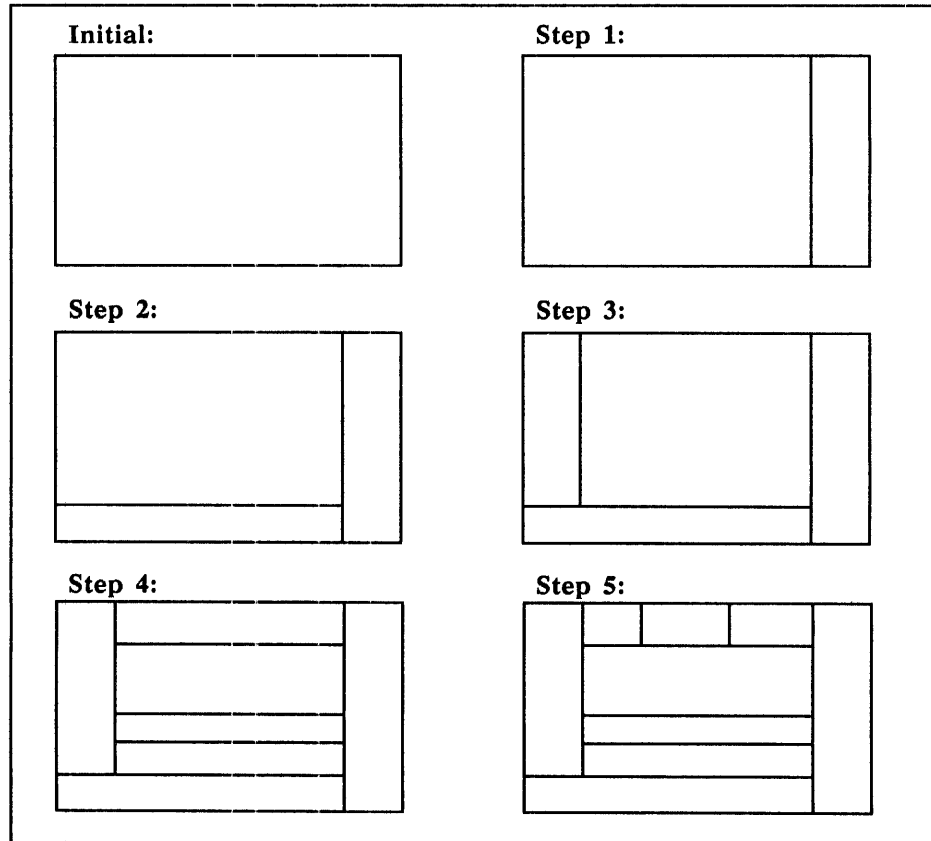


After you click to divide the pane, WINIFRED displays the border of the pane with an arrow pointing to it. You can drag the border within its parent frame to determine the size and position of the pane. When you have the border in its desired position, click either left or right to select that position; click middle to abort that pane division. (Once you click to set the position for that pane, that pane exists. If you accidentally click and create a pane that you do not want, you must restart the current frame.) When you subdivide an existing pane, that pane disappears and the two new panes appear, as indicated by the new borders, pane names, and pane flavors.

For example, consider the following configuration.



To create this configuration, you must create the panes in a specific order, dividing each larger pane into smaller panes. For panes with the same width or the same height (as in steps 4 and 5 shown in the following figure), the order in which you create the panes does not matter. In more detail, the WINIFRED window appears similar to the following at the end of each step of dividing the frame into panes:



If desired, you could use an alternate order for creating the panes shown in steps 4 and 5. For example, you could create the small upper pane and then divide that pane into three. Finally, you would divide the center pane into its smaller lower panes.

When you want to return to other parts of WINIFRED, click middle to invoke the main WINIFRED menu.

NOTE: The division between subdividing panes and supplying their information is arbitrary. You can move freely between these two procedures. However, first creating all the panes and then specifying their names and flavors is slightly quicker than mingling the two procedures.

*Specifying Pane
Flavor Type
and Name*

15.3.2.4 After you have created the panes, you are now ready to specify the names and flavor of those panes. If you do not specify either of these values for a pane, WINIFRED supplies either a default or a dummy value for them. To specify other values, follow these steps:

1. Select the EDIT NAMES AND FLAVORS OF CONSTRAINT PANES item.
2. Choose which pane to describe by pointing at it and clicking either left or right with the mouse.

WINIFRED invokes a choose-variable-values window similar to the following that prompts you for the name and flavor of the pane.

```
Centered WINIFRED Frame
Pane Name : ..... PANE12
Pane Flavor Menu: w:window
Pane Flavor : ..... W:WINDOW
Abort  Do It 
```

3. Enter the information.
 - The internal name of the pane must be a unique symbol. This pane *name* is used within the code to identify the pane; this name is separate from the *pane label* that appears when the constraint frame is visible. When you click on the value next to the prompt, you can enter the new value.
 - The flavor type of the pane must be the name of a valid flavor. Either the Pane Flavor Menu field or the Pane Flavor field can be used to change the flavor type of the pane; both affect the same variable, and the last value entered in either field is the value used. Selecting the Flavor Type Menu pops up a menu with several commonly used flavors. If the flavor you want is not on this menu, you can enter the name of a flavor by selecting the Flavor Name item and typing the name.

Flavors that are not system-defined do not need to be defined before you specify them in this menu. However, if a flavor is not defined, you cannot successfully create a sample constraint frame (described in paragraph 15.3.2.6).
4. Select one of the margin choices to exit from the choose-variable-values window. After you select the Do It item or you press end, WINIFRED updates the pane name and flavor displayed in the frame.
5. Repeat steps 2 through 5 for each pane whose information you want to change.

If you are using similar flavors or names for several panes, you may be able to use the input history and other input editor commands to save keystrokes. Pressing CTRL-C inserts the last term you typed into the input buffer; pressing META-C inserts the next-to-last term; pressing META-C again inserts the term input before that, and so on.

6. Return to the main WINIFRED menu by clicking middle.

Writing the Code to a Buffer or File

15.3.2.5 When you have generated your constraint frame, write the code either to a Zmacs buffer or to a file to save it for later use. If you choose either of these options, WINIFRED prompts you for the name of a buffer or file, and then writes the code to that buffer or file.

For example, the following code was generated for the configuration described in paragraph 12.3.2.3.

```
;;; -*- Mode: Common-Lisp; Package:FRED; Base: 10; Fonts:hl12,HL12B,HL12BI -*-

(setq *WINIFRED-CONSTRAINT-LIST*
'((*WINIFRED-DEFAULT-CONSTRAINT* (USER::DUMMY1)
  ((USER::DUMMY1 :HORIZONTAL (:EVEN) (USER::DUMMY2 RIGHT-MENU)
    ((USER::DUMMY2 :VERTICAL (0.79941005) (USER::DUMMY3 BOTTOM)
      ((USER::DUMMY3 :HORIZONTAL (0.8051771) (LEFT-MENU USER::DUMMY4) ((LEFT-MENU 0.16482165))
        ((USER::DUMMY4 :HORIZONTAL (:EVEN) (USER::DUMMY5 MIDDLE HISTORY2 HISTORY1)
          ((USER::DUMMY5 :HORIZONTAL (0.24873096) (REG1 REG2 REG3)
            ((REG1 0.2827688) (REG2 0.36524302)) ((REG3 :EVEN))))
            (MIDDLE 0.43824026) (HISTORY2 0.1573604))
          ((HISTORY1 :EVEN))))))
        ((BOTTOM :EVEN))))
      ((RIGHT-MENU :EVEN))))))
)

(setq *WINIFRED-PANE-LIST*
'((RIGHT-MENU CMD-MENU) (BOTTOM W:WINDOW) (HISTORY1 W:WINDOW) (HISTORY2 W:WINDOW)
  (MIDDLE W::LISP-LISTENER) (REG3 W:WINDOW) (REG2 W:WINDOW) (REG1 W:WINDOW)
  (LEFT-MENU CMD-MENU))
)

(setq *WINIFRED-EDGE-LIST*
'(1 1 1018 735)
)
```

Creating a Sample Frame

15.3.2.6 Alternately, you can first write the code to a buffer (by selecting the Build a Frame in a Zmacs Buffer entry) and then evaluate the buffer to generate a sample frame. If you specified a user-defined flavor as one of the pane flavors (such as CMD-MENU in the example), that flavor must be defined in memory before you build the test frame. For example, if you evaluate the following buffer before you define CMD-MENU, the system returns an error.

```
;;; -*- Mode: Common-Lisp; Package:FRED; Base: 10; Fonts:hl12,HL12B,HL12BI,cptfont -*-
```

First, evaluate the following Setq forms. Then evaluate the function BUILD-TEST-FRAME, which will ask you to supply the flavor of the frame, make an instance called *winifred-test-frame*, and then Expose your frame. You can make changes to the lists and see the results by re-executing BUILD-TEST-FRAME.

Press <Term>-f to return to WINIFRED.
Press <System>-e to go to ZMACS.

```
(setq *WINIFRED-CONSTRAINT-LIST*
'((*WINIFRED-DEFAULT-CONSTRAINT* (USER::DUMMY6)
  ((USER::DUMMY6 :HORIZONTAL (:EVEN) (USER::DUMMY7 RIGHT-MENU)
    ((USER::DUMMY7 :VERTICAL (0.79941005) (USER::DUMMY8 BOTTOM)
      ((USER::DUMMY8 :HORIZONTAL (0.8051771) (LEFT-MENU USER::DUMMY9) ((LEFT-MENU 0.16482165))
        ((USER::DUMMY9 :VERTICAL (:EVEN) (USER::DUMMY10 MIDDLE HISTORY2 HISTORY1)
          ((USER::DUMMY10 :HORIZONTAL (0.24873096) (REG1 REG2 REG3)
            ((REG1 0.2827688) (REG2 0.36524302)) ((REG3 :EVEN))))
            (MIDDLE 0.43824026) (HISTORY2 0.1573604))
          ((HISTORY1 :EVEN))))))
        ((BOTTOM :EVEN))))
)



---



```

```

) ((RIGHT-MENU :EVEN))))))
)

(setq *WINIFRED-PANE-LIST*
' ((RIGHT-MENU CMD-MENU) (BOTTOM W:WINDOW) (HISTORY1 W:WINDOW) (HISTORY2 W:WINDOW)
(MIDDLE W::LISP-LISTENER) (REG3 W:WINDOW) (REG2 W:WINDOW) (REG1 W:WINDOW)
(LEFT-MENU CMD-MENU))
)

(setq *WINIFRED-EDGE-LIST*
'(1 1 1018 735)
)

(BUILD-TEST-FRAME *WINIFRED-CONSTRAINT-LIST* *WINIFRED-PANE-LIST* *WINIFRED-EDGE-LIST*)

```

Exiting WINIFRED 15.3.2.7 You can exit WINIFRED at any time by selecting the Exit item or by selecting another system on the Explorer by pressing a SYSTEM keystroke sequence or by selecting a program from the System menu.

Editing the Generated Code 15.3.3 As you can see from the examples of code, the code would be more readable with certain minor changes, including the following:

- Delete the USER:: package specifiers.
- Round the numeric specifiers to one or two digits.
- Stack the pane list vertically. (The Zmacs editor command, META-X Stack List Vertically, was used to stack the list in the example.)

You then need to add initialization options to the panes list, such as labels, item lists, and so on. In addition, you should add the code that defines any required flavors. The following code includes a definition of the user-defined flavor `cmd-menu`, labels and blinkers for the history and register panes (some of which are `nil` in this example), and menu item lists for the menus. Note the use of the backquote feature with the menu item lists.

```

;; -*- Mode: Common-Lisp; Package:FRED; Base: 10; Fonts:hl12,HL12B,HL12BI,cptfont -*-

(defflavor cmd-menu () (w:menu)
 (:default-init-plist
 :command-menu t
 :dynamic t
 :permanent t))

(defvar rt-menu-list '(("foo" :value 'foo :documentation "This is a foo.")
 ("bar" :value 'bar :documentation "That is a bar.")
 ("frobboz" :value 'frobboz :documentation "Another frobboz.")))

(defvar lt-menu-list '(("name" :value 'name :documentation "The name of this thing.")
 ("command" :value 'command :documentation "The command that drives it.")
 ("etc" :value 'and-so-on :documentation "Lots of others too numerous to specify.")))

(setq *WINIFRED-CONSTRAINT-LIST*
' ((*WINIFRED-DEFAULT-CONSTRAINT* (DUMMY6)
 ((DUMMY6 :HORIZONTAL (:EVEN) (DUMMY7 RIGHT-MENU)
 ((DUMMY7 :VERTICAL (0.8) (DUMMY8 BOTTOM)
 ((DUMMY8 :HORIZONTAL (0.8) (LEFT-MENU DUMMY9) ((LEFT-MENU 0.15)
 ((DUMMY9 :VERTICAL (:EVEN) (DUMMY10 MIDDLE HISTORY2 HISTORY1)
 ((DUMMY10 :HORIZONTAL (0.25) (REG1 REG2 REG3)
 ((REG1 0.33) (REG2 0.33) ((REG3 :EVEN)))
 (MIDDLE 0.4) (HISTORY2 0.15))
 ((HISTORY1 :EVEN))))))
 ((BOTTOM :EVEN))))))

```

Frames

```

    ((RIGHT-MENU :EVEN))))))
  )

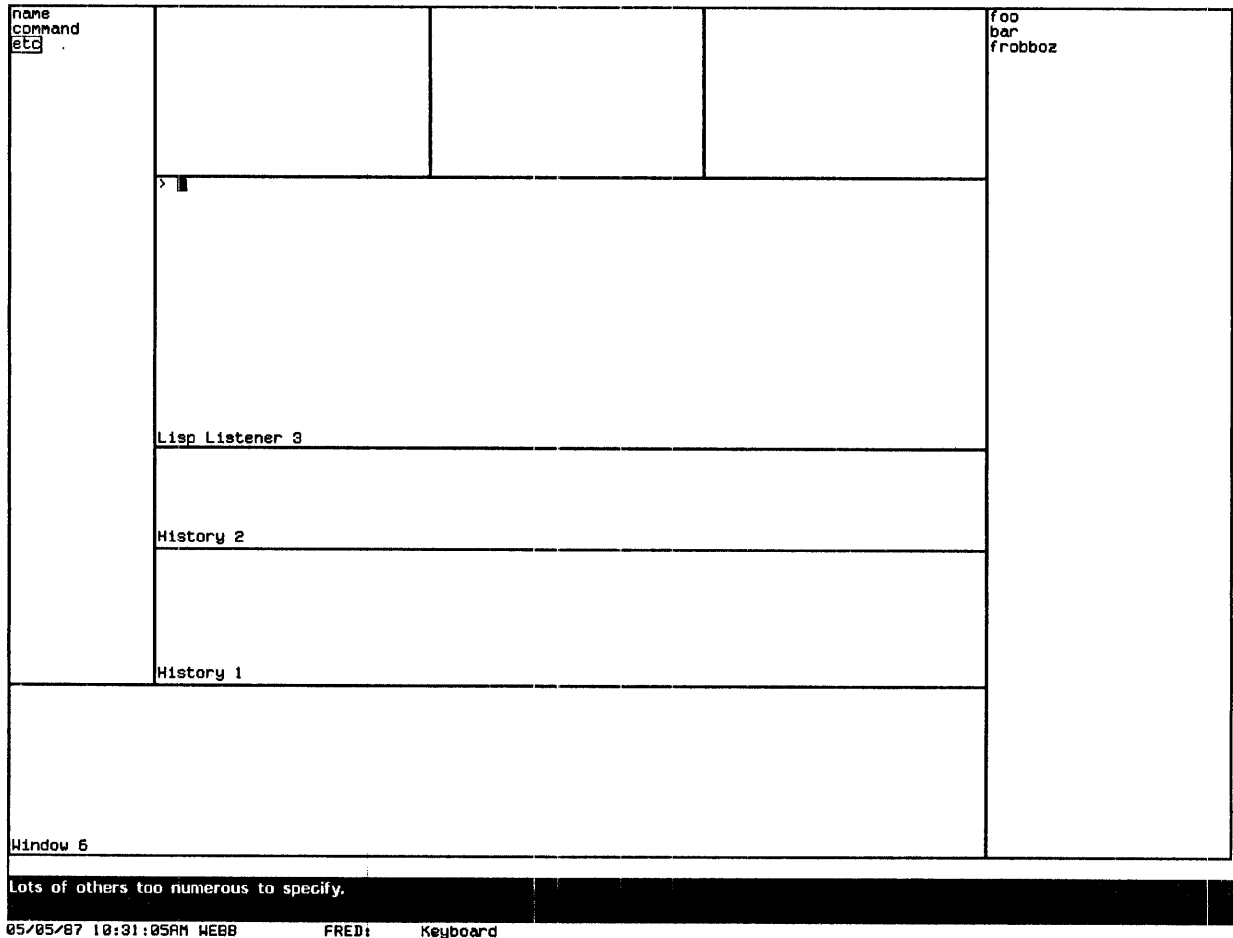
(setq *WINIFRED-PANE-LIST*
'((RIGHT-MENU CMD-MENU :item-list ,rt-menu-list)
  (BOTTOM W:WINDOW :blinker-p nil)
  (HISTORY1 W:WINDOW :blinker-p nil :label "History 1")
  (HISTORY2 W:WINDOW :blinker-p nil :label "History 2")
  (MIDDLE W::LISP-LISTENER)
  (REG3 W:WINDOW :blinker-p nil :label nil)
  (REG2 W:WINDOW :blinker-p nil :label nil)
  (REG1 W:WINDOW :blinker-p nil :label nil)
  (LEFT-MENU CMD-MENU :item-list ,lt-menu-list))
)

(setq *WINIFRED-EDGE-LIST*
'(1 1 1018 735)
)

(BUILD-TEST-FRAME *WINIFRED-CONSTRAINT-LIST* *WINIFRED-PANE-LIST* *WINIFRED-EDGE-LIST*)

```

If you evaluate this code and build the test frame again, the frame appears similar to the following:



Using the Generated Code in Your Application

15.3.4 After you are satisfied with the frame as it appears, you are ready to put the code into your own application. Follow these steps:

1. Copy the `setq` that define `*WINIFRED-CONSTRAINT-LIST*`, `*WINIFRED-PANE-LIST*`, and `*WINIFRED-EDGE-LIST*` to another buffer and change their names to something unique for your application. In this example, they are renamed to `*MY-CONSTRAINT-LIST*`, `*MY-PANE-LIST*`, and `*MY-EDGE-LIST*`, respectively.
2. Replace `*WINIFRED-DEFAULT-CONSTRAINT*` with `t`.
3. Add the `make-instance` to create the frame along with any supporting code. For example, the following code creates a constraint frame that has a label for the frame as a whole:

```
(defflavor labelled-frame ()
  (w:label-mixin
   w:bordered-constraint-frame))

(defun make-frame ()
  (setq my-frame
    (make-instance 'labelled-frame
      :save-bits t
      :label '(:string "my constraint frame"
                :font fonts:tr18)
      :edges MY-EDGE-LIST
      :panes MY-PANE-LIST
      :constraints MY-CONSTRAINT-LIST
      :expose-p t)
    )
  (send (car (send my-frame :inferiors)) :select))
```

You can then call the function from your application.

Constraint Frame Flavors

15.4 Several flavors can be used to construct a constraint frame. The simplest flavor is `w:constraint-frame`.

`w:basic-frame`

Flavor

The basic flavor that provides frame handling. All frame flavors are built from `w:basic-frame`. The `w:frame-forwarding-mixin` flavor, when mixed with the `w:basic-frame` flavor, provides a nonconstraint frame to which code only needs to be added to determine when to expose the panes and what their sizes should be.

`w:basic-frame` is nearly the same as the `w:minimum-window` flavor. The `w:basic-frame` flavor does not contain all of the mixins that are in the `w>window` flavor. In particular, `w:basic-frame` does not provide for borders or a label because the frame is a collection of other windows, each of which has its own borders or labels. Furthermore, `w:basic-frame` cannot be the selected window. A frame cannot be selected for the same reason—one of the panes within the frame must ultimately be selected.

w:recursion Instance Variable of **w:basic-frame**

Distinguishes between methods sent by the frame's code to its panes, such as **:expose**, and methods sent by other programs. When an outside program sends an **:expose**, **:deexpose**, or **:bury** message to one of the panes, methods on the frame (such as **:inferior-expose**) simply expose, deexpose, or bury the frame itself and do not allow the method on the pane to proceed. When the frame's code itself exposes a pane, it does so with **w:recursion** temporarily set to a non-nil value so that when the **:inferior-expose** instance variable is invoked, it returns **t** and lets the pane expose.

w:constraint-frame Flavor

The basic kind of constraint frame. The rest of this section describes its behavior in detail. This flavor, like **w:basic-frame**, does not provide borders or a label, nor does it allow the frame to be selected.

w:bordered-constraint-frame Flavor

Creates a window exactly like a window created by the **w:constraint-frame** flavor, except that borders are placed around the outside edges of the outermost panes. By default, the **:border-margin-width** of the constraint frame is 0, which means that the border of the constraint frame window is adjacent to the borders of the panes. If you set **:border-margin-width** to a value of 1, a one-pixel gap is left inside the border of the constraint frame.

Bordered constraint frames are used most often. Usually, for appearances, the frame as well as each of the panes has a border. (Panels require borders only as a visual reminder of their sizes and locations.) When two adjacent panes have borders, their adjacent borders appear as a double-width line. If the constraint frame does not have a border, then the outermost frames have outer borders that appear as single-width lines. Placing a border on the constraint frame makes this outer border appear to be the same width as the interior borders separating the frames.

One potential problem with constraint frames arises when you are typing on the keyboard. If you are typing ahead, then you need to ensure that the proper pane is selected so that the input goes to the proper place. If this is not done, then the type-ahead may go to another pane in the constraint frame, which may not know how to deal with the input, or even worse, may perform undesirable actions. A convenient way to solve this problem is to make all of the panes of a constraint frame use the same input buffer.

You can restrict all of the panes of a constraint frame to the same input buffer by using one of the following flavors.

w:constraint-frame-with-shared-io-buffer Flavor

Exactly like the **w:constraint-frame** flavor, except that all of the panes of the frame share the same input buffer.

w:bordered-constraint-frame-with-shared-io-buffer Flavor

Exactly like the **w:bordered-constraint-frame** flavor, except that all of the panes of the frame share the same input buffer.

`:io-buffer io-buffer`Initialization Option of
`w:constraint-frame-with-shared-io-buffer``:io-buffer io-buffer`Initialization Option of
`w:bordered-constraint-frame-with-shared-io-buffer`

Used to specify an I/O buffer for a window created using the specified flavor. If this initialization option is present, *io-buffer* is used as the input buffer for the frame and its panes. Otherwise, a default input buffer is created.

Examples of Specifications of Panes and Constraints

15.5 The complete description of how to use constraint frames, including the language used to specify constraints, is quite complex. Examples are given here for the more common cases involving constraint frames. All of these examples are given in their complete form, allowing you to test them on your Explorer system.

Simple Constraint Frame

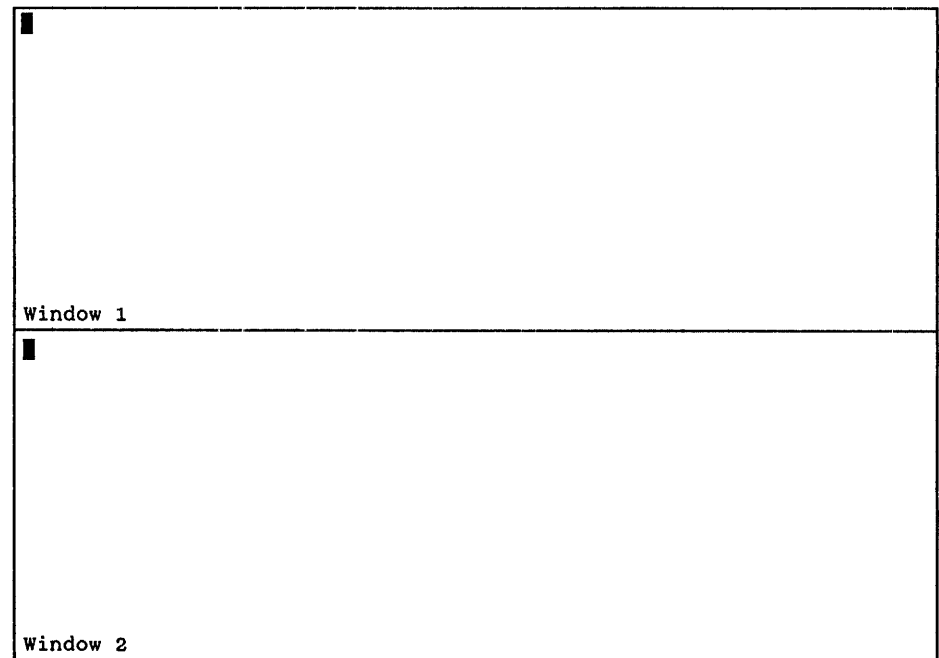
15.5.1 The following code creates a constraint frame with two panes, one in the top half of the available screen, the other in the bottom half; each of the panes takes up half of the frame.

```
(defun simple (&aux simple-window-instance)
  "Create a simple constraint frame."
  (setq simple-window-instance
    (make-instance 'w:constraint-frame
      :panes
        `((top-pane w:window)
          (bottom-pane w:window))
      :constraints
        `((main . ((top-pane bottom-pane)
                  ((top-pane 0.5)
                   ((bottom-pane :even))))))
    )
    (send simple-window-instance :expose)
  )
```

The `simple` function creates the constraint frame and exposes it.

- ① The constraint frame is made up of two panes named `top-pane` and `bottom-pane`, which are created from the `w:window` flavor. This arrangement is specified in the `:panes` description.
- ② The `:constraints` description states that the pane `top-pane` is placed on top and that the pane `bottom-pane` is placed on the bottom. This arrangement is specified by the ordering of the pane names in the list `(top-pane bottom-pane)`.
- ③ The remainder of the constraint description states that the pane `top-pane` uses 50 percent of the window and that the pane `bottom-pane` uses the remainder—the other 50 percent.

The frame appears as follows:



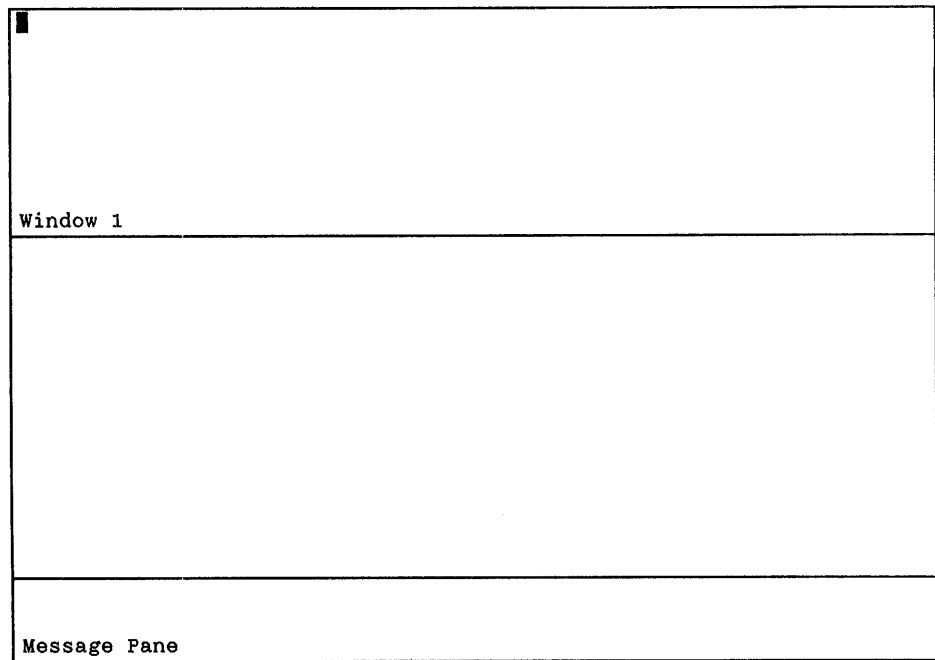
Graphics Constraint Frame

15.5.2 This paragraph discusses how to create a constraint frame consisting of three panes: an interaction pane, a graphics pane, and a message pane. The following code creates the constraint frame. The three panes are named `interaction-pane`, `graphics-pane`, and `message-pane`. All three panes are created using the `w:window` flavor. `graphics-pane` is created with no label and no blinker. `message-pane` is created with a label and no blinker. These characteristics are specified in the `:panes` descriptions.

The constraint description specifies the layout of the panes and their use of the window space. The layout of the panes is `interaction-pane`, `graphics-pane`, and `message-pane`, from top to bottom. `message-pane` requires 4 lines of vertical space, `graphics-pane` requires 400 pixels of vertical space, and `interaction-pane` uses the remainder of the available vertical space.

```
(defun graphics (&aux gr-window-instance)
  (setq gr-window-instance
        (make-instance 'w:bordered-constraint-frame
                       :panes
                       `((graphics-pane w:window
                          :label nil
                          :blinker-p nil)
                         (message-pane w:window
                          :label "Message Pane"
                          :blinker-p nil)
                         (interaction-pane w:window))
                       :constraints
                       `((main . ((interaction-pane
                                   graphics-pane
                                   message-pane)
                                  ((message-pane 4 :lines))
                                  ((graphics-pane 400))
                                  ((interaction-pane :even)))))))
  (send gr-window-instance :expose)
)
```

The `graphics` function creates and exposes a frame similar to the following:



Multiple- Configuration Constraint Frame

15.5.3 The following example, which is more complex than the previous examples, shows how to create a constraint frame that contains multiple configurations. The first configuration has three small windows arranged horizontally across the top of the constraint frame and one large window beneath them. In the second configuration, the same large window is located at the top, and the bottom part of the constraint frame is split between another window and a menu window. The following code creates the constraint frame and allows the user to type input to the `main-pane`.

Frames

```
(defun multi-config (&aux
  char          ; Single input character
  my-io-buffer  ; An I/O buffer
  main-pane-window ; Window for main-pane pane
  menu-window   ; Window for menu pane
  old-window    ; Previously selected window
  ; Create the constraint frame if it does not already exist.
  (unless (boundp 'window-instance)
    (setq window-instance (make-instance 'w:bordered-constraint-frame
      :save-bits t
      :panes
      `((huey w:window :label "huey")
        (dewey w:window :label "dewey")
        (louie w:window :label "louie")
        (main-pane w:window :label "Main Pane")
        (random-pane w:window)
        (menu w:command-menu
          :item-list ("Foo" "Bar" "Baz")))
      :constraints
      `((first-config . ((top-strip main-pane)
                          ; 0.3 of frame height
                          ((top-strip :horizontal (.3)
                            (huey dewey louie)
                            ((huey :even)
                             (dewey :even)
                             (louie :even))))
                          ; Remainder of height
                          ((main-pane :even))))
        (second-config . ((main-pane bottom-strip)
                          ; 0.2 of frame height
                          ((bottom-strip :horizontal (.2)
                            (random-pane menu)
                            ((menu :ask :pane-size)
                             (random-pane :even))))
                          ; Remainder of height
                          ((main-pane :even))))))
    ))
  ; Store the window this function was called from so the function can return to it.
  (setq old-window w:selected-window)

  ; Set up the constraint frame for the first configuration, expose the first configuration, display it for 5 seconds,
  ; and then expose the second configuration.
  (send window-instance :set-configuration 'first-config)
  (send window-instance :expose)
  (process-sleep (* 60 5))

  ; Get the window for the menu pane so we can set up its I/O buffer.
  ; This I/O buffer will tie the menu pane to the main-pane pane.
  (setq menu-window (send window-instance :get-pane 'menu))
  (setq my-io-buffer (w:make-io-buffer 50.))
  (send menu-window :set-io-buffer my-io-buffer)

  ; Get the window for the main-pane and tie the I/O buffer to it.
  (setq main-pane-window (send window-instance :get-pane 'main-pane))
  (send main-pane-window :set-io-buffer my-io-buffer)

  ; Set up the constraint frame for the one that contains the menu.
  (send window-instance :set-configuration 'second-config)

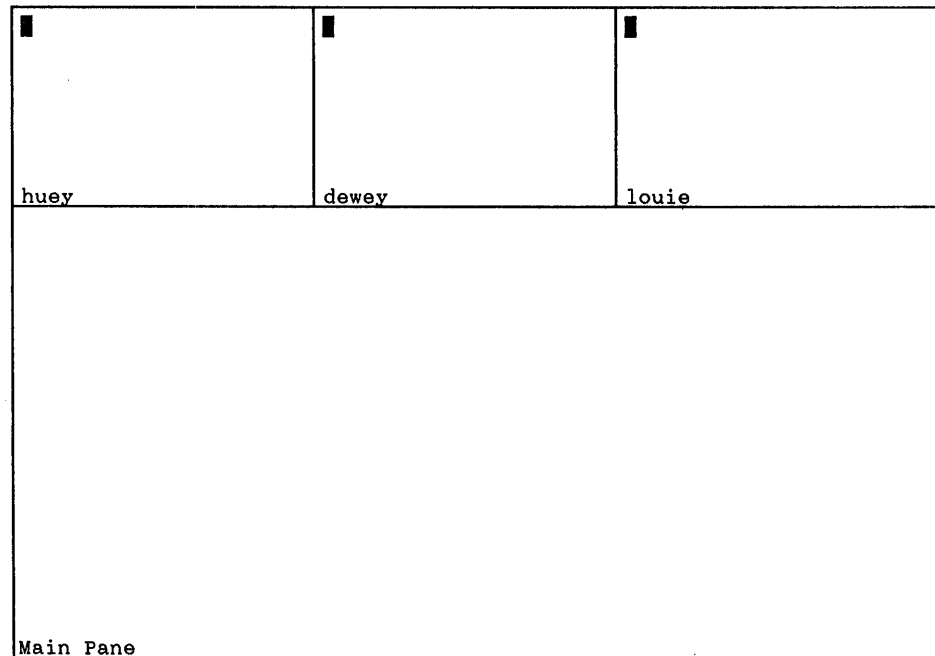
  ; Make the main-pane the one that is tied to the keyboard.
  (send main-pane-window :select)

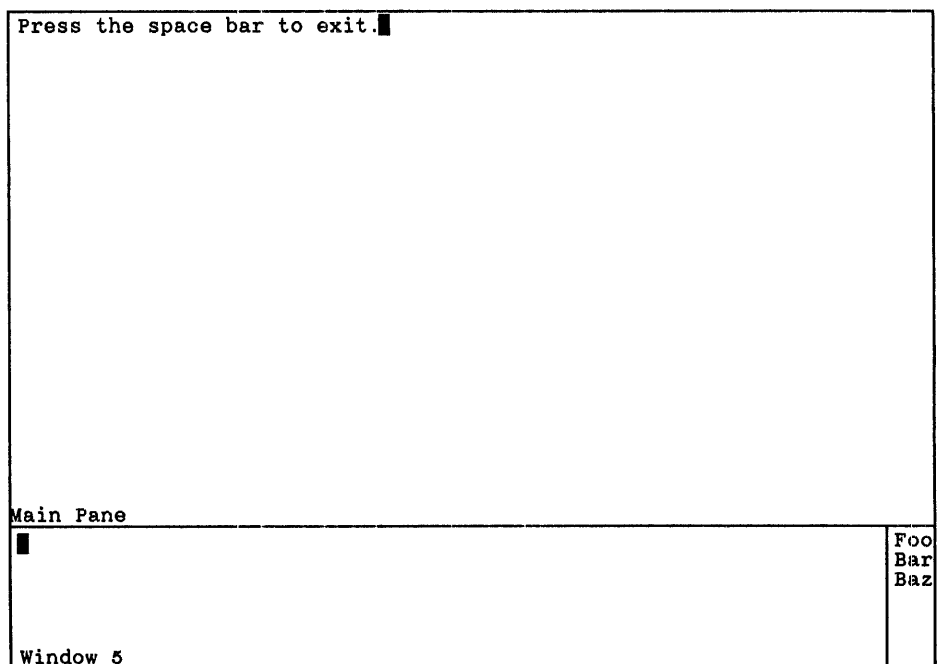
  ; Display this window until the user presses the space bar, then return to the original window.
  (send main-pane-window :string-out "Press the space bar to exit.")
  (loop until (equal (setq char (read-char main-pane-window)) #\SPACE)
    do (send main-pane-window 'string-out (format nil "-%Input=-A" char)))
  (send old-window :select)
)
```

The three small panes in the first configuration are named `dewey`, `huey`, and `louie` (named after the three drones in the movie *Silent Running*); the other pane in this configuration is named `main-pane`. In the second configuration, the top pane is named `main-pane`, and the other two panes at the bottom are named `random-pane` and `menu-pane`. All of these panes, except the `menu-pane`, are created using the `w:window` flavor. The `menu-pane` is created using the `w:command-menu` flavor.

The constraint description specifies the two configurations and the layouts of the panes within them. The first configuration, named `first-config`, specifies that the ordering of the panes is `top-strip` at the top and `main-pane` at the bottom. The name `top-strip` represents another embedded configuration. For a top-level configuration, the screen layout is arranged top-down (in other words, vertically) by default. In the `top-strip` the panes are laid out in a horizontal fashion, using up 30 percent of the vertical space. Within `top-strip`, the ordering of the panes is `huey`, `dewey`, and `louie`, from left to right. Each of these three panes takes up the same amount of space, as specified by the keyword `:even`. The other pane in this configuration, `main-pane`, takes up the space not used by the `top-strip` ($100\% - 30\% = 70\%$).

The second configuration, named `second-config`, specifies that the ordering of the panes is `main-pane` at the top and `bottom-strip` at the bottom. The pane `main-pane` is the same as in the configuration `first-config`. The `bottom-strip` is another name for an embedded configuration. This embedded configuration has its panes laid out horizontally and takes up 20 percent of the vertical space on the window. Thus, `main-pane` is a different size in each configuration. The ordering of the horizontal windows is `random-pane` and `menu-pane`, from left to right. The horizontal space needed by the `menu-pane` is determined by computing the space needed by the menu. The `random-pane` uses up the remainder of the horizontal space for this configuration. The other pane in the `second-config` configuration, `main-pane`, uses up the vertical space not used by `bottom-strip`.





Towards the middle of the `multi-config` function, the `:expose` method is invoked for `window-instance`. This function exposes the `first-config` configuration. After this configuration is visible for five seconds, the `:set-configuration` method is invoked for `window-instance`. This method changes the current configuration from `first-config` to `second-config`. After this change is made, the `:select` method is invoked for `main-pane-window` to show the second configuration.

Note that the pane `menu-pane` in the `multi-config` function is a list and not a form to be evaluated. If the list were a value stored in a variable, then it would have been necessary to write the `:panes` option using the backquote feature. For example, if the variable name were `the-list-of-items`, then the following code shows how the `:panes` option would look.

```
:panes
  `(huey w:window)
    (dewey w:window)
    (louie w:window)
    (main-pane w:window)
    (random-pane w:window)
    (menu-pane w:command-menu
      :item-list ,the-list-of-items))
```

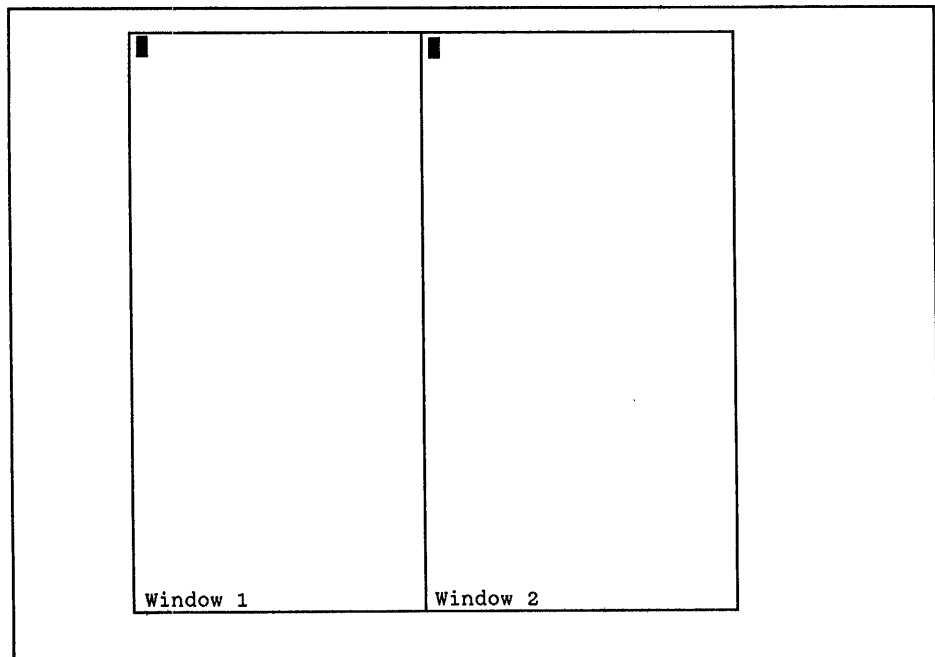
Note the backquote in front of the `huey` pane and the comma in front of the variable name `the-list-of-items`. The backquote causes the comma to be handled in a special way. For this case, the comma causes the value of the variable `the-list-of-items` to be inserted at that place in the `:panes` option. (For more details on backquoting, refer to the *Explorer Lisp Reference* manual.)

**Horizontal
Constraint Frame**

15.5.4 The following code produces a constraint frame that consists of two panes: a left pane and a right pane. In this example, the `:edges` initialization option is specified—in pixel coordinates—before the `:panes` initialization option. These edges are 100 for the left, 100 for the top, 600 for the right, and 600 for the bottom. That is, the upper left corner of the constraint frame is at the pixel coordinate (100,100), with the width and height being 500 pixels. Note that options other than `:edges` could have been specified. These initialization options must be those allowed by the `w:minimum-window` flavor. Recall that frames are built from the `w:minimum-window` flavor, not from the `w>window` flavor.

```
(defun side-by-side (&aux side-by-side-window-instance)
  (setq side-by-side-window-instance
    (make-instance 'w:bordered-constraint-frame
      :edges '(100 100 600 600)
      :panes '((left-pane w>window)
              (right-pane w>window))
      :constraints '((main . ((whole-thing)
                              ((whole-thing :horizontal (:even)
                                (left-pane right-pane)
                                ((left-pane :even)
                                 (right-pane :even))))))))))
  (send side-by-side-window-instance :expose))
```

The panes for the constraint frame are named `left-pane` and `right-pane`. Both panes are created using the `w>window` flavor. The constraint description specifies the layout of the panes and their use of the window space. The layout of the panes is defined in the embedded configuration named `whole-thing`. This embedded configuration is to be laid out horizontally with the `left-pane` on the left and `right-pane` on the right. Both of these panes take up an equal amount of horizontal space.



Specifying Panes and Constraints

15.6 The following pages explain how to specify the panes of a constraint frame.

NOTE: Most of the code described in the following pages can be generated automatically by using the constraint frame editor (WINIFRED). The description that follows explains the details of the code generated by WINIFRED. Most users can skip to paragraph 15.7, Constraint Frame Keys.

When a constraint frame is created, you must specify two initialization options—**:panes** and **:constraints**. The **:panes** option specifies which panes are to be in the constraint frame. The **:constraints** option specifies the set of constraints for each of the configurations that the window can assume. To allow windows within the constraint frame to be referenced, you give them internal names. These names are used as arguments to constraint frame methods such as the **:set-configuration** method, which was used in the example in paragraph 15.5.3, Multiple-Configuration Constraint Frames.

Both of the following initialization options work by initializing instance variables that are then examined by the **:init** methods of constraint frames. Instead of using the initialization options, you can set the instance variables directly in a **:before :init** method.

:panes *pane-descriptions* Initialization Option of *all constraint frame flavors*

Required for all flavors of constraint frames. *pane-descriptions* is a list of pane descriptions. Every pane description has the following form:

(pane-name flavor . options)

where:

pane-name is the internal name (a symbol) of a pane.

flavor is the flavor of which the pane is an instance.

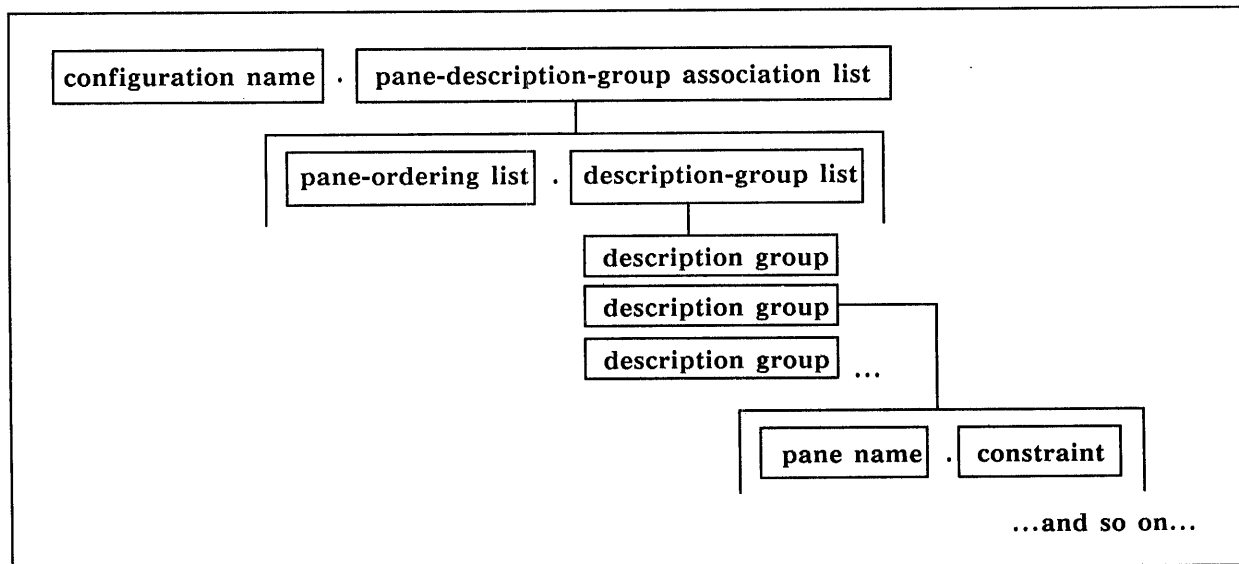
options is a list that is appended to the initialization property list for the pane when it is created. (See paragraph 15.5.2, Graphics Constraint Frame, for an example of these options.)

When the frame is first created, all of its panes are created using the *flavor* and *options* specified. The frame adds some of its own options to control the position and shape of the window. You need not specify any of these controlling factors in the *options* list because these factors are automatically provided.

:constraints *configuration-description-list* Initialization Option of *all constraint frame flavors*

Required for all flavors of constraint frames. *configuration-description-list* is a list of configuration descriptions. The format of the configuration descriptions is complex and is explained in the following paragraphs.

Configuration 15.6.1 A configuration specifies how panes are arranged within a window. A configuration description (such as that shown in the code in paragraph 15.5.3) is a hierarchical list of descriptions and constraints. The following figure shows the entire diagram at once. In the discussions that follow, the portion of the diagram being discussed is surrounded by dark borders.



In the following code, there are two *configuration-name*, *pane-description-group* association list pairs. These are `((first-config. (top-strip main-pane)...`) and `((second-config. (main-pane bottom-strip)...`). The configuration names are `first-config` and `second-config`. Because the first configuration is more complicated, it is the only one discussed. Likewise, the first description group within the first *description-group list* is the more complicated, so it is discussed but the second is not.

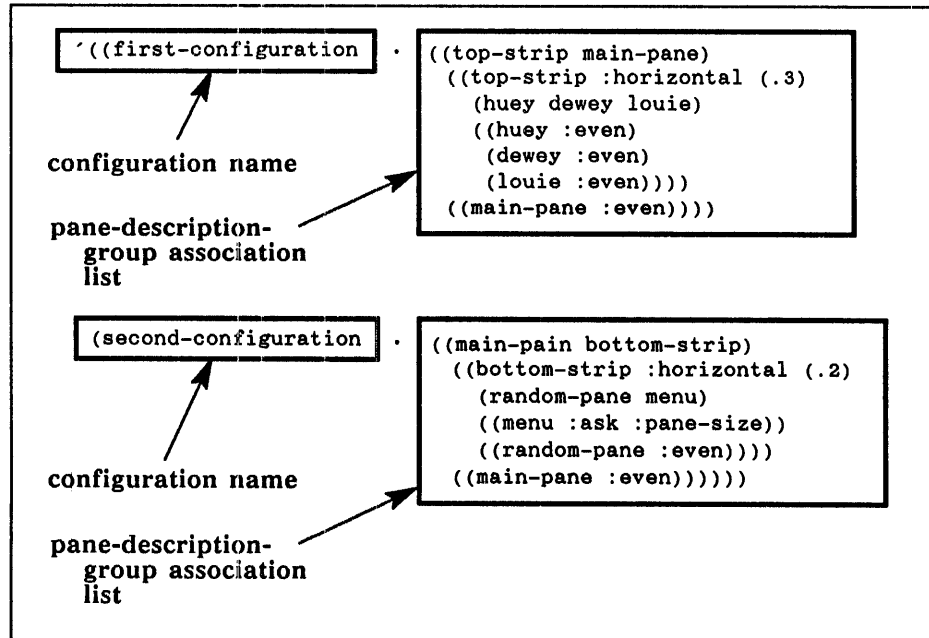
```
:constraints
^((first-configuration . ((top-strip main-pane)
  ((top-strip :horizontal (.3)
    (huey dewey louie)
    ((huey :even)
    (dewey :even)
    (louie :even))))
  ((main-pane :even))))
  (second-configuration . ((main-pane bottom-strip)
    ((bottom-strip :horizontal (.2)
      (random-pane menu)
      ((menu :ask :pane-size)
      ((random-pane :even))))
    ((main-pane :even))))))
```

The following pages describe the structure of a constraint frame. Each description includes three parts: the name of the step, a brief description of the step, and a detailed discussion of what the step is. In some cases the detailed description is omitted because it would be redundant.

Configuration Description 15.6.1.1

Brief Description At the highest level of the hierarchy is the *configuration description*, which contains a configuration-name associated with a *pane-description-group association list*. The frame has a configuration description for each configuration the frame can assume.

Detailed Description A configuration description associates a *configuration-name* with a specific configuration. This association allows you to define several configurations and display the appropriate configuration of panes when that configuration best displays the information contained in the panes.

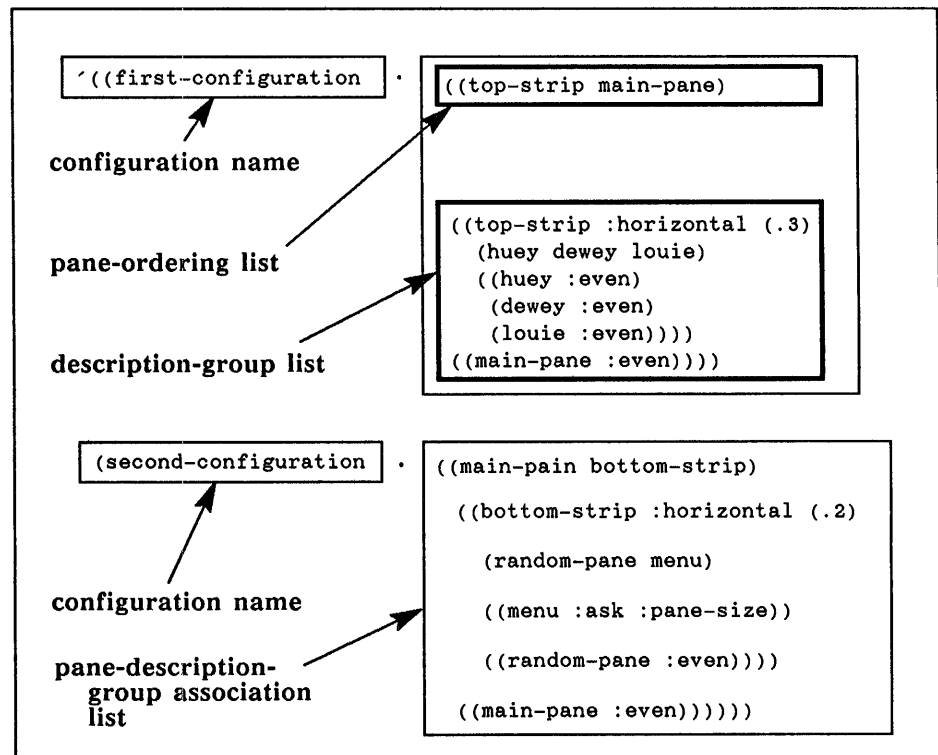


Pane-Description-Group Association List

15.6.1.2

Brief Description The *pane-description-group association list* contains the *pane-ordering list* associated with a *description-group list*.

Detailed Description The *pane-ordering list* names the panes contained in the *first-configuration frame*. *top-strip* and *main-pane* are names for the panes. The order of the pane names within the *pane-ordering list* determines the ordering of the panes as they appear in the window: left to right if the panes are stacked horizontally, or top to bottom if the panes are stacked vertically. The first pane in the list appears at the left or top of the configuration, and the last pane in the list appears at the right or bottom. You can have as many panes as you can fit into the window.



Description-Group List 15.6.1.3

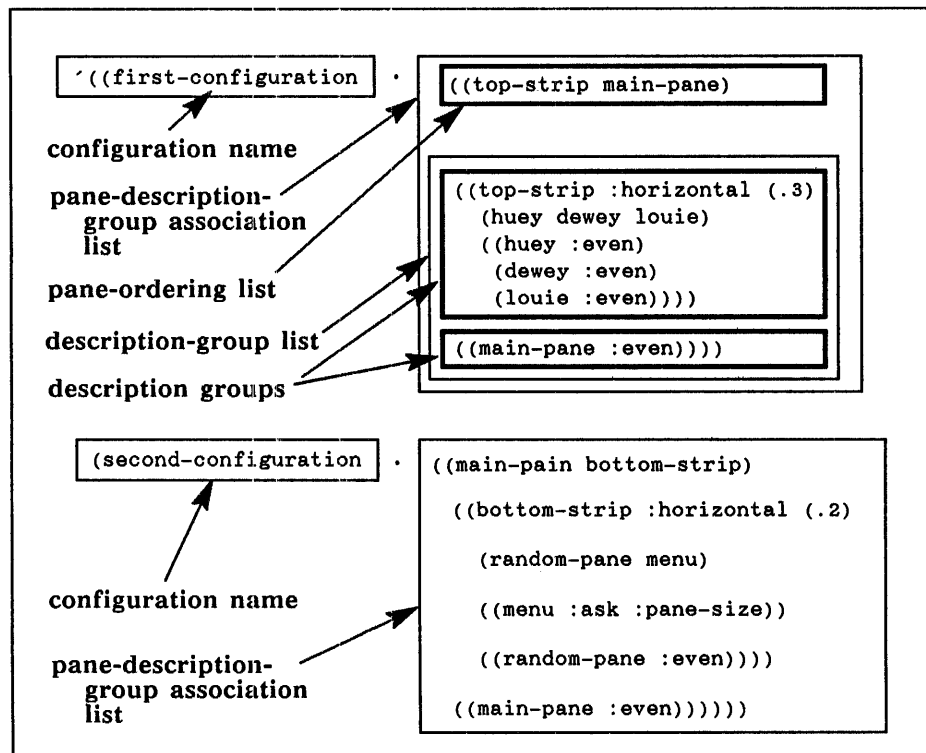
Brief Description The *description-group list* contains a *description-group* for each of the panes named in the *pane-ordering list* within the *pane-description-group association list*.

Detailed Description The *description-group list* in this example contains two pane names associated with two constraints. Your *description-group list* can contain more or fewer panes, depending on your requirement. The *top-strip* pane is associated with a complex constraint, while the *main-pane* is associated with the keyword `:even`.

The ordering of the panes within the *description-group list* determines which panes are allocated space in the window first, and need not be the same as the ordering of the panes within the *pane-ordering list*. The first pane named in the *description-group list* is allocated window space first, and the last pane in the *description-group list* is allocated window space last.

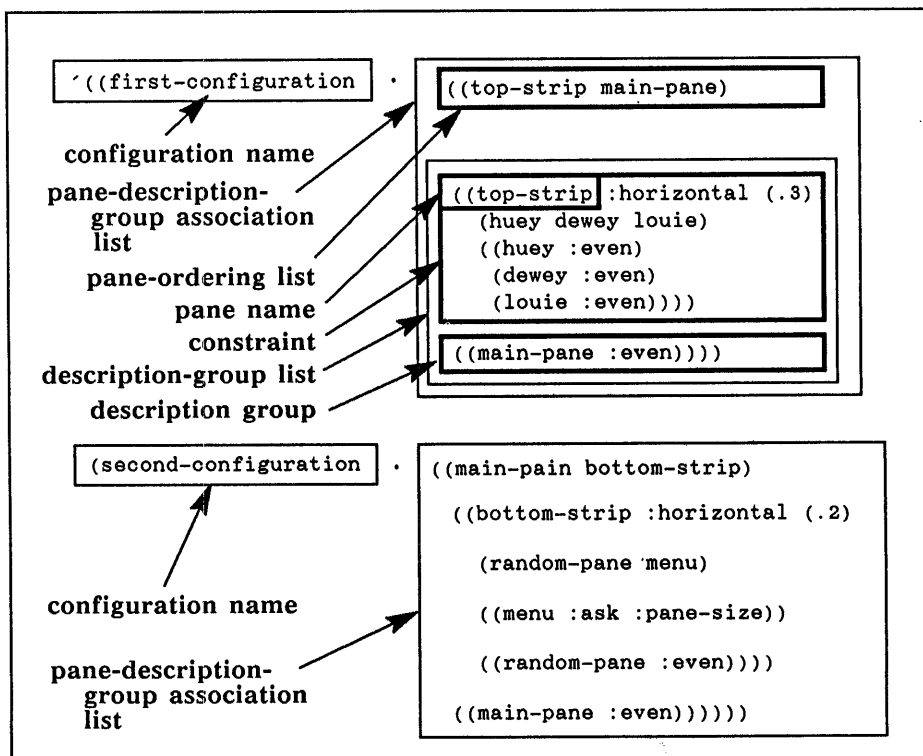
Description-Group 15.6.1.4

Brief Description The *description-group* contains a *pane-name* and a *constraints*.



Constraint 15.6.1.5

Brief Description The *constraint* for `top-strip` contains a key that specifies how this pane is filled followed by any arguments that are needed. The format of the *constraint* is `(key arg1 arg2...)`.



Detailed Description The key can be either a fixnum, flonum, or a specific keyword. The arguments depend on what key is used. Paragraph 15.7, Constraint Frame Keys, discusses the keys that can be used and what argument(s), if any, apply to each key.

Specifically, in the example code:

- The `:horizontal` keyword specifies that the subpanes are to be stacked one beside the other—that is, horizontally—within the `top-strip` pane.
- The `(.3)` is a constraint that specifies that `top-strip` is allocated 30 percent of the window `top-strip` appears in.
- The lower-level *pane-description-group association list* is an argument of the `:horizontal` keyword.

At this point you have defined the constraints of the pane. However, in this example, `top-strip` is not a pane because the code as shown in paragraph 15.5.3 does not include `top-strip` as part of the `:panes` description. Here, `top-strip` is a name used to manipulate several panes as a group; `top-strip`, then, contains another *pane-description-group association list*. The following discussions apply only to the example code in paragraph 15.5.3. The general descriptions given previously for each component still apply.

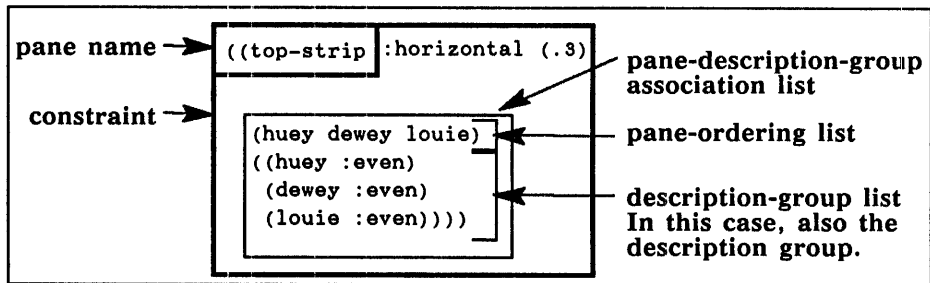
Pane-Description-Group Association List

15.6.1.6

Brief Description The *pane-description-group association list* contains another *pane-ordering list* and another *description-group list*.

Detailed Description The difference between this *pane-description-group association list* and the previous one is that the *pane-ordering list* contains the names of three actual panes: *huey*, *dewey*, and *louie*.

If you wanted to, you could name groups of panes—or a mixture of panes and groups of panes—at this point. Each additional group adds another level to the complexity of the configuration.



Description-Group List

15.6.1.7

Brief Description The *description-group list* is a list of description groups.

Description-Group

15.6.1.8

Brief Description The *description-group* contains the names of three panes associated with the keyword `:even`.

Detailed Description The panes are *huey*, *dewey*, and *louie* and each is associated with the `:even` keyword; `:even` specifies that each pane gets an even portion of `top-strip`.

**Constraint
Frame Keys**

15.7 Constraint frame keys are used to determine the size and sometimes the position of panes within a constraint frame. For example, the **:even** keyword used in the preceding examples was used to specify that each of the three panes was the same size.

**Arguments Used With
w:basic-frame**

15.7.1 You can specify a fixed number, a floating-point number, or the **:even** and **:ask** keywords in a constraint for a pane.

fixnum

15.7.1.1 A *fixnum* argument lets you specify the absolute size of the pane in pixels. An argument can be optional; it can be the symbol **:lines** or the symbol **:characters**, meaning that the *fixnum* is in units of lines or characters. The *fixnum* is multiplied by the value of the **w:line-height** or **w:char-width** instance variable of the window to compute the actual size of the pane. If a second argument is given, it must be the name of a pane. In this case, it is that pane's **w:line-height** or **w:char-width** instance variable that is multiplied by the *fixnum* to compute the actual size of the pane. Note that embedded configurations do not have a **w:line-height** or **char-width** instance variable; thus, for this case, if the first argument is present, then the second must be present, too. Embedded configurations are described in paragraph of the same name, 15.8.

flonum

15.7.1.2 A *flonum* argument lets you specify that a certain fraction of the remaining space should be taken up by this window. An argument may be optionally given: **:lines** or **:characters**. When a *flonum* is present, the pane's size is rounded down to the nearest multiple of the pane's **w:line-height** or **w:char-width** instance variable. If a second argument is given, it must be the name of a pane. In this case, it is that pane's **w:line-height** or **w:char-width** instance variable that is used to perform the rounding calculation. Note that embedded configurations do not have a **w:line-height** or **w:char-width** instance variable; thus, for this case, if the first argument is present, then the second must be present, too.

The distinction between descriptors in the same group and descriptors in different groups is important when you use the *flonum* constraint. If there is one description group with two descriptions, each of which requests 20 percent of the remaining space, then both panes receive the same amount of space. However, if there are the same two descriptions but they are in successive description groups, then the first pane receives 20 percent of the remaining space, and the second pane receives 20 percent of what remains after the first pane is allocated. Thus, the amount of space remaining is recomputed at the end of each description group, but not at the end of each description.

:even

15.7.1.3 The **:even** keyword divides all of the remaining space among all of the panes in this description group. Because all of the remaining space is allocated, this description group must be the last one in a configuration. Furthermore, if there are any other descriptions in this description group, they also must specify the **:even** constraint.

Typically, you should use **:even** for at least one pane in every configuration so that the entire frame will be filled, and the panes are adjusted to fit together and use up all of the available space.

Remember that even though the `:even` constraint must be in the last description group of a configuration, these panes must also be at the bottom or the right side of the frame. The ordering of the panes is determined by the *ordering* list, which is specified before the first description group of a configuration, not by the order in which the descriptions are listed.

`:ask` 15.7.1.4 The `:ask` keyword lets the window determine its size. The format of a constraint that uses `:ask` follows:

```
(:ask method arg1 arg2 ...)
```

The *method* is invoked with the specified arguments. The result returned by the method is the height required if the stacking is vertical or the width required if the stacking is horizontal. Note that the arguments are the actual values of the arguments, and that they are not forms to be evaluated. If you want evaluated forms passed to the method, you must use something like the backquote feature to control the evaluation.

The arguments passed to the method are a combination that describes the pane and that can be specified by the user. The first five arguments describe the pane, and the remaining arguments are those provided by the user. The first five arguments are as follows:

remaining-width — The amount of width remaining to be used at the time this description is elaborated (that is, after all of the panes in previous description groups and all earlier panes in this description group have been allocated).

remaining-height — The amount of height remaining to be used at the time this description is elaborated (that is, after all of the panes in previous description groups and all earlier panes in this description group have been allocated).

total-width — The amount of width remaining to be used by all of the sections of this description group (that is, the amount of room left after all panes in previous description groups have been allocated but none of the panes in this description group have been allocated).

total-height — The amount of height remaining to be used by all of the sections of this description group (that is, the amount of room left after all panes in previous description groups have been allocated but none of the panes in this description group have been allocated).

stacking — Either `:vertical` or `:horizontal`, depending on the current stacking.

The `:limit` Clause 15.7.2 Any constraint can be optionally preceded by a `:limit` clause. The format of the `:limit` clause looks like the following:

```
(:limit limit-specification key arg1 arg2 ...)
```

The `:limit` clause allows you to specify a minimum value and a maximum value that is applied to the size computed for the pane. If the computed size is smaller than the minimum, then the minimum value is used; if the

computed size is larger than the maximum, then the maximum value is used. The format of the *limit-specification* is one of the following:

Format of Clause	Description
<i>(minimum maximum)</i>	The minimum and maximum values represent pixel limits.
<i>(minimum maximum :lines)</i> <i>(minimum maximum :characters)</i>	The minimum and maximum values represent the number of lines or characters, respectively. The line and character height are from the pane being constrained.
<i>(minimum maximum :lines pane-name)</i> <i>(minimum maximum :characters pane-name)</i>	The minimum and maximum values represent the number of lines or characters, respectively, using the line or character height, respectively, from another pane. The line and character height are from another pane. The latter case is useful when the pane being constrained does not have a line height or character width, as is the case with embedded configurations. Embedded configurations are described in the paragraph of the same name, 15.8.

The other values in the `:limit` clause, *key*, *arg1*, *arg2*, ..., are the same as specified in a constraint. Recall that the `:limit` clause precedes the constraint.

:pane-size and Similar Methods

15.7.3 Different flavors of windows accept certain messages suitable for use with the `:ask` keyword. By convention, several kinds of windows, such as menus, define a method named `:pane-size`. For example, the `:pane-size` method for menus computes the amount of space needed to display all of the items of the menu, given the amount of space available. No additional arguments need be specified. Other useful methods, which are defined for all windows, are `:square-pane-size` and `:square-pane-inside-size`. The `:square-pane-size` method makes the pane take up enough room to form a square. The borders are included in the calculation. The `:square-pane-inside-size` method is similar except that the square is inside of the borders.

The keywords used by the `:square-pane-inside-size` method are `:ask-window`, `:funcall`, and `:eval`:

- The `:ask-window` keyword is a variation of `:ask`. The format follows:

```
(:ask-window pane-name message-name arg1 arg2 ...)
```

The `:ask-window` keyword works like the `:ask` keyword except that the method is sent to the pane named *pane-name* instead of the pane being described. This keyword is primarily for embedded configurations when the screen space requirements of a pane within the embedded configuration are to control the size of the embedded configuration.

- The `:funcall` keyword allows you to supply a function to be invoked that computes the amount of space needed. The format is as follows:

```
(:funcall function arg1 arg2 ...)
```

where:

function is the function invoked with the first six arguments as described below.

arg1, arg2 ... are the remaining arguments—if any—and are supplied by the user. The first argument is *constraint-node*, an internal data structure that the user can examine. The next five arguments are the same as the five arguments to the `:ask` keyword, described previously.

- The `:eval` keyword is like the `:funcall` keyword but provides a form instead of a function and arguments. The format is as follows:

```
(:eval form)
```

The six special values that are passed as arguments when the `:funcall` constraint type is used can be accessed by the form as the values of the following special variables:

```
w:**constraint-node**
w:**constraint-remaining-width**
w:**constraint-remaining-height**
w:**constraint-total-width**
w:**constraint-total-height**
w:**constraint-stacking**
```

Embedded Configurations

15.8 Embedded configurations are useful when you want either a more complex pane layout or something other than a pane to appear in the constraint frame. For example, the font editor window uses an embedded configuration. The difference between a pane and an embedded configuration is in the constraint description. The format of a constraint for an embedded configuration is as follows:

```
(embedded-configuration-name key arg1 arg2 ...)
```

where:

embedded-configuration-name is the name of the embedded configuration.

The keys for this embedded configuration are described below, and the arguments supplied depend upon which key is selected:

- The `:blank` keyword fills this section with some constant pattern. The format of the description follows:

```
(embedded-configuration-name :blank pattern . constraint)
```

The constraint is used to compute the size of the section, as is done for ordinary panes. The pattern can be any of the following:

- The `:white` pattern fills the section with all 0s.
- The `:black` pattern fills the section with all 1s.
- An array fills the section with the contents of the array. This pattern is replicated across the section. The `bitblt` function is used to perform this replication.
- A symbol that is the name of a function that accepts six arguments. The function is invoked to actually perform the filling of the section. The following values are passed as arguments to the function:

Value	Description
<i>constraint-node</i>	An internal data structure being passed
<i>x-position</i>	The pixel coordinate of the left side of the embedded configuration
<i>y-position</i>	The pixel coordinate of the top of the embedded configuration
<i>width</i>	The width in pixels of the embedded configuration
<i>height</i>	The height in pixels of the embedded configuration
<i>screen-array</i>	The two-dimensional array into which the function should write the pattern for the embedded configuration

- A list that is similar to a symbol (as previously discussed) but that allows you to pass additional arguments. The first element of the list is the name of the function to be called. The remaining elements of the list are the additional arguments. When the function is invoked, the arguments that are passed to it are the six listed for the symbol; any additional arguments in the list follow those six arguments.
- The `:horizontal` or `:vertical` keyword provides a description of an embedded configuration that is used to subdivide the section into more panes and embedded configurations. If you specify `:vertical`, the section is split up with vertical stacking. If you specify `:horizontal`, the section is split up with horizontal stacking. The format of this description is:

(embedded-configuration-name
`:horizontal constraint . configuration-description`)

or:

(embedded-configuration-name
`:vertical constraint . configuration-description`)

where:

constraint is an ordinary constraint as was described previously.

configuration-description is a configuration description as was described previously. This configuration description specifies how this section is subdivided into its own components.

**Constraint
Frame Methods**

15.9 The following methods are useful when constraint frames are used.

- :pane-size** *remaining-width remaining-height total-width total-height stacking* Method of *windows*
- :square-pane-size** *remaining-width remaining-height total-width total-height stacking* Method of *windows*
- :square-pane-inside-size** *remaining-width remaining-height total-width total-height stacking* Method of *windows*

Invoked when a constraint uses the **:ask** keyword with one of these methods specified. The **:pane-size** method returns the size in pixels that the pane requires, for either the vertical or horizontal stacking direction. The **:square-pane-size** method returns the outside size required to make the pane the largest square possible; the **:square-pane-inside-size** method returns the inside size required to make the pane the largest square possible.

- :get-pane** *pane-name* Method of **w:basic-constraint-frame**

Takes a pane name as an argument and returns the pane itself, which is a window. Recall that pane names are symbols that are used to refer to panes. They are not actually the panes themselves. If you perform something requiring a pane window, then **:get-pane** retrieves the pane window. For example:

```
(setq temp-window (send self :get-pane 'foo-window))
```

- :pane-name** *pane* Method of **w:basic-constraint-frame**

Takes a pane window as an argument and returns the pane name to which it corresponds. This is the inverse of the **:get-pane** method. For example:

```
(setq temp-pane (send self :pane-name 'foo-window))
```

- :create-pane** *name flavor &rest options* Method of **w:basic-constraint-frame**

Creates and returns a window that serves as a pane of this frame. The **:create-pane** method creates this frame's panes using the panes's constraints as arguments. *name* is the name of the pane to be used. The pane is of the specified *flavor* with the initialization options set by the *options* arguments.

- :send-pane** *pane-name method &rest arguments* Method of **w:basic-constraint-frame**

Sends *method* to *pane-name*, which must be in the **:panes** specification of this frame.

- :send-all-exposed-panes** *method &rest arguments* Method of **w:basic-constraint-frame**

- :send-all-panes** *method &rest arguments* Method of **w:basic-constraint-frame**

Sends *method* to all the exposed panes of this frame (for **:send-all-exposed-panes**) or to all the panes of this frame, including the nonexposed panes (for **:send-all-panes**).

- :configuration** *configuration-name* Initialization Option of **w:basic-constraint-frame**
Gettable, settable

Configures the frame to the parameters specified by *configuration-name*.

- :get-configuration** *configuration-name* Method of **w:basic-constraint-frame**
Returns the internal—parsed—data structure describing the specified configuration of *configuration-name*. *configuration-name* describes which windows are included and the constraints for the windows.
- :redefine-configuration** *config-name new-config* Method of **w:basic-constraint-frame**
&optional (*parsed-p t*)
Redefines the configuration from that given by *config-name* to the configuration specified by *new-config*. If *parsed-p* is *t*, *new-config* must be in parsed form, such as the value returned by the **:get-configuration** method. If *parsed-p* is *nil*, *new-config* is treated as a configuration-description such as that used when initially specifying the frame's constraints.

Pane-Frame Interaction

15.10 Several fundamental window methods actually ask the window's superior what to do. This inquiry has no effect for a top-level window but becomes important when the window's superior is a frame. The superior can decide whether the methods should actually proceed as requested. The following discuss how the **:expose**, **:deexpose**, **:bury**, **:select**, and **:set-edges** methods are handled.

- The **:expose**, **:deexpose**, **:bury**, and **:select** methods first send a method to the superior using the **:inferior-expose**, **:inferior-deexpose**, **:inferior-bury**, or **:inferior-select** method. The pane itself is the argument for the method used.

If the method sent to the superior returns **non-nil**, the method is performed on the pane as usual. Otherwise, it is skipped.

- The **:set-edges** method in conjunction with **:inferior-set-edges** sets the edges of the pane. An **:inferior-set-edges** method is sent to the superior; **:inferior-set-edges** arguments consist of the pane followed by the arguments of the **:set-edges** method. If the first value returned by **:inferior-set-edges** is **non-nil**, the pane's edges are changed as requested. Otherwise, the pane's edges are not changed, and the remaining values from **:inferior-set-edges** are returned from **:set-edges**.

Of course, the frame can change the pane's edges in another way and then return **nil**.

The **w:basic-frame** flavor defines only the **:inferior-select** method to do anything nontrivial; **:inferior-select** makes the pane the frame's selection substitute and then sends a **:select** message to the frame. The other methods merely return **non-nil**. Thus, there is minimal interaction between the frame and its inferiors.

The **w:frame-forwarding-mixin** flavor defines the **:inferior-expose**, **:inferior-deexpose**, and **:inferior-bury** methods to ensure that the frame and its panes are all exposed or deexposed together.

w:frame-forwarding-mixin

Flavor

Defines the **:inferior-expose**, **:inferior-deexpose**, and **:inferior-bury** methods for a frame that normally causes the **:expose**, **:deexpose**, or **:bury** methods on panes to expose, deexpose, or bury the frame rather than the pane.

An `:inferior-set-edges` method is also defined for internal reasons only. Its purpose is to avoid a user-visible change in behavior rather than to provide such a change.

The `w:frame-forwarding-mixin` flavor is a mixin of the `w:constraint-frame` flavor and the other standard instantiable flavors of a constraint frame.

:pane-types-alist Method of *frames*

Returns a menu item list used for handling the Create item in the screen editor when editing the panes of this frame. The value of the menu item should be a window flavor to be created or a list to be evaluated in order to return a flavor.

The menu item's value, or the result of evaluating it, can also be `t`, which directs the screen editor to read a flavor name from the user.

The Selected Pane

15.10.1 A frame is normally operated with one of its inferiors as a selection substitute. The selection substitute of a frame is also called the *selected pane*; this feature was formerly available only in frames. Unless you mix `w:select-mixin` into your frame flavor, the frame itself cannot be the selected window. Therefore, it is important to provide a selection substitute when creating the frame.

You can provide a selection substitute by sending a `:set-selection-substitute` message in an `:after :init` method, as shown in the following example:

```
(defmethod (my-frame :after :init) (ignore)
  (send self :set-selection-substitute
            (send self :get-pane 'interaction-pane)))
```

Explicitly selecting a pane with the `:select` method actually sets the frame's selection substitute by means of the forwarding mechanism shown in the previous example.

In a constraint frame, or any other frame that uses `w:frame-forwarding-mixin`, you should not attempt to select a pane that is not already exposed, because of the effects of forwarding on the `:expose` method.

:selected-pane *pane-name* Initialization Option of `w:basic-constraint-frame`

Initializes the selection substitute for the constraint frame. *pane-name* specifies the pane to be selected

w:interaction-pane Flavor

Used to create a pane for reading and echoing multicharacter commands in systems using frames. The pane using `w:interaction-pane` should be the selected pane.

TEXT SCROLL WINDOWS

Using Text Scroll Windows

16.1 Text scroll windows provide a means to display a number of lines that are of the same type with scrolling. For example, the Inspector uses text scroll windows to display the components of a structure.

```

#<INSPECTOR-INTERACTION-PANE Inspector Interaction Pane 5 4106057 exposed>
An object of flavor TV::INSPECTOR-INTERACTION-PANE. Function is #<EQ-HASH-TABLE (Funcallable) 34221037>
TV:SCREEN-ARRAY:          #<ART-1B-56-1024 15241030>
TV:LOCATIONS-PER-LINE:   32
TV:OLD-SCREEN-ARRAY:     NIL
TV:BIT-ARRAY:           #<ART-1B-741-1024 7447750>
TV:NAME:                 "Inspector Interaction Pane 5"
#<FONT CPTFONT 71301633>
Named structure of type FONT
TV:FONT-FILL-POINTER:    256
TV:FONT-NAME:           FONTS::CPTFONT
TV:FONT-CHAR-HEIGHT:    11
TV:FONT-CHAR-WIDTH:     8
TV:FONT-RASTER-HEIGHT:  11
#<STANDARD-SCREEN Main Screen 40100143 exposed>
An object of flavor TV::STANDARD-SCREEN. Function is #<EQ-HASH-TABLE (Funcallable) 11761072>
TV:SCREEN-ARRAY:          #<ART-1B-754-1024 71301442>
TV:LOCATIONS-PER-LINE:   32
TV:OLD-SCREEN-ARRAY:     #<ART-1B-754-1024 71301442>
TV:BIT-ARRAY:           NIL
TV:NAME:                 "Main Screen"
TV:LOCK:                 NIL
TV:LOCK-COUNT:           0
TV:SUPERIOR:            NIL
TV:INFERIORS:           (#<INSPECT-FRAME Inspect Frame 3 4104142 exposed> #<ZMACS-FRAME Znacs Frame 1 4100172 desxpo
TV:EXPOSED-P:           T
TV:EXPOSED-INFERIORS:   (#<INSPECT-FRAME Inspect Frame 3 4104142 exposed*)
TV:X-OFFSET:            0
TV:Y-OFFSET:            0
TV:WIDTH:               1024
TV:HEIGHT:              754
TV:CURSOR-X:            0
TV:CURSOR-Y:            0
TV:MORE-UPOS:          740
TV:TOP-MARGIN-SIZE:     0
TV:BOTTOM-MARGIN-SIZE: 0
TV:LEFT-MARGIN-SIZE:    0
TV:RIGHT-MARGIN-SIZE:  0
TV:FLAGS:               32768
TV:BASELINE:            9
TV:FONT-MAP:            #<ART-Q-26 -65544460>
TV:CURRENT-FONT:        #<FONT CPTFONT 71301633>
TV:BASELINE-ADJ:        0
TV:LINE-HEIGHT:         13
Flavors Doc Exit Delete #<INSPECTOR-INTERACTION-PANE Inspector Interaction Pane 5 4106057 exposed>
Set= Refresh Modify Config #<FONT CPTFONT 71301633>
Mode Print Edit #<STANDARD-SCREEN Main Screen 40100143 exposed>
Inspect: tv:selected-window
Inspect: fonts:cptfont
Inspect: tv:main-screen
Inspect:
Inspect Frame 3
M2: Lock/Unlock inspector pane, R: System Menu.
03/18/87 08:45:04AM RALPH USER: Keyboard + Romeo: WEBB; SYMBOLS.LISP#1 0

```

A text scroll window updates its display based on a sequence of *items*. Each item generates one line of display. An item can be any Lisp object. How the `:print-item` method is defined controls how the object is displayed. For example, defining `:print-item` to use the `:string-out` method requires the items to be strings. By default, `:print-item` uses the function `prin1`, so each item is a Lisp object to be printed.

If you want to create text scroll windows of your own, you can use the `w:text-scroll-window` flavor.

- w:text-scroll-window** Flavor
- The base flavor for all text scroll windows. It is not instantiable by itself, so you must use it as a mixin with other flavors.
- :print-item** *item line-no index* Method of w:text-scroll-window
- The primitive used by all other text scroll window methods to perform output of items. As defined by **w:text-scroll-window**, **:print-item** invokes **prin1** on *item*, ignoring the other arguments. Other flavors built on **w:text-scroll-window** are expected to redefine this method. In any case, no item can print out as more than one line. This restriction is enforced by truncating output at the margin.
- Arguments:*
- item* — The item to be displayed.
 - line-no* — The line number on the window on which the item is displayed.
 - index* — The array index of the item to be displayed. The array is the **w:items** instance variable.

Specifying the Item List

16.2 When specifying an item list, you usually specify an array of items to be displayed or a list of items (which is converted into an array). Items are sometimes referenced by their indexes in the array. A more sophisticated technique is to specify an *item generator*, which is a function that enables you to handle an arbitrarily large number of items without requiring you to actually create them initially. Item generators are described in paragraph 16.4.

- :items** *array* Method of w:text-scroll-window
- Returns the array whose elements are the items to be scrolled through. **w:item** contains the entire set of items to be scrolled through, not just those that are on the screen at any time.
- :set-items** *new-items* &optional (*no-display* nil) Method of w:text-scroll-window
- Sets a new array of items. The item generator of the window is set to **nil**, turning off that feature so that the array of items is actually used. The *new-items* argument can be a suitable array (it should have a fill pointer), a list of items (an array is made from it), or a number of items (the array is made the specified number of items long, but is initially empty).
- If *no-display* is **t**, the items are updated but not redisplayed. The default is **nil**, which causes the items to be redisplayed.
- :top-item** Method of w:text-scroll-window
- Settable*. Default: 0, the first item in the list
- Returns the index of the first item currently being displayed on the first line of the window. The **w:top-item** instance variable, which the **:top-item** method returns, is used to remember the current scroll position. Methods such as **:scroll-redisplay** and **:scroll-to** set this instance variable.
- :number-of-items** Method of w:text-scroll-window
- Returns the number of items this window is currently scrolling through; this is the number of items in the **w:items** instance variable.

- :number-of-item** *item* Method of **w:text-scroll-window**
 Searches the **w:items** instance variable for an element that is the same as the *item* argument and returns the item number (index) of the item where the match is found.
- :item-of-number** *index* Method of **w:text-scroll-window**
 Returns the item identified by the *index* argument located in the **w:items** instance variable.
- :last-item** Method of **w:text-scroll-window**
 Returns the value of the last item to be scrolled through (that is, the one whose index is one less than the number of items).
- :put-item-in-window** *item* Method of **w:text-scroll-window**
 Scrolls the window so that the item specified by the *item* argument appears in the window.
- :put-last-item-in-window** Method of **w:text-scroll-window**
 Scrolls the window so that the last item in the list appears in the window.
- :delete-item** *index* Method of **w:text-scroll-window**
 Modifies the list of displayable items, which is the **w:items** instance variable, removes the item located at the position specified by *index*, and updates the window if that index is within the portion currently displayed.
- :insert-item** *index item* Method of **w:text-scroll-window**
 Adds a new item to the list of items to be displayed. The new item is added before the item currently in the position identified by *index*.
- :append-item** *item* Method of **w:text-scroll-window**
 Appends *item* to the end of the list to be displayed.

The following auxiliary methods are also defined.

- :redisplay** *start end* Method of **w:text-scroll-window**
 Causes a **:print-item** message to be sent for each line in the range identified by the *start* and *end* arguments, which are screen line indexes. This is an internal function.
- The **:redisplay** method should not be redefined, but **:before** and **:after** methods can be placed on **:redisplay** when it is important to know when changes have been made in the screen layout.
- :scroll-redisplay** *new-top delta* Method of **w:text-scroll-window**
 Causes partial redisplay using the **bitblt** function and then uses the **:redisplay** method to redisplay the rest. *new-top* is the new value for the **w:top-item** instance variable. *delta* is the number of lines actually to be scrolled. This is an internal function. This method should not be redefined, but **:before** and **:after** methods can be placed on it.

The **:scroll-position**, **:scroll-to**, and **:new-scroll-position** methods are defined for text scroll windows. They are also required for other kinds of scrolling windows, such as windows that implement scroll bars.

:scroll-position Method of *scrolling windows*

Returns the following four values:

1. *top-line-num* — The line number of the line currently at the top of the window
2. *total-lines* — The total number of lines available to scroll through
3. *line-height* — The height (in pixels) of a line
4. *n-items* — The number of lines that the window has room for

:scroll-to *to type* Method of *scrolling windows*

Scrolls a specified number of lines. *to* specifies the number of lines to scroll forward or backward. Because *to* is not guaranteed to be legal, both types of scrolling must check their arguments for errors. *type* is one of the following values:

- **:absolute** places the line numbered *to* at the top of the window.
- **:relative** adjusts the line displayed at the top of the window by the number of lines specified in the *to* argument. If *to* is positive, text moves upward on the screen.

:new-scroll-position Method of *scrolling windows*

Used by the program managing the window to tell the scrolling facilities that the contents of the window have changed under program control. **:new-scroll-position** should be invoked whenever either the total number of lines to scroll through or the line number at the top of the window is changed by anything except the scrolling facilities.

Scrolling facilities put **:before** or **:after** methods on this method to update their displays when the situation changes.

**Function Text
Scroll Windows**

16.3 Another type of scroll window is the *function text scroll window*. Function text scroll windows allow you to dynamically specify different functions to display items. A function text scroll window has an instance variable that contains the display function used.

w:function-text-scroll-window Flavor

An instantiable function text scroll window.

:print-function *function* Initialization Option of **w:function-text-scroll-window**
Gettable, settable.

Sets the function called to display an item.

:print-function-arg *arg* Initialization Option of **w:function-text-scroll-window**
Gettable, settable. Default: nil

Sets the additional argument to be passed to the print function. The print function's complete list of arguments includes the item itself, the value of **w:print-function-arg**, the window, and the item number.

:setup *list* Method of **w:function-text-scroll-window**

Sets up the text scroll window parameters to the values specified by *list*, which is a list of the following form:

```
(print-function print-function-arg (item... )
  top-item-number label item-generator )
```

where:

print-function is the function that displays the item.

print-function-arg specifies the arguments.

item is the list of items to be displayed. If *item-generator* is non-nil, *item* is ignored.

top-item-number is the first item to be displayed.

label is passed to the **:set-label** method.

item-generator should be nil if *item* is specified. If *item-generator* is specified, it should have the form discussed in paragraph 16.4, Item Generators.

Because a text scroll window updates a display according to a fixed pattern, it is often useful for it to have an inferior that is a typeout window for the sake of occasional output that is not part of the standard display (such as output produced by pressing the HELP key in the Inspector).

w:text-scroll-window-typeout-mixin Flavor
Required flavor: **w:text-scroll-window**

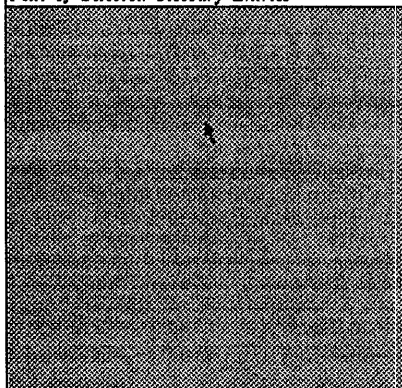
Can be added to a flavor containing **w:text-scroll-window** to provide a typeout window. The **w:text-scroll-window-typeout-mixin** flavor also arranges for proper interaction with the typeout window and partial redisplay over the area it uses.

:flush-typeout Method of **w:text-scroll-window-typeout-mixin**

If the typeout window is active, **:flush-typeout** deexposes it and ensures that redisplay knows the lines have been used by the typeout window.

w:text-scroll-window-empty-gray-hack Flavor
Required flavor: **w:text-scroll-window**

A mixin that goes with **w:text-scroll-window**. When windows of this type have an empty array for the **w:items** instance variable or an item generator indicating that the number of items is 0, the interior of the window becomes gray. This flavor is used in some panes of the window-based debugger frame and the glossary. For example, as shown in the following figure, the Text Pane of the Glossary is gray when the Glossary is first invoked, that is, until you display a definition.

Menu of Glossary Entries	Index	Text of Selected Glossary Entries
<i>a-list</i>	A	
<i>abnormal shutdown</i>	B	
<i>ABORT key</i>	C	
<i>access function</i>	D	
<i>access operation</i>	E	
<i>accessor function</i>	F	
<i>activation environment</i>	G	
<i>active element</i>	H	
<i>active process</i>	I	
<i>active window</i>	J	
<i>adjust</i>	K	
<i>advise</i>	L	
<i>after daemon</i>	M	
<i>alu function</i>	N	
<i>ancestor</i>		
<i>application</i>		
<i>apropos</i>		
<i>apropos completion</i>		
<i>area</i>		

Item Generators

16.4 The item generator feature allows the Inspector to scroll through the elements of a large array without having to create another equally large array of items. This feature generates the item description of the values to be displayed using a function called the *item generator function*.

:item-generator Method of **w:text-scroll-window**
Settable. Default: **nil**

Returns the item generator function, or **nil** if no item generator is in use. The item generator is a function that simulates the effect of an array of items. It overrides any explicit array of items; the value of **w:items** is still an array, but it does not affect the display.

The item generator function expects its first argument to be an item generator method keyword. Table 16-1 shows the defined keywords.

Table 16-1 Keywords for the Item Generator Function

Keyword	Description
:number-of-items	Returns the total number of items to scroll through (the equivalent of the fill pointer in an actual array of items).
:number-of-item <i>item</i>	Returns the index of the item specified by <i>item</i> . If an actual array were used, this would be the index in the array where <i>item</i> is found. The test used for comparison is eq .
:item-of-number <i>index</i>	Returns the item at the index specified by <i>index</i> . If an actual array were used, this would be the element located at the array position pointed to by <i>index</i> .
:insert-item <i>index item</i>	Inserts a new item before another item specified by <i>index</i> . <i>item</i> specifies the item to be inserted. If an actual array were used, this insertion would be performed by moving the elements down one, beginning with and including the element at <i>index</i> . The item generator needs to support this keyword only if you wish to use the :insert-item or :append-item methods on the window.
:delete-item <i>index</i>	Deletes the item at the index specified by <i>index</i> . The items following the item at <i>index</i> move up one position. The item generator needs to support this keyword only if you wish to use the :delete-item method on the window.

The Inspector uses an item generator to display the elements of an array so that it does not have to create another array of items as big as the array being displayed (that is, so the Inspector does not cons a very long list of consecutive integers for the array). If L is the length of the array's leader, then item numbers 0 through $L - 1$ correspond to the leader, and item number $L + I$ corresponds to array element I (a multidimensional array being treated as a one-dimensional array).

The value of the item at item number n is simply n . In other words, the virtual array of items that the item generator simulates is an array of consecutive integers, independent of the data being displayed. This may seem to be an unusual way of doing things, but consider this: the line for the I th element should not print out as simply that element. The line should contain the number I as well. So the item value is simply $L + I$, and the **:print-item** method is redefined to print such a number by printing I followed by the I th array element.

The following code is the item generator used by the Inspector. Note that the array whose elements are being displayed is found as (car w:print-function-arg), and (cadr w:print-function-arg) is non-nil if the leader should be displayed. The w:print-function-arg form is an instance variable from the flavor w:function-text-scroll-window:

```
(defun inspect-array-item-generator (msg &optional arg1)
  (declare (:self-flavor basic-inspect))
  (case msg
    (:number-of-items (+ (if (cadr print-function-arg)
                            (or (array-leader-length (car print-function-arg)) 0) 0)
                        (length (caddr print-function-arg))
                        (array-total-size (car print-function-arg))))
    (:number-of-item (if (numberp arg1) (+ arg1 (length (caddr print-function-arg)))
                        (position arg1 (the list (caddr print-function-arg)) :test #'eq)))
    (:item-of-number (if (< arg1 (length (caddr print-function-arg)))
                        (nth arg1 (caddr print-function-arg))
                        (- arg1 (length (caddr print-function-arg))))))
```

The :insert-item and :delete-item methods are not supported because the Inspector does not try to insert or delete items.

The Inspector uses w:function-text-scroll-window so that :print-object is handled by calling a dynamically changeable *print function*. The following example shows a version of the print function used by the Inspector when displaying an array:

```
(defun inspect-array-printer (item arg stream item-number &aux (obj (car arg)))
  (leader-length-to-mention (if (cadr arg) (array-leader-length obj) 0))
  "The print-function used when inspecting an array."
  ;; (CAR ARG) is the array. (CADR ARG) is T to display the leader.
  ;; ITEM is usually a number. A small number is an index in the leader.
  ;; Numbers too big for that start moving through the array elements.
  ;; Make sure base is consistent since sometimes this is called from the mouse process.
  (let ((*print-base* (send (send self :superior) :*inspect-print-base*)))
    (cond
      ((not (numberp item)) (inspect-printer item obj stream item-number))
      ((< item leader-length-to-mention)
       (let ((pntr (locf (array-leader obj) item)))
         (send stream :item1 item `leader-slot
                #'(lambda (item stream) (format stream "leader -d" item)))
         (format stream ":-12t ")
         (if (%p-contents-safe-p pntr)
             (send stream :item1 (array-leader obj) item) :value #'print-item-concisely)
             (format stream "#<-a -o>"
                    (or (nth (%p-data-type pntr) q-data-types)
                        (%p-data-type pntr))
                    (%p-pointer pntr))))))
      (t
       (let ((item (- item leader-length-to-mention))
             (rank (array-rank obj))
             (indices))
         (or (= rank 1) (setq indices (array-indices-from-index obj item)))
         (send stream :item1 (cons item (if (= rank 1) item indices)) `array-slot
                #'(lambda (datum stream) (format stream "elt -d" (cdr datum))))
         (format stream ":-9t ")
         (if
          (or (cdr (assoc (array-type obj) array-bits-per-element :test #'eq))
              (%p-contents-safe-p (ap-1-force obj) item))
            ;; Deal with data types that are objects, and with numeric arrays.
            (send stream :item1 (ar-1-force obj) item) :value #'print-item-concisely)
            ;; Deal with data types that aren't really objects.
            (format stream "#<-a -o>"
                    (or (nth (%p-data-type (ap-1-force obj) item)) q-data-types)
                    (%p-data-type (ap-1-force obj) item))
                    (%p-pointer (ap-1-force obj) item)))))))))
```


Mouse-Sensitive Text Scroll Windows

16.5 The following flavors, methods, and so forth, discuss ways to define and use mouse-sensitive items.

w:mouse-sensitive-text-scroll-window Flavor

Allows the lines to contain mouse-sensitive items exactly like those of **w:basic-mouse-sensitive-items**, though the implementation is different. You must create a **:print-item** method for any flavor you create using **w:mouse-sensitive-text-scroll-window**. The existing **:print-item** method puts the item only on the window; it does not put the item into the array of items that are currently displayed. This causes an item to appear after scrolling but not be selectable.

The following code handles simple cases:

```
(defmethod (your-flavor-name :print-item) (item ignore ignore)
  (let ((item-coming-into-window
        (list item `your-type cursor-x (w:sheet-inside-right))))
    (prin1 item self)
    (push item-coming-into-window
          (aref displayed-items (w:sheet-line-no)))))
```

Refer to the source code for the **w:displayed-items-text-scroll-window** flavor for more information on writing a suitable **:print-item** method.

:mouse-click *button x y* Method of **w:mouse-sensitive-text-scroll-window**

Examines the item clicked on.

- If the item includes an item type in its association list, the **:mouse-click** method causes the **:force-kbd-input** method to insert the list (*type item self* (**merge-shift-key** *button*)) into the input buffer; **:mouse-click** then returns *t*.
- If the item does *not* include an item type in its association list, **:mouse-click** returns the *x* and *y* coordinates of the item.

:item *type item &rest format-args* Method of **w:mouse-sensitive-text-scroll-window**

Can be used in the **:print-item** method to include mouse-sensitive items in the output.

item and *type* are not necessarily used in printing the item, but they are used in handling a click on the item. *type* is used to look up a function in the item-type association list, and *item* is placed directly into the **:typeout-execute** blip. This blip is handled automatically by the **:mouse-click** method. *format-args* represents any arguments that are used to format its output.

The **:item-list** and **:primitive-item** methods of the **w:basic-mouse-sensitive-items** flavor are not provided, because they are not really useful in this context.

:item1 *item type print-function &rest args* Method of **w:mouse-sensitive-text-scroll-window**

Enables output to include mouse-sensitive items. Unlike **:item**, **:item1** specifies how *item* is output by calling *print-function* using *item*, the window, and the elements of *args* as arguments. *item* and *type* have the same meanings as for the **:item** method.

In a typical `w:basic-mouse-sensitive-items` window, mouse-sensitive items are output on specific occasions, and only because they are supposed to be present and mouse-sensitive at that time. In a text scroll window, a single display is usually maintained at all times, but the parts that should be sensitive to the mouse may need to depend on other things. For example, in the Inspector, the values of components are sensitive, but when you are specifying a component to store into, the names of the components are sensitive instead.

:sensitive-item-types *alist* Initialization Option of
w:mouse-sensitive-text-scroll-window

Gettable, settable. Default: `t`

Controls which items are mouse-sensitive item types. A mouse-sensitive item is sensitive to the mouse if its type (as specified in the `:item` method) is a member of the list that is `w:sensitive-item-types`.

If `w:sensitive-item-types` is `t` (the default), all mouse-sensitive items actually are sensitive.

w:mouse-sensitive-text-scroll-window-without-click Flavor

A component of the `w:mouse-sensitive-text-scroll-window` flavor that provides everything but the `:mouse-click` method. Because this method uses the `:or` method combination, it is not possible to override a method once it is present.

:mouse-sensitive-item *x y* Method of
w:mouse-sensitive-text-scroll-window-without-click

Returns either a list describing the mouse-sensitive item found at cursor position *x*, *y* in the window, or `nil` if there is no item there.

The list has the following format:

(type item left top right bottom)

where:

type and *item* are as specified in the `:item` method.

left, *top*, *right*, and *bottom* are cursor position coordinates relative to the outside top left corner of the window.

The Inspector's `print` function discussed in the previous paragraphs performs its output using the `:item1` method so that the output becomes mouse-sensitive. The following example for the `cond` clause handles leader elements:

```
((< item leader-length-to-mention)
 (funcall window :item1 item 'leader-slot
  #'(lambda (item window)
    (format window "Leader ~D" item)))
 (format window "~12T ")
 (funcall window :item1 (array-leader array item)
  :value #'w:print-item-concisely))
```

In this example, `leader-slot` and `:value` are item types that the Inspector makes mouse-sensitive at various times.

Mouse Blips 16.5.1 When you click the mouse on a mouse-sensitive item, a blip is placed in the window's input buffer. The blip has the following form:

(type item window mouse-character)

where:

type is the item type, such as `leader-slot` or `:value`.

item is the actual item value specified in the `:item` or `:item1` method.

window is the text scroll window itself. (This is how the Inspector can tell which inspect pane you click on.)

mouse-character is a character whose mouse bit is 1. *mouse-character* tells the program which button was clicked.

w:line-area-text-scroll-mixin Flavor

Required flavors: `w:margin-region-mixin`, `w:text-scroll-window`

When added to the `w:text-scroll-window` flavor, this flavor creates a line area near the left edge where the mouse cursor changes to a right-pointing arrow, and a mouse click means something different than it normally does. The line area is an additional part of the left margin and does not overlap the space used for displaying the items.

A mouse click in the line area puts a blip into the input buffer that has the following format:

(:line-area item window button-mask)

where:

item is the actual item value specified in the `:item` or `:item1` method.

window is the text scroll window itself. (This is how the Inspector can tell which inspect pane you click on.)

button-mask is a mask of bits corresponding to mouse buttons. (See the `w:mouse-last-buttons` variable in paragraph 11.4.1, Button Masks, for information on how to interpret *button-mask*.)

:line-area-width *number* Initialization Option of `w:line-area-text-scroll-mixin`

Default: 24.

Initializes the width of the line area in pixels to *number*.

:line-area-mouse-documentation Method of `w:line-area-text-scroll-mixin`

Default: "Select a line."

Returns a string to display in the mouse documentation window while the mouse cursor is in the line area.

w:line-area-mouse-sensitive-text-scroll-mixin Flavor

Required flavor: `w:line-area-text-scroll-mixin`

Used instead of `w:line-area-text-scroll-mixin` if the `w:mouse-sensitive-text-scroll-window` flavor is used to create the window.

w:current-item-mixin Flavor
 When added to **w:line-area-text-scroll-mixin**, identifies one of the items with an arrow in the line area.

:current-item Method of **w:current-item-mixin**
Settable. Default: **nil**
 Returns the item to be marked with an arrow, or **nil** if there is no item. An arrow marks this item if it is on the screen, no matter where it is located.

The following flavors are part of the implementation of the **w:mouse-sensitive-text-scroll-window** flavor.

w:displayed-items-text-scroll-window Flavor
 Records additional information about the items that are displayed. This flavor provides an instance variable, **w:displayed-items**, which is an array indexed by line number. In this array, the **:print-item** method stores any relevant information about what was displayed on the line.

This flavor does not define the meaning of elements of the array. The **:print-item** method is responsible for storing whatever information is useful into the appropriate element of the array. The flavor, however, does move elements of the array when scrolling is performed, and sets them to **nil** when parts of the window are cleared or when they are about to be redisplayed.

The **w:displayed-items-text-scroll-window** flavor is essentially a subroutine of the **w:mouse-sensitive-text-scroll-window** flavor, which uses each element of **w:displayed-items** to hold information on the mouse-sensitive items for the line.

w:displayed-items Instance Variable of **w:displayed-items-text-scroll-window**
 The array of information about lines on the screen.

Inspector Flavors 16.6 The Inspector uses text scroll windows. The following flavor and resource are available to use to create an Inspector frame.

w:inspect-frame Flavor
 A self-contained Inspector with its own process to update the display.

w:inspect-frame-resource &optional (*superior w:mouse-sheet*) Resource
 A resource of Inspector frames that are created in a special way so that they do not have their own processes but instead are invoked in another process by the **inspect** function.

GENERAL SCROLL WINDOWS

Using General Scroll Windows

17.1 General scroll windows are used to exhibit a continuously maintained display of items, each of which can vary in size. Peek uses general scroll windows. General scroll windows (from now on called scroll windows) are not a generalization or a building block of text scroll windows, but rather an independent facility.

The scroll window's display is made up of items, which are not the same as items in text scroll windows. The same term is used because the item is a similar structure for these types of windows.

An *item* in a scroll window always occupies one or more entire lines. An item can be composed of subitems that are displayed vertically, each subitem occupying and filling up a certain number of lines. The subitems can, in turn, be composed of more items. New subitems can be dynamically added or deleted at any level, and the display is updated automatically to match the list of items by moving lines around on the screen.

Eventually, this process of subdivision comes to an end, with *lowest-level* items made up of *entries*, which are sequenced horizontally.

An entry displays a single string or quantity, updating its display if the value changes. The entry must record how to obtain a value to display, how to tell when the value has changed after the screen was updated, and how to output the new value. A single entry can wrap around at the right margin exactly like ordinary output. Entries can be added to and removed from an item dynamically.

In Peek's Active Processes display (shown on the following page), a single item displays the entire set of processes. It is composed of subitems, one for each process. If a new process appears, a new subitem is created to display it. The subitem for a single process is a lowest-level item. Each characteristic displayed about a process—its name, its run state, its priority, its percentage of use of the central processing unit (CPU)—is displayed by a single entry in that item.

The line of column headings at the top of the display is also a lowest-level item, and the column heading entries display constant strings.

Every character displayed on a scroll window comes from an entry. The items group entries and control the automatic insertion and deletion of entries.

Entries can be either of fixed or variable width. A variable-width entry takes up as much space as is needed to print its data; this spacing can change when the window is redisplayed. During redisplay, the remaining entries in the item must all move left or right. A fixed-width entry specifies an amount of horizontal space and always occupies that much space. As a result, it can be redisplayed without redisplaying the rest of the item afterward. The entries used in the Active Processes display are all fixed-width so that they line up in columns.

General Scroll Windows

Process Name	State	Priority	Quantum	%	Idle
Flavor Inspector 2	Keyboard	0.	60/60.	0.02	22 min
Inspect Frame 2	Keyboard	-1.	60/60.	0.02	27 min
Dormant FTP Connection GC	Sleep	0.	60/60.	0.02	26 min
Vt100 Frame 1-Typein	Keyboard	0.	60/60.	0.02	2 hr
Vt100 Frame 1-Typeout	Never-open	0.	60/60.	0.02	2 hr
File Server	Chaosnet Input	0.	60/60.	0.02	2 min
Glossary Frame 1	Keyboard	-1.	60/60.	0.02	19 hr
Profile Frame 1	Keyboard	-1.	60/60.	0.02	20 hr
Print Daemon	Queue Empty	-1.	60/60.	0.02	20 hr
Peek Frame 1	Run	0.	56/60.	15.02	
Zmacs Frame 1	Keyboard	-1.	60/60.	6.02	18 sec
Mail Daemon	Mailer Sleep	-5.	60/60.	0.02	6 min
TCP Background	Background Task	15.	60/60.	0.02	2 sec
TFTP Server	UDP Input	-5.	60/60.	0.02	22 hr
IP Packet Fragment timer	Sleep	-15.	60/60.	0.02	25 sec
ICMP Listener	ICMP Listen	15.	60/60.	0.02	1 min
Suggestions Frame 1	Arrest	0.	60/60.	0.02	20 hr
Pop-Up Keystrokes	Pop Up Keystrokes	0.	10/10.	0.02	20 hr
Hardware Monitor	Hardware Event Wait	-50.	60/60.	0.02	22 hr
Tn-Update	Sleep	-5.	60/60.	0.02	1 min
GC Daemon	GC Daemon	0.	60/60.	0.02	22 hr
GC-PROCESS	Stop	0.	60/60.	0.02	forever
Dormant FILE connection GC	Qfile gc sleep	-2.	60/60.	0.02	17 min
NUBUS Receiver, NMF0	Wait NuEther Input	25.	60/60.	7.02	
Chaos RUT transmitter	Stop	30.	0/60.	0.02	forever
Chaos Background	Background Task	25.	60/60.	0.02	7 sec
Screen Manager Background	Screen Manage	3.	60/60.	0.02	1 min
Mouse	MOUSE	30.	60/60.	3.02	2 sec
Keyboard	terminal-	30.	60/60.	0.02	
Initial Process	Keyboard	-1.	60/60.	0.02	1 hr
Page-Background	Sleep	-100.	60/60.	0.02	17 min
Clock Function List					
UPDATE-FUNCTION-HISTOGRAM					
BLINKER-CLOCK					
Peek Processes					
Modes		Processes		Commands	
Windows	Process	Servers	Counters	Documentation	Host Status
Areas	FILE STATUS	Network	Function Histogram	Get Timeout	Exit
List status of every process -- why waiting, how much run recently.					
Key assignments: p					

NOTE: The window redisplay algorithm is not designed to handle an entry that specifies a fixed width if the printing of the entry's contents exceeds that width.

The data structure that represents an item is either a list or an array. If it is a list, its cdr is a list of component items, and its car contains information on how to update the list (add or remove component items). The item is then displayed simply as the concatenation of its components. If it is an array, then it is a lowest-level, single-line item, and the elements of the array represent entries on the line. The array also has leader slots whose meanings are described in the following paragraphs.

Specifying Items and Entries

17.2 You do not generally create an array item or an entry yourself. They are made by calling the `w:scroll-parse-item` function, which is given a descriptive data structure made out of lists. Examples of its use are at the end of this section.

The arguments to `w:scroll-parse-item` are called *entry descriptors*, each of which specifies how to create one entry. The entries thus specified all go together into one list of items, called the root item, by using the following code fragment:

```
(list nil
  (w:scroll-parse-item entry-descriptor1 entry-descriptor2...))
```

The following are the possible kinds of entry descriptors. The arguments enclosed in brackets in the following descriptions (such as `[width]`) are optional arguments.

- A *string* or a list (`(:string string [width])`) — The entry is displayed by printing *string*. A string entry never varies because it always displays precisely the specified string and is always of fixed width. The width can be specified as a means of controlling the position of the following entry; otherwise, the actual width needed to print the string is the width of the item. For example, either `(:string "Foobar" 10.)` or `"Foobar "` specifies an entry that prints as `Foobar` followed by four spaces.
- A list (`(:symeval symbol [width-or-nil] [format-string])`) — The entry is displayed by printing the value of *symbol* by passing it to `format` together with *format-string*. If *format-string* is omitted, the value is printed with `princ`. This type of entry is automatically updated when the value of *symbol* changes. *width-or-nil* can be either a number of pixels to specify a fixed-width entry, or `nil` to specify a variable-width entry. For example, `(:symeval *print-base* nil "~ D. ")` specifies an entry that prints the value of `*print-base*` in decimal with a following period and a space before and after. It is of variable width, so the space it takes up is three plus however many digits are needed to print the value of `*print-base*`.
- A list (`(:function function list-of-args [width-or-nil] [format-string])`) — The value to display is obtained by applying *function* to *list-of-args*. If this value has changed since the last time it was checked, it is displayed by passing it to `format` together with *format-string*. If *format-string* is `nil`, the value is simply printed with `princ`. *width-or-nil* can be a number of pixels to specify a fixed-width entry, or `nil` to specify a variable-width entry. For example, the following code creates an entry descriptor that specifies an entry calling `sys:process-quantum-remaining` on some process and prints the result in decimal, followed by a slash, in a field five characters wide:

```
`(:function sys:process-quantum-remaining
  (,process) 5. ("~ 4D/"))
```

- A compiled function or certain interpreted functions — An entry descriptor is considered a function if it is either a compiled function (a functional entry frame, or FEF) or a list starting with `lambda` or `named-lambda`. It is treated as an abbreviation for `(:function function)`, which specifies no arguments, variable width, and no format string (the value is printed with `princ`).

- A list (`:value index [width-or-nil] [format-string]`) — The value to be displayed is found at *index* in the window's *value-array*.

Two other keywords can make the entry mouse-sensitive in an entry descriptor. The keywords can be used only in scroll windows that have `w:essential-scroll-mouse-mixin`. To use these keywords, you first construct an entry descriptor to specify how the entry should print, according to the preceding discussion of entry descriptors. Then you add one of these keywords and a value to go with it at the front of the list. The `:mouse` keyword makes the entry mouse-sensitive but has no effect on how the entry appears on the screen.

- The `:mouse` keyword is used in an entry descriptor that has the following format:

`(:mouse mouse-data . another-entry-descriptor)`

Such an entry descriptor is handled by creating an entry from *another-entry-descriptor* and then modifying it by recording *mouse-data* as the mouse sensitivity of the entry. The resulting entry prints according to *another-entry-descriptor* but is mouse-sensitive as well. *mouse-data* is a string of constant width. If neither a width for the item nor `nil` is specified, the default is `nil`, which causes the item to use however much space is required. If a formatted string is not supplied, the value is printed.

For example, the entry descriptor for Peek is as follows:

```
(" " (:MOUSE (NIL :EVAL (PEEK-WINDOW-MENU
                    (QUOTE #<PEEK-FRAME Peek Frame 1 3343152 deexposed>))
                    :DOCUMENTATION "Menu of useful things to do to this window.")
          :STRING "Peek Frame 1"))
```

- The `:mouse-item` keyword is used in an entry descriptor that has the following format:

`(:mouse-item mouse-data . another-entry-descriptor)`

`:mouse-item` is like `:mouse`, except that the `w:item` symbol is replaced throughout *mouse-data* with *item*, which is the item this entry becomes a part of. *mouse-data* should be a list. There is no way to cause the entry itself to be inserted into its own mouse-sensitive data because this is not useful when scroll windows are used in the intended manner.

`w:scroll-parse-item &rest keyword-args-and-entry-descriptors` Function

Creates and returns an array item containing entries constructed according to *keyword-args-and-entry-descriptors*.

The argument begins optionally with alternating keywords and values, which specify information that applies to the item as a whole. They are followed by entry descriptors, one for each entry you want in the item. Keywords and

entry descriptors are distinguished by the fact that an entry descriptor is never a symbol. The following keywords are defined:

Keyword	Description
<code>:mouse</code>	The value stored as the mouse sensitivity of the entire item. This value is meaningful only if the window flavor includes <code>w:essential-scroll-mouse-mixin</code> .
<code>:mouse-self</code>	The value stored as the mouse sensitivity of the entire item, but first the symbol <code>self</code> is replaced wherever it appears beside the item itself (the array that this function is constructing). This value is meaningful only if the window flavor includes <code>w:essential-scroll-mouse-mixin</code> .
<code>:leader</code>	Requests extra slots to be allocated in the array leader of the item array. <code>:leader</code> is either a number, the number of extra slots desired, or a list whose length is the number of extra slots and whose contents are used to initialize these slots.

`w:scroll-string-item-with-embedded-newlines` *string* Function
 Returns an item that displays the string specified by *string*. This item is composed of one item for each line making up *string*.

The following code is an example taken from Peek. The code creates the item for a process (the value of `process`) in Active Processes mode. The entries that use the process as a function work because the process is a flavor object; the argument given to the process is a flavor operation. Note that `w:peek-process-menu` is a function in Peek that asks for a choice with a pop-up menu:

```
(w:scroll-parse-item
  ;; The first entry is mouse-sensitive.
  `(:mouse-item
    (nil :eval (peek-process-menu `process `item 0)
      :documentation "Menu of useful things to do to this process.")
    :string,(process-name process) 30.)
  `(:function ,`peek-whostate ,(ncons process) 25.)
  `(:function ,process (:priority) 5. ("~D."))
  `(:function ,process (:quantum-remaining) 5. ("~4D/"))
  . . . more entries...)
```

Using a Scroll Window

17.3 The following flavors, methods, and initialization options allow a scroll window to be used.

`w:basic-scroll-window` Flavor
 The basis for all flavors of scroll windows; provides all the facilities specific to scroll windows. The flavor is not instantiable by itself.

`w:scroll-window` Flavor
 An instantiable scroll window flavor. This flavor provides scroll bars, margin scrolling, borders, and labels.

In addition to being able to create a tree of items and entries, you must tell the scroll window to display them. At the highest level, the entire display is grouped into a single item—the *root item*. Switching modes in Peek works by switching to a new root item.

:display-item *root-item* Initialization Option of **w:basic-scroll-window**
Gettable, settable. Default: nil

Sets the root item of the window that determines the window's display. The root item usually contains several subitems. Setting the root item redisplay the window. Typically, *root-item* is a value returned by the **w:scroll-parse-item** function.

:truncation *t-or-nil* Initialization Option of **w:basic-scroll-window**
Gettable, settable. Default: nil

Determines whether entries wrap around at the right margin. If **w:truncation** is nil, entries can wrap around at the right margin. Otherwise, each item can occupy only one line. Setting the flag redisplay the window.

A scroll window has a *value array* whose elements can be used to hold arbitrary data to be displayed by entries using the keyword **:value**. Such an entry specifies the index of a slot in the value array whose contents are the data to display. Putting appropriate data in the value array is up to you. One technique is to have an automatically updating item whose update function stores data into the value array and to have entries in the item use those value-array slots. There can be many such items, all using the same value-array slots.

:value-array *array-or-length* Initialization Option of **w:basic-scroll-window**
Gettable, settable.

Initializes the window's **w:value-array** instance variable to the value of *array-or-length*; that is, **w:value-array** specifies how many elements are in the array. If *array-or-length* is an array, then **:value-array** uses that array. If *array-or-length* is a number, then **:value-array** creates an array of that size.

The **:redisplay** method updates the display based on the current root item and automatically reprints the entries whose contents have changed. The window system executes **:redisplay** automatically at certain times (such as when the window size is changed or the screen is refreshed), but if you want this method to execute simply because some of the displayed data has changed, you must send a **:redisplay** message yourself.

The **:redisplay-selected-items** method is another way to request display updating. This method allows you to control which items are checked.

:redisplay &optional (*full-p* nil) (*force-p* nil) Method of **w:basic-scroll-window**
 Redisplay the contents of the scroll window.

Arguments: *full-p* — If the *full-p* argument is nil, the window assumes that its screen bits contain the result of the last redisplay that was performed, and only items and entries whose contents are different from those of last time are actually output. If *full-p* is non-nil, everything that is supposed to be on the screen is redrawn.

force-p — If the *force-p* argument is non-*nil*, **:redisplay** updates the contents of the window even if it is not exposed. Normally, **:redisplay** waits if the window is not exposed.

:redisplay-selected-items *list-of-items* Method of **w:basic-scroll-window**
 Redisplays the items in the *list-of-items* argument if they are present on the screen. Other items in the current item hierarchy are not considered for redisplay. Each element of *list-of-items* is an entry descriptor.

Because a scroll window shows a constantly updated display, it is often useful to have a typeout window in it for occasional output that is not part of the display usually shown.

w:scroll-window-with-typeout-mixin Flavor
 Required flavors: **w>window-with-typeout-mixin**, **w:basic-scroll-window**
 Should be used in addition to **w>window-with-typeout-mixin** on any scroll window that is to have a typeout window. The **w:scroll-window-with-typeout-mixin** flavor handles interfacing between typeout window output and redisplay of the scroll window.

w:scroll-window-with-typeout Flavor
 Produces a scroll window that has an inferior typeout window.

Inserting and Deleting Items

17.4 Scroll windows provide methods for replacing, inserting, and deleting items explicitly. Because the items form a hierarchy, the position at which to replace, insert, or delete the item must be specified as a list of numbers. For example, (1 3 0) as a position means item number 0 within item number 3 within item number 1 (within **w:display-item**). Specifying *nil* as a position refers to the root item itself.

These methods also update the window on the screen, as necessary.

:get-item *position* Method of **w:basic-scroll-window**
 Returns the item at the window location specified by *position*.

:set-item *position item* Method of **w:basic-scroll-window**
 Stores the value of *item* into the hierarchy at *position*.

:insert-item *position item* Method of **w:basic-scroll-window**
 Inserts *item* at *position* and before the item that was originally at *position*. In other words, the **:insert-item** method moves all the items in the window—beginning with the item at the position specified by the *position* argument—down one position, then inserts the value of the *item* argument at the position specified by the *position* argument.

:delete-item *position* Method of **w:basic-scroll-window**
 Deletes the item at *position* and moves the following item to that position. In other words, the **:delete-item** method deletes the item in the window position specified by the *position* argument, then moves the following items up so that the gap is closed.

Automatically Updating Items

17.5 Just as an entry automatically updates the value it displays, an item can automatically update the list of items it contains. For example, the Active Processes display contains one item that displays a list of all active processes. This item contains a list of component items, one item per process. Just before the displayed entries for each process are updated—if necessary—additional items should be created and inserted in the list if there are any newly active processes, and items should be removed if processes have become inactive.

The first element of an item that is a list is used to store the data of a property list for the item. The `:pre-process-function` and `:function` properties have the following standard meanings:

- `:pre-process-function` — A function called whenever it is time to display this item. The sole argument of `:pre-process-function` is the item itself. The function called by `:pre-process-function` can modify the item. The returned value is ignored.
- `:function` — A function to update an individual component of this item. The function specified by `:function` is called each time any component item is about to be displayed or used.

The arguments given to the function called by `:function` are the component item, the reverse of the *position* of that item—a list of integers—and the location of the property list of the containing item. (See the discussion of position in paragraph 17.4, Inserting and Deleting Items.) The third argument represents the same property list on which the `:function` property appears (this can be passed directly to `get`). To repeat, the second argument is the *reverse* of the position that would be passed to the `:get-item` operation or related operations.

The function called by `:function` should return an updated component item.

Other properties can be used for any purpose. Some of the commonly used preprocess functions use other properties for their internal state information and additional parameters.

<code>w:scroll-maintain-list</code>	<i>init-fun item-fun</i>	Function
	<i>&optional per-elt-fun stepper compact-p pre-proc-fun</i>	
<code>w:scroll-maintain-list-unordered</code>	<i>init-fun item-fun</i>	Function
	<i>&optional per-elt-fun stepper</i>	

Returns an item that maintains a list of component items, one for each element of a *driving list*—the list used to specify an item. The item updates automatically so that component items appear and disappear as elements of the driving list do. These functions are not useful for recursive applications.

`w:scroll-maintain-list-unordered` is similar to `w:scroll-maintain-list`. `w:scroll-maintain-list-unordered` always adds new component items at the front of the combined item, no matter where they appear in the driving list. Changes in the order of that list have no effect at all. This is why this function is called *unordered*.

Arguments: *init-fun* — A function of no arguments that returns the current value of the driving list.

item-fun — A function that, given an element of the list, returns a component item to use to display for that element. The item created starts out with no component items. The function specified by *item-fun* is called each time a new element appears in the driving list. The appropriate set of component items is created by adding them one by one in this way the first time the item is updated.

per-elt-fun — The `:function` property. This item works because it is given a suitable preprocess function. The other arguments to `w:scroll-maintain-list` are also stored on the property list of the item created.

stepper — Either a function to step through a list or `nil`. The function specified by *stepper* is called with one argument, a list, and returns three values:

1. The first element extracted from it.
2. A list of the remaining elements.
3. Non-`nil` to indicate that there are no more elements—a `nil` list is always recognized as being empty, regardless of the third value.

compact-p — If the *compact-p* argument is non-`nil`, then the `w:scroll-maintain-list` function recopies the list each time an element is inserted or deleted, so the list remains compact and localized.

pre-proc-fun — A function that executes before the function specified by *per-elt-fun*.

Normally, the value returned by the function specified by the *init-fun* argument is a list, with objects from which the items are made as its elements. However, it is possible to extract the objects in other ways. The function specified by *stepper* is first called with the value returned by *init-fun*. The first value goes—if it is new—to the *item-fun* argument; the second value is fed back to the function specified by the *stepper* argument unless either the value is `nil` or the third value is non-`nil`.

The following code shows an example of a *stepper* function that can step through the properties in a property list:

```
(defun plist-stepper (plist-tail)
  (values (car plist-tail) (caddr plist-tail)))
```

The following code shows how Peek, in Window Hierarchy mode, recursively creates a tree of automatically updating items:

```
;; Make an item to describe the entire window hierarchy.
(defun peek-window-hierarchy (ignore)
  (w:scroll-maintain-list
   ;; The init-fun. When called, it returns a current list of screens.
   #'(lambda () w:all-the-screens)
   ;; The item-fun, which makes an item for a screen.
   #'(lambda (screen)
       (list ()
             (w:scroll-parse-item
              (format nil "Screen ~A" screen))
             (peek-window-inferiors screen 2)
             (w:scroll-parse-item ""))))))
;; No per-elt-fun is needed. Also, the default stepper works
;; because our list really is a list.
```

```
;; Make an item to describe window and its inferiors.
;; Indent is an indentation to print with.
(defun peek-window-inferiors (window indent)
  (declare (special window indent))
  (w:scroll-maintain-list
   (closure `(window) #'(lambda () (w:sheet-inferiors window)))
   (closure `(indent)
            #'(lambda (this-window)
                ;; Make an item with two subitems.
                (list ()
                      ;; one for this window,
                      (w:scroll-parse-item
                       (format nil "-V@T" indent)
                       `(:mouse
                        (nil :eval (peek-window-menu ` ,this-window)
                          :documentation
                          "Menu of useful things to do to this window.")
                          :string ,(send this-window :name)))
                      ;; and one with subitems for its inferiors.
                      (peek-window-inferiors this-window
                                             (+ indent 4))))))))
```

The following code shows how Peek makes the item that displays a Chaosnet connection's packets. Instead of a list of packets, there is a chain of packets, with each packet pointing to the next one. An explicit *stepper* is therefore required. The `chaos:pkt-link` function, when given one packet, returns the next packet in the chain (or nil at the end):

```
(w:scroll-maintain-list
 `(lambda () (chaos:read-pkts ` ,conn))
 `(lambda (x)
   (peek-chaos-packet-item x ,(+ indent 2))
   nil
   #'(lambda (state)
       (values state (chaos:pkt-link state)
              (null (chaos:pkt-link state))))))
```

w:scroll-maintain-list-update-states *elements window &optional item* Function

Redisplays some of the component items of an item.

Arguments: *elements* — A list that specifies which component items to update. If the element of the driving list from which a component item was made is *memq* of *elements*, then the component item is updated.

window — The window being operated on.

item — An item of the type created by `w:scroll-maintain-list` or `w:scroll-maintain-list-unordered`.

Representation of Items

17.6 An item is either a list or an array. A list item contains other items, while an array item contains entries.

List items use the following functions:

Functions	Returned Value
<code>w:scroll-item-component-items</code> <i>item</i>	The list of component items of this item.
<code>w:scroll-item-plist</code> <i>item</i>	The contents of the property list of this item.

Array items use the following access functions, which refer to array leader slots (the array elements themselves hold the entries in the item):

Functions	Returned Value
<code>w:scroll-item-size</code> <i>item</i>	The number of entries in the item.
<code>w:scroll-item-mouse-items</code> <i>item</i>	A list of mouse-sensitive areas of entries in this item.
<code>w:scroll-item-line-sensitivity</code> <i>item</i>	What was specified for mouse sensitivity of the item as a whole (using the <code>:mouse</code> or keyword in <code>w:scroll-parse-item</code>).

`w:scroll-item-leader-offset` Variable

The number of standardly defined slots in an item's array leader. The slot with this or any higher number can be used by applications for their own purposes.

Entries are also arrays. They have many components, all managed internally, and users should not access them directly. Peek never needs to do so.

Mouse-Sensitive Scroll Windows

17.7 The `w:scroll-parse-item` function provides syntax, described previously, for associating mouse sensitivity to any item or entry. The mouse sensitivity is a list whose purpose is to identify which mouse-sensitive area was clicked on and to specify what to do when that happens.

If the car of the mouse sensitivity is `nil`, then the mouse sensitivity is interpreted as a menu item. When the mouse-sensitive area is clicked on, the `:execute` method executes the menu item, but this is done in the mouse process. Unfortunately, there is no way to avoid this convention because mouse clicks on scroll windows can happen at any time, and no other process can use the `:choose` method to process mouse clicks.

If the car of the mouse sensitivity is non-`nil`, a click is handled by putting a blip into the scroll window's input buffer. The blip has the following form:

(blip-type sensitivity window mouse-character)

where:

blip-type is the car of that list.

sensitivity is the mouse sensitivity list.

window is the scroll window itself.

mouse-character is a character such as `#\mouse-R-1` that indicates which button was clicked.

blip-type is extracted and put at the front so that programs using scroll windows can handle blips from many sources. By specifying the car of each mouse sensitivity, the program can distinguish these blips from blips coming from menus, typeout windows, and so on, and process each one in the correct fashion.

A scroll window often displays many similar items that describe different data objects. These items all have the same patterns of mouse sensitivity. By using the **:mouse-self** and **:mouse-item** keywords, the program can set the mouse sensitivity. The **:mouse-self** keyword sets the sensitivity for an item, and the **:mouse-item** keyword sets the sensitivity for an entry. These keywords insert the items themselves into the sensitivity in place of the symbols **self** or **w:item**, respectively.

w:essential-scroll-mouse-mixin Flavor

Gives a scroll window the ability to make items or entries mouse-sensitive.

w:scroll-mouse-mixin Flavor

In addition to the features provided by **w:essential-scroll-mouse-mixin**, **w:scroll-mouse-mixin** also defines the **:execute** method to allow the mouse to be used on general scroll windows.

Peek Flavor

17.8 The Peek utility uses general scroll windows. The following flavor is available to use to create a Peek frame.

tv:peek-frame Flavor

A self-contained Peek display with its own process to update the display.

MISCELLANEOUS FEATURES

Introduction

18.1 This section discusses miscellaneous features available in the window system, including notifications; creating, recording, and playing sound; specific types of windows (such as Lisp Listeners and Zmacs editors); the mouse documentation window and status line; the System menu; window resources; and functions to help you find existing windows.

Notifications

18.2 *Notifications* are asynchronous messages that come from something other than the selected window. For example, when an interactive message from another user comes in (which was sent with the `qsend` function), the message is printed as a notification. Notifications can also be specified as part of a window's deexposed typeout action, the action taken when a process tries to get input from a window whose input buffer is empty and which is neither selected nor exposed.

A deexposed typeout action of `:notify` means that input from the window should notify the user when it is deexposed with a message like `Process x wants typein`. (See paragraph 7.4.1, Deexposed Typeout Actions, for more information about the actions.) Sometimes a notification is printed out immediately, and sometimes a message appears in the mouse documentation window. The selected window controls where the notification is displayed.

Another way a window can handle a notification is to send the notification to another window. For example, editor windows (of the `zwei:zmacs-window-pane` flavor) ask the containing Zmacs frame to send the notification, and the Zmacs frame in turn asks the Echo Area window to handle the notification. The window then displays the notification if the notification fits.

`w:notify` *window-of-interest* *format-string* &rest *format-args* Function

`w:careful-notify` *window-of-interest* *careful-p* *format-string* &rest *format-args* Function

Makes a notification. Unlike `w:notify`, if *careful-p* of `w:careful-notify` is non-`nil` and the notification cannot be printed now because the windows are locked or because no window is selected, `w:careful-notify` returns a value immediately. The value returned is non-`nil` if the notification was printed successfully.

Arguments: *window-of-interest* — The window that appears when you move the mouse blinker into the notification window and click a mouse button. For example, a notification about a message from another user supplies the Converse window as this argument. This window can also be selected with the `TERM 0 S` keystroke sequence.

careful-p — Non-`nil` if care must be taken with regard to disturbing *window-of-interest*.

format-string — Controls how the text is printed. This argument specifies the field and data type as well as anything else that is printed (such as string constants) when the string is printed. This argument is passed to the `format` function to print the text of the notification.

format-args — The name of the variable to be printed. This argument is passed to the `format` function to print the text of the notification.

:print-notification *time string window-of-interest* Method of *windows*

Invoked by the system on the selected window to make a notification. In most cases, it is simpler to use the `w:notify` or the `w:careful-notify` functions rather than this method.

Arguments: *time* — The time mentioned in the notification. Typically, you should specify the value returned by the `time` function; however, any value in universal time format is acceptable.

string — The text to print.

window-of-interest — The window that appears when you move the mouse blinker into the notification window and click a mouse button. For example, a notification about a message from another user supplies the Converse window as this argument. You can also select this window using the TERM 0 S keystroke sequence.

w:print-notifications &optional (*from* 0) *to* Function

Prints on `*standard-output*` *from* number of notifications that have happened in this session, with the most recent first. Because the default for *from* is 0, the function prints all the messages if you do not specify otherwise. *to* specifies the number of the last message to print.

w:notification-mixin Flavor

Causes a selected window to handle notifications by printing them out on the window itself, if the window is big enough. Many types of Lisp Listeners and typeout windows use this mixin. Window flavors incorporating `w:notification-mixin` sometimes redefine `:print-notification-on-self` to meet specific requirements.

:print-notification-on-self *time string window-of-interest* Method of
`w:notification-mixin`

Does the actual work of printing a notification on the window, once the window has been definitely chosen to do so. Window flavors incorporating `w:notification-mixin` sometimes redefine `:print-notification-on-self` to meet specific requirements.

Arguments: *time* — The time mentioned in the notification.

string — The text to print.

window-of-interest — The window that appears when you move the mouse blinker into the notification window and click a mouse button. For example, a notification about a message from another user supplies the Converse window as this argument. You can also select this window using the TERM 0 S keystroke sequence.

w:delay-notification-mixin Flavor

Implements the default procedure for handling notifications, which depends on the window's deexposed typeout action (discussed in paragraph 7.4.1). This flavor is a component of `w>window`, as well as of anything that contains `w:select-mixin`. The `w:notification-mixin` flavor works by overriding `w:delay-notification-mixin`.

If a notification arrives while a window of the **w:delay-notification-mixin** flavor type is selected, the notification is put on a list called **w:pending-notifications**. The only action that happens immediately is a beep. The presence of a non-**nil** value for **w:pending-notifications**, however, causes the mouse documentation window to display a message that notifications are waiting, with blinking asterisks at each end of the documentation window.

As soon as a window that can print the notifications is selected, the notifications are printed. For example, a Lisp Listener can print notifications. If you are in the editor, selecting the typeout window by pressing the **BREAK** key causes the notifications to be printed.

Also, the **TERM N** keystroke sequence selects a window that prints the notifications. Alternatively, the **TERM 2 N** keystroke sequence can return the mouse documentation window to its normal function. The **TERM 2 N** keystroke sequence works by transferring everything on **w:pending-notifications** to the **w:deferred-notifications** list. These deferred notifications are still printed when you switch to a suitable window.

w:pending-notifications Variable

A list of notifications that are waiting for the user to switch windows or press **TERM N** before the system displays them. When **w:pending-notifications** is non-**nil**, the mouse documentation window announces notifications with a message.

w:deferred-notifications Variable

A list similar to **w:pending-notifications**. Unlike **w:pending-notifications**, when **w:deferred-notifications** becomes non-**nil**, the mouse documentation window does *not* display a message.

w:find-process-in-error Function

Returns a process that, having made a notification, has an error and is waiting for a window to be selected so that the debugger can be run. If no such process is waiting, **w:find-process-in-error** returns **nil**. If there are several such processes, the most recent one to make its notification is returned. The second value returned is the window for which the process is waiting.

w:choose-process-in-error Function

Similar to **w:find-process-in-error**, but asks the user about each process that is in error and is waiting to be selected so that the debugger can be run on the chosen process. When the user types an upper- or lowercase **y** in response to a particular prompt, the process specified by that prompt is returned. If the user types an upper- or lowercase **n** in response to each prompt, **w:choose-process-in-error** returns **nil**. The second value returned is the window for which the process is waiting.

:notice eventMethod of *windows*

Reports certain events so that flavors can redefine what to do when the events occur. **:notice** uses the **:or** method combination—all the methods are run until one method returns non-nil. Using the **:or** method allows you to redefine the **:notice** effect more easily.

event is an event name—a keyword. Additional arguments are allowed but have no meaning for any of the events currently defined. The defined events can be **:input**, **:output**, **:input-wait**, and **:error**:

- **:input** and **:output** — The window is being used for input (output) and is not exposed, and its deexposed input (output) action is **:notify**. The default action makes a notification and waits.
- **:input-wait** — The window is being used for input, and the process is waiting because no input is available now. The default action adjusts the vertical position at which the next ****MORE**** happens.
- **:error** — The window is being used for the debugger and is not exposed. The default action makes a notification and waits, or it gets another window if this one is too small.

Creating and Recording Sounds

18.3 The Explorer system can create many different kinds of sounds. The **beep** function and method are a simple interface to the kinds of sounds created by the system with no additional equipment. You can also program sounds to be generated. If you add a microphone to the Explorer system, however, you can record and playback any sound.

All the sounds produced by the Explorer system are created by the sound chip on the Explorer System interface board. Technical details about the chip that generates these sounds are documented in the **SYS:WINDOW;SOUND** file.

Beeps **18.3.1** Beeps on the Explorer system are implemented by the **:beep** method on a window stream. There is also an easy-to-use **beep** function. The window system defines several varieties of sounds that can be used as beeps.

:beep & optional *beep-type*Method of *windows*Default: **w:default-beep**

Attempts to attract the user's attention by either making a sound or flashing the video display into and out of reverse video, or both. *beep-type* indicates why the beep is being done. For example, **telnet:terminal-beep** indicates that the VT100™ emulator's beep is being done. *beep-type* can either be specified explicitly with the **:beep** message, or it can default to the value of the **w:beep** variable. The following are possible values for *beep-type*:

Value	Action
nil , :silent	Does nothing—neither beeps nor flashes
:flash	Flashes the video display but makes no sound
:beep	Uses the value of w:default-beep
t	Uses the value of w:default-beep and flashes

VT100 is a trademark of Digital Equipment Corporation.

- w:beep** Variable
 The action taken by the **:beep** method. The **w:beep** variable can have any of the values described with the **:beep** method.
- beep** &optional (*beep-type* nil) (*stream* *standard-output*) Function
 Sends a **:beep** message to a window. If the window does not handle the **:beep** method, a beep sounds, regardless of what action is specified by the **:beep** method. *beep-type* can be an element of either **w:*beeping-functions*** or of **w:*beep-types***.
- w:flash-duration** Variable
 Default: 100,000 microseconds
 The interval in microseconds between a double complement of the video display.
- w:*beeping-functions*** Variable
 All the available beeping functions. These include, but are not limited to, the following:
- | | | |
|-------------------|-----------------------|--------------------|
| :beep | :flash | :shoop |
| :beep-high | :flying-saucer | :warm-up |
| :beep-low | :missile | :whoop |
| :bomb-drop | :pip | :vt100-beep |
| :chime | :races | :zowie |
| :doorbell | :random-beep | |
- You can try the various beeps by executing the **tv:sound-menu** function. **tv:sound-menu** pops up a menu that lists the beeping functions. When you select one of the functions, the system sounds that beep.
- w:def-beep-function** *name function . args* Macro
 Defines a new beeping function and adds it to **w:*beeping-functions***. When (**beep name**) is called, the system calls *function* with *args*.
- w:remove-beep-function** *beep-function* Macro
 Removes *beep-function* from the list in **w:*beeping-functions***.
- w:*beep-types*** Variable
 All valid beep types. These include, but are not limited to, the following:
- | | |
|------------------------------|---------------------------------------|
| telnet:terminal-beep | w:default-beep |
| w:notify | zwei:no-completion |
| zwei:converse-problem | zwei:converse-message-received |
- w:def-beep-type** *beep-type sound-name* Macro
 Defines a new beep type and adds it to **w:*beep-types***. When (**beep beep-type**) is called, the system executes (**beep sound-name**). Thus, when you use the **beep** function within an application, you can make the call to **beep** using a parameter that describes why the system is beeping. *sound-name* must be a member of **w:*beeping-functions***.

For example, the following code defines the `doorbell` beeping function as the Converse notification, then illustrates how that beep is called in code.

```
(w:def-beep-type 'zwei:converse-message-received :doorbell)
(beep 'zwei:converse-message-received)
```

- w:remove-beep-type** *beep-type* Macro
Removes *beep-type* from the list in **w:*beep-types***.
- w:default-beep** *wavelength duration* Function
Sounds a single-tone beep. *wavelength* is in hertz; *duration* is in microseconds.
- w:default-beep** Variable
The default beep that sounds when the **w:beep** variable is **t** or **:beep**. Its value can be any of the beeping functions.
- w:defsound** *name &body body* Macro
Defines a new sound so that executing (**beep name**) reproduces the sounds created in *body*. *body* is executed at macro expansion time so that when a file containing several **w:defsound** macros is loaded, the code to produce the file does not have to be loaded.

Making Sounds **18.3.2** You can program the Explorer system to make a number of sounds. In general, to create sounds by programming, do the following:

1. Enable the sound chip. Typically, you surround the remaining code in a **w:with-sound-enabled** macro.
2. Set the volume of the sound chip to an audible level.
3. Issue commands to the sound chip to wait or to produce sounds.

In most cases, both steps 2 and 3 are accomplished by using a low-level sound function to create the command byte and then sending the command byte to the chip using the **w:do-sound** function. For example, the following code implements the `whoop` beep.

```
(defun whoop (&optional (wavelength 600.) (increment 70.)
             (interval-time 7.))
  (with-sound-enabled
    (do-sound (volume 0 :on))
    (dotimes (i 8)
      (do-sound (tone 0 wavelength) interval-time)
      (incf wavelength increment))))
```

- w:reset-sound** Function
Resets the sound chip so that, should the monitor lose synchronization over the fiber optic cable, the monitor does not create an annoying tone. In other words, **w:reset-sound** either enables or disables the creation of sound.
- w:with-sound-enabled** *&body body* Macro
Executes *body* with sound enabled. This macro automatically executes **w:reset-sound** before and after it executes *body*.

w:do-sound *byte* &optional (*delay* 0) Function

Sends the command byte to the sound chip. This function does no error checking and returns nothing. *byte* is a single command byte, usually the result of **w:noise**, **w:tone**, **w:volume**, or **w:wait**. *delay* is the delay in milliseconds.

```
(w:with-sound-enabled
  (w:do-sound (w:volume 0 :on))
  (dotimes (i 100)           ; Repeat the tone 100 times to produce
    (w:do-sound (w:tone 0 500)) ; a long-sounding tone.
  ))
```

The following functions are low-level sound functions that return one or two command bytes suitable for passing to the sound chip via **w:do-sound**.

w:noise *noise-type* *hiss* Function

Returns a single command byte to control the noise type and hiss created by the sound chip. Unlike the command bytes produced by other low-level sound functions, this command byte turns the volume *off*, so you must reset the volume after you execute this command.

noise-type can be either 0 (:periodic noise) or 1 (:white noise). *hiss* can be one of the following:

Value	Description
0	Creates a higher frequency with the least coarse noise
1	Creates a medium frequency
2	Creates a lower frequency, most coarse noise
3 or :override	Causes register #2 (tone generator 3) to control the noise frequency

w:tone *reg* *freq* Function

Returns the command bytes needed to sound a tone of the specified frequency. *freq* is the frequency in hertz, which is clipped to the nearest frequency that can be produced. *reg* is an integer between 0 and 3, inclusive, that refers to one of the registers on the chip. Register 3 controls **:noise**.

w:tone-frequency *tone* Function

Given the results from *tone*, returns the frequency in hertz that *tone* produces. This is useful because the sound chip only accepts a limited range of values; using this function, you can determine the exact frequencies that the chip generates.

```
(w:tone-frequency (w:tone 1 500)) ==> 501
(w:tone-frequency (w:tone 1 1)) ==> 62
```

w:volume *reg* *volume* Function

Returns a single command byte that controls the volume of the sound. *reg* can be an integer between 0 and 3 inclusive. Register 3 controls the noise (that is, has the value of **:noise**). *volume* can be an integer between 0 and 15 inclusive, with a value of 2 equal to **:on** and a value of 15 equal to **:off**.

w:on-volume

Variable

The default volume level for the sound chip.

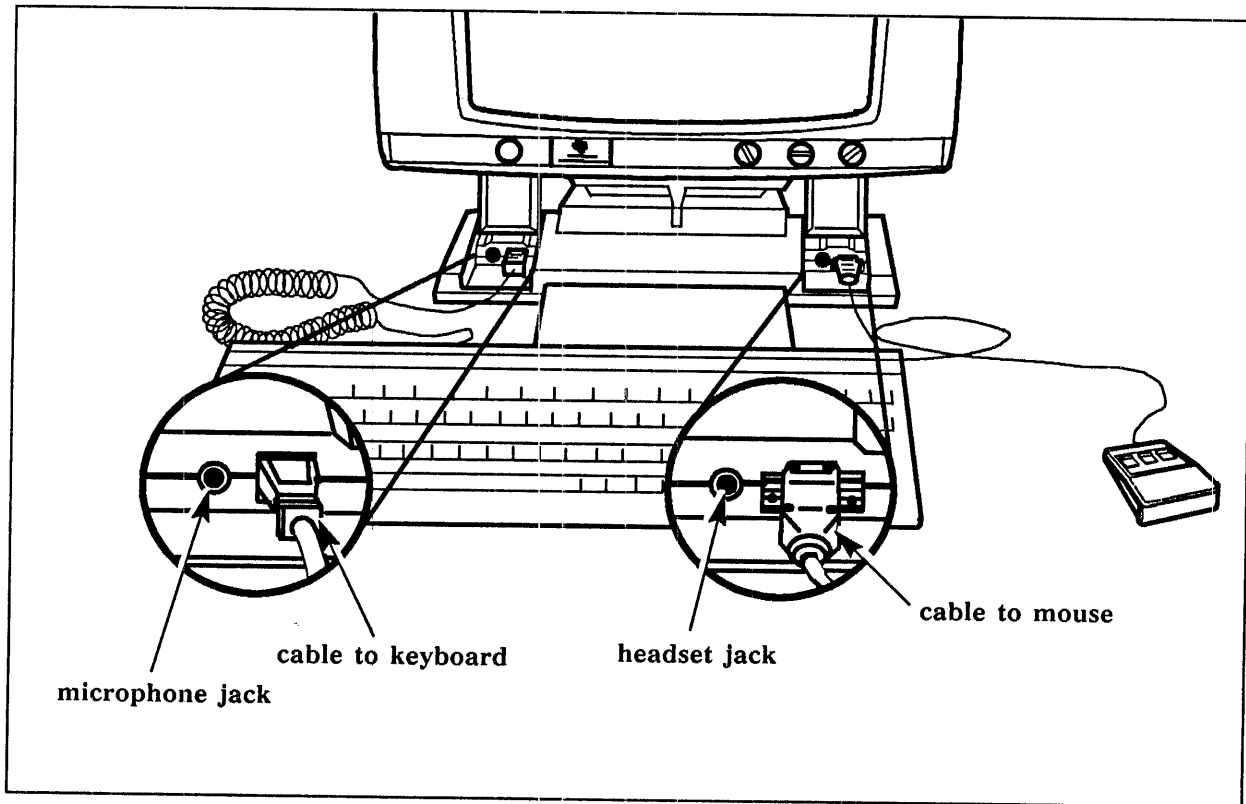
w:wait *ms-time*

Function

Returns a single command byte that causes **w:do-sound** to wait *ms-time* milliseconds.

Recording and Playing Sounds

18.3.3 You can record sounds with your Explorer system by installing a microphone. You should use a low-impedance microphone (150 ohm) with a standard 1/4-inch phone jack. Plug this microphone into the standard jack to the left of the monitor base, next to the cable to the keyboard. (Note that on a color system, the cable to the keyboard is left of the microphone jack.)



The functions that play record sounds access the analog-to-digital and the digital-to-analog converter chips on the SIB board rather than using the sound chips as **beep** does. The general interface to this feature is the **w:rec** function; there are functions that are analogous to each of the menu items offered by the **w:rec** function.

The following example shows how you would use the functions to save and replay a sample of speech.

```
;; Be sure the microphone is plugged into the Explorer microphone jack.
;; Hold down the r key, say "hello" into the microphone, then press END.
(w:rec 'hello)

;; Play back the "hello" sound
(w:play 'hello)
```



```
;; Saves the speech to disk
(w:save-speech "lm:me;hello" ^hello)

;; Define a beep-type for hello
(w:def-beep-function :hello w:play hello "lm:me;hello")

;; Play back the "hello" sound. If necessary, this function loads the speech
;; from the file "lm:me;hello.speech".
(beep :hello)

;; When a converse message is received, the system says "Hello".
(w:def-beep-type zwei:converse-message-received :hello)
```

w:rec &optional *ref-symbol* (*loud t*) (*max-length-in-seconds 30*) Function

Presents a menu that enables the user to choose from **:record**, **:play**, **:loop-back**, **:show**, and **exit**.

ref-symbol is a user-specified symbol (usually a keyword) that you can use to reference the sound later. If *loud* is **t**, the system increases the sensitivity of the microphone. *max-length-in-seconds* specifies the maximum number of seconds to record and thus also specifies the size of the array needed to store the sound.

w:play &optional *ref-symbol* *pathname* Function

Plays the speech in *ref-symbol*. If *ref-symbol* has no speech defined for it, this function loads the speech sample from *pathname* into *ref-symbol*.

If *ref-symbol* is **nil** and a *pathname* is defined, this function reads and plays the speech without placing the speech in virtual memory. This operation is useful for sounds that are played infrequently.

w:save-speech *pathname* &optional *ref-symbol* Function

Saves a sample of speech in a file specified by *pathname*.

w:read-speech *pathname* &optional *ref-symbol* Function

Restores a sample of speech from the file specified by *pathname* and saves it in *ref-symbol*. By default, *ref-symbol* is the filename in *pathname*. *ref-symbol* is interned in the current package.

w:rename-speech *old-name* *new-name* Function

Renames a sample of speech from *old-name* to *new-name*.

w:copy-speech *from-name* *to-name* Function

Copies a sample of speech from *from-name* to *to-name*.

Specific Types of Windows

18.4 The following flavors create special windows, including Lisp Listeners, Zmacs editor windows and frames, and various special-purpose windows.

Lisp Listeners 18.4.1
w:lisp-listener

Flavor

The flavor initially selected when the system starts up. **w:lisp-listener** creates a Lisp Listener window when you ask for a Lisp Listener window with any of the System menu commands.

w:lisp-interactor

Flavor

Works exactly like a Lisp Listener, except that the SYSTEM L keystroke sequence cannot select this kind of window, nor can **w:idle-lisp-listener** return one.

w:initial-lisp-listener

Variable

The Lisp Listener window that is selected when you boot the system.

w:idle-lisp-listener &optional (*screen* **w:default-screen**)
(*size-not-important* nil)

Function

Returns a Lisp Listener that is waiting for input at the top level and is the same size as the default screen. If no such window exists, then **w:idle-lisp-listener** creates the window. If *size-not-important* is **t**, **w:idle-lisp-listener** verifies that the size of the Lisp Listener and the inside size of *screen* are equal and creates a new Listener if the existing Listener is not the correct size.

w:listener-mixin-internal

Flavor

Sets the default for the window to run the Lisp top-level read-eval-print loop **sys:lisp-top-level1**; this allows the window to function as a Lisp Listener. That is, when you type a closing parenthesis, the window evaluates the expression within the enclosing parentheses. **w:listener-mixin-internal** contains **w:process-mixin** and is primarily responsible for making a Lisp Listener behave the way it does.

w:listener-mixin

Flavor

Inherits its entire definition from **w:listener-mixin-internal**. The only difference is that the SYSTEM L keystroke sequence is defined to look for windows of flavor **w:listener-mixin**, and not with the **w:listener-mixin-internal** flavor.

:package
Method of **w:listener-mixin-internal**

Retrieves the package being used by the read-eval-print loop; that is, the Lisp Listener reads data in, evaluates it, and prints out the results. (See the variable in the *Explorer Lisp Reference* manual for more information about the read-eval-print loop.) The returned value can be **nil**.

:set-package *package*
Method of **w:listener-mixin-internal**

Sets the package being used by the read-eval-print loop. *package* can be either a package or a package name.

Editor Windows 18.4.2 A Zmacs frame is useful for providing the user with an opportunity to edit whatever he or she likes. It is sometimes useful for a program, for its own purposes, to offer the user specific text to edit. For example, both the bug report utility and the graphics editor offer the user text to edit based on the user's actions.

zwei:zmacs-frame Flavor

The flavor of the window displayed when you press the SYSTEM E keystroke sequence. **zwei:zmacs-frame** has its own process and can select any Zmacs buffer. Editor-specific methods usually should not be invoked on this window; such operations should be left up to the window's own process. Requests to this process, which generally ask the process to select a buffer, are passed to it as blips of the following form:

(:execute zwei:edit-thing *spec*)

where *spec* is anything valid as the argument to the `ed` function that invokes the Zmacs editor.

zwei:standalone-editor-window Flavor

Produces a window without panes that serves as an editor. This flavor has a minibuffer and a type-in window that pop up as its inferiors when they are needed. This window has no process of its own; you should use the `:edit` method in any process to perform editing in the window.

zwei:standalone-editor-frame Flavor

Another kind of standalone editor window, but this window is a frame with a permanently visible mode line and type-in window or minibuffer, just as a Zmacs frame is.

:comtab *comtab* Initialization Option of *standalone editor windows*
Default: **zwei:*standalone-comtab***

Specifies the command table to use in editing in this frame. *comtab* is the Zmacs command table.

:edit Method of *standalone editor windows*

Invokes the editor command loop on this window. Pressing the END key returns control to the caller.

Recall that an *interval* in a Zmacs buffer is the text between point and mark—for example, the text region that is marked for highlighting. This text typically is bound to a variable. For more information about Zmacs, see the *Explorer Zmacs Editor Reference* manual.

:interval-string Method of *editor windows*

Returns a string giving the current text in the window.

:set-interval-string *string* Method of *editor windows*

Sets the text in the window to the value specified by the *string* argument.

:interval Method of *editor windows*
 Returns the interval being edited in the window. If the window is a Zmacs frame, this window is the selected buffer. Standalone editor windows have their own nonshared intervals, which they edit. Many of the editor primitives that work on Zmacs buffers also work on these intervals.

:set-interval *interval* Method of *editor windows*
 Sets the interval that this window is displaying and editing to *interval*. On a Zmacs window, *interval* must be a Zmacs buffer; **:set-interval** actually tells the window to select the new buffer.

zwei:pop-up-standalone-editor-frame Flavor
zwei:pop-up-standalone-editor-frame Resource
 &optional (*superior w:mouse-sheet*)

Produces a temporary window in the form produced by the **zwei:standalone-editor-frame** flavor. The resource is a resource of standalone windows and is used by the **zwei:pop-up-edstring** function.

zwei:pop-up-edstring *string* &optional (*near-mode '(:mouse)*) Function
mode-line-list min-width min-height initial-message
*(comtab zwei:*standalone-comtab*)*

Produces a pop-up editor window containing a string to be edited.

Arguments: *string* — The string popped up in an editor window so that the user can edit it. When the user presses the END key, the function returns a string containing whatever the user left in the editor buffer. If the user presses ABORT, the value is nil.

near-mode — How to position the window before popping it up. It is passed to **w:expose-window-near**. See the **:expose-near** method, described in paragraph 5.7.3, Symbols That Manipulate Screen Arrays and Exposure, for more details.

mode-line-list — A list of things to be displayed in the mode line. This argument can have more than one element, and the elements can be a combination of strings, symbols, lists, or lists starting with **:right-flush:**

- A string — The string is displayed.
- A symbol and the symbol is non-nil — The value of the symbol is printed. If the symbol is nil, nothing is printed for this element.
- A list — The list is either of the form:

(atom strings-or-symbols-1)

or is of the form:

(atom strings-or-symbols-1 :else strings-or-symbols-2)

where:

atom is an atom to be evaluated.

strings-or-symbols-1 and *strings-or-symbols-2* constitute a series of strings or symbols (or both) handled as described previously for strings or symbols. This element of the *mode-line-list* argument evaluates the atom first. If the atom is non-**nil**, then *strings-or-symbols-1* is processed. If the atom is **nil**, the *strings-or-symbols-1* is skipped and the *strings-or-symbols-2* following the **:else** (if an **:else** exists) is processed.

- A list starting with **:right-flush** — The second element of the list must be a string. The **:right-flush** keyword causes the string to be displayed flush against the right margin.

min-width, *min-height* — Pixel minimums for the size of the window. The window is larger than these minimums if *string* requires more space to display.

initial-message — If non-**nil**, is displayed in the type-in window immediately after the frame pops up.

comtab — The command table to be used for editing.

zwei:temporary-mode-line-window-with-borders Flavor
zwei:temporary-mode-line-window-with-borders-resource Resource
 &optional (*superior w:mouse-sheet*)

The temporary mode line window in Zmacs that contains only a mode line and a minibuffer. The flavor allows for a program to request a small piece of input while allowing the user to edit with the Zmacs. The following function uses this resource.

zwei:read-defaulted-pathname-near-window *window prompt* Function
 &optional (*defaults zwei:pathname-defaults special-type*)

Pops up a temporary mode line window near a window. The window does not need to be an editor window. This function, as used in the constraint frame editor, appears similar to the following:

```
WRITE TO BUFFER: (Default is Lima: WEBB; *BUFFER-1*.LISP#) (Completion)
w-test-buffer
```

Arguments: *window* — The window next to which the function pops up a temporary mode line. *window* can be any window on the screen, or **:mouse**, causing the mode line window to pop up near the mouse.

prompt — Specifies the mode line and lets the user edit text that (when the user presses the RETURN key) is parsed into a pathname using *defaults* and *special-type*. For example, a typical prompt is the text **I-search:** that is displayed when a user requests an incremental search by pressing CTRL-S.

defaults — A string specifying defaults for the pathname. The default for this argument is the result returned by the **zwei:pathname-defaults** function.

special-type — A string specifying the file type for the pathname.

:call-mini-buffer-near-window *window function &rest args* Method of
zwei:temporary-mode-line-window-with-borders

Pops up this minibuffer near another window, then uses a function to read the input and returns the value the function returns.

Arguments:

- window* — The window that this minibuffer pops up near.
- function* — An editor function that invokes the minibuffer using **zwei:edit-in-mini-buffer**. The first argument to *function* is a stream reading from the text the user edited.
- args* — A list of arguments passed to *function* as additional arguments.

Window Flavors for Other Programs 18.4.3

supdup:telnet Flavor

A self-contained Telnet window with its own pair of processes to transfer data to and from the network.

supdup:telnet-windows &optional (*superior w:mouse-sheet*) Resource

A resource of Telnet windows for use by the **telnet** function when operating in the mode of substituting for another window.

w:pop-up-text-window Flavor

w:pop-up-text-window &optional (*superior w:mouse-sheet*) Resource

Similar to **w:window**, except that **w:pop-up-text-window** is a temporary window with shadow borders.

w:truncating-pop-up-text-window Flavor

Similar to **w:window** except that **w:truncating-pop-up-text-window** is a temporary window that truncates lines of output.

w:truncating-pop-up-text-window-with-reset Flavor

w:pop-up-finger-window &optional (*superior w:mouse-sheet*) Resource

Similar to **w:pop-up-text-window** but truncates lines and resets the associated process when deexposed. The TERM F keystroke sequence uses **w:truncating-pop-up-text-window-with-reset** windows for output; this flavor is useful for many similar applications.

The Who-Line

18.5 The *who-line* is the historic term for the screen that includes the *mouse documentation window* and the status line. It is located at the bottom of the main Explorer video display and shows the current status of the machine. The mouse documentation window displays documentation showing what operations mouse clicks can now invoke, based on the actual position of the mouse. The status line, the bottom line of the video display and of the who-line screen, displays the time, your login name, the current process package and process run state, and file or network server information.

```

L: This line to top      LH: Continuous next line    L2: Next page      M: To fraction of buffer  MH: Drag lines
R: Top line to here    RH: Continuous previous line  R2: Previous page.  Bump top or bottom for single line scrolling.
03/19/87 10:34:33AM WEBB  USER: Keyboard      + Line: WEBB; IMAGE.XLDR3 0

```

The window system treats the who-line as a separate screen, preventing windows on the rest of the screen (called the *main screen*) from being moved or reshaped to overlap it. The mouse documentation window is displayed by a window of its own, as is each field of the status line.

Mouse Documentation Window

18.5.1 The documentation displayed by the mouse documentation window is obtained by sending the window under the mouse blinker a `:who-line-documentation-string` message or is obtained from the `w:who-line-mouse-grabbed-documentation` variable when the mouse is grabbed.

The following functions manipulate the mouse documentation window. In addition to these functions, if you are running in a color environment, you can also set the foreground and background color of the mouse documentation window as described in paragraph 19.7, Profile Variables for Color.

w:process-who-line-documentation-list *who-sheet new-state* Function

Processes information for the mouse documentation window from *new-state*, a list of keyword-value pairs. This function can accept any keyword accepted by the `:who-line-documentation-string` method of windows described in paragraph 11.6, How Windows Handle the Mouse.

The exact appearance of the display depends on the number of lines in the mouse documentation window.

w:number-of-who-line-documentation-lines Variable
Default: 2.

The number of lines in the mouse documentation window.

w:set-number-of-who-line-documentation-lines Function
&optional (*who-line-doc-lines* 2)
(*who-line-doc-font* (list fonts:h112b)) (*who-vsp* 3)

Changes the number of lines in the mouse documentation window. This involves setting the value of `w:number-of-who-line-documentation-lines` and redisplaying the mouse documentation window screen and the main screen. The total height of this window is limited to half the physical screen height.

Arguments: *who-line-doc-lines* — The height, in lines, of the mouse documentation window.

who-line-doc-font — The fonts displayed in the mouse documentation window. *who-line-doc-font* can be a single font, a list of font names (which is put into the font map), or an array that is the new font map.

who-vsp — The vertical spacing, in pixels, between the lines in the mouse documentation window.

Status Line 18.5.2 The status line includes separate windows for the time, the current process, file streams, and servers. The following paragraphs describe the functions, methods, and variables that deal with the various windows in the status line. Also, if you are running in a color environment, you can change the foreground and background colors of the status line by using the Profile variables described in paragraph 19.7, Profile Variables for Color.

Time and Date 18.5.2.1

w:12-hour-clock-setup Function
w:24-hour-clock-setup Function

Changes the clock that appears in the status line to display time in the 12-hour mode followed by AM and PM or in the 24-hour mode, respectively.

```
; 12-hour-mode          24-hour-mode
06/10/86 06:36:01PM     06/10/86 18:36:01
```

Current Process 18.5.2.2 The package name and run state displayed in the status line describe only one process. They normally describe the process associated with the selected window, which is a different process if a new window is selected. However, the status line can be locked on a particular process, independent of the selected window.

w:who-line-process Variable

Either the process that describes the mouse documentation window, or `nil`. This variable is used to display the process associated with the selected window. If this variable is `nil`, the process name to display in the status line is obtained by invoking the `:process` method on the selected window.

The user can set `w:who-line-process` using the `TERM W` keystroke sequence.

w:last-who-line-process Variable

The process most recently described in the mouse documentation window, regardless of how that process was chosen. This variable may be `nil` if there was no process to describe (for example, if the mouse documentation window was supposed to describe the selected window, but there was no selected window or the window had no process).

w:who-line-clobbered Function

Informs the who-line screen that it must redisplay everything.

File Streams 18.5.2.3 The following paragraphs discuss recording open file streams for display.

When the mouse documentation window describes an open file, the name to display for it is obtained with the `:string-for-wholine` pathname method, described in the *Explorer Input/Output Reference* manual.

w:who-line-file-state-sheet Variable

An instance of the flavor **w:who-line-file-sheet** that is the status of an open stream or active network server. (That is, this instance is the window, part of the status line, on which the file being read appears.) Typical values include `file serving c9` and `<- MH:WEBB;FOO.LISP#>`. This variable can also display the idle time if there is no stream or server. For example, if no input has been read from the keyboard for the last hour and a half, the status line might read `console idle 1 hour 30 minutes`. The value of this variable is displayed in the status line.

This variable also maintains the lists of streams and servers that can be displayed. New streams and servers are reported to **w:who-line-file-state-sheet** by means of the methods discussed in the following descriptions.

w:who-line-file-sheet Flavor

Supplies the methods that update the **w:who-line-file-state-sheet** variable, as follows.

:add-stream *stream update-p* Method of **w:who-line-file-sheet**

Adds *stream* to the list of open I/O streams recorded by the file state sheet. If *update-p* is non-nil, the field in the status line is immediately updated.

:delete-stream *stream* Method of **w:who-line-file-sheet**

Removes *stream* from the list of streams for the status line.

:delete-all-streams Method of **w:who-line-file-sheet**

Clears the list of streams for the status line.

:open-streams Method of **w:who-line-file-sheet**

Returns the list of streams recorded for the status line.

Servers **18.5.2.4** The following functions and methods deal with servers reported in the status line.

:servers Method of **w:who-line-file-sheet**

Returns a list of all active servers.

:add-server *conn contact-name process function* Method of **w:who-line-file-sheet**

Adds an entry to the list of active network servers recorded by the file state sheet.

Arguments: *conn* — The network connection of this server
contact-name — The contact name to which *conn* responded
process — The process in which the server is running
function — A file server function

:delete-server *conn* Method of **w:who-line-file-sheet**

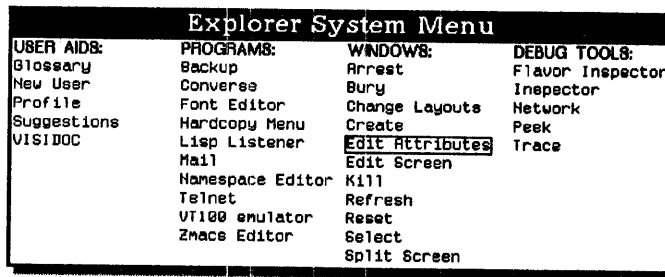
Removes the entry for a connection from the list of servers for the status line. This happens automatically if the connection is broken or closed. *conn* is the network connection of this server.

:delete-all-servers Method of **w:who-line-file-sheet**
 Clears the list of servers for the status line.

:close-all-servers Method of **w:who-line-file-sheet**
w:close-all-servers &optional (*reason* "Closing all remote servers.") Function
 Closes the connections of all network servers. *reason* is a string that is the reason for closing the connections of all network servers. (See the *Explorer Networking Reference* manual for more information about network namespace servers.)

w:describe-servers Function
 Prints descriptions of all active network servers. A sample output is as follows:
 NOMAD serving HAL-9000 in KEYBOARD-PROCESS

The System Menu 18.6 This paragraph describes how to interface with and customize the System menu. The following figure shows the System menu.



The System menu is an instance of the **w:dynamic-multicolumn-momentary-window-hacking-menu** flavor, which means that its menu items are grouped by columns, and each column's items come from the value of a corresponding variable that is examined each time the menu is popped up in case more items have been added. This enables you to add items to the menu and control where they go.

w:add-to-system-menu-column *column-type name form documentation* Function
 &optional *after*

Adds a named item to a column of the System menu.

Arguments: *column-type* — Which of the columns is to be changed. This can be one of **:user-aids**, **:programs**, **:windows**, or **:debug**.

name — The named item to be added.

form — What is executed if the user clicks on the item.

documentation — The mouse documentation string.

after — Can be one of the following:

Value	Description
A string	The item after which <i>name</i> is added.
t	<i>name</i> is added to the top of the menu.
nil	<i>name</i> is added to the bottom of the menu.
:sorted	<i>name</i> is added and then the entire column is sorted alphabetically.

- w:delete-from-system-menu-column** *column-type name* Function
 Deletes *name* from the specified *column* of the System menu. Leading and trailing blanks and alphabetic case are ignored when the arguments are matched with entries in the columns.
- w:*system-menu-user-aids-column*** Variable
 A menu item list that forms the User Aids column of the System menu. By convention, the menu items are different window creation, selection, and editing commands.
- w:*system-menu-programs-column*** Variable
 A menu item list that forms the Programs column of the System menu. By convention, the menu items are the most commonly-used programs. You can add entries to this list by using the **w:add-to-system-menu-column** function.
- w:*system-menu-edit-windows-column*** Variable
 A menu item list that forms the Windows column of the System menu. By convention, the menu items are different window creation, selection, and editing commands.
- w:*system-menu-debug-tools-column*** Variable
 A menu item list that forms the Debug Tools column of the System menu. By convention, the menu items are different tools for developers to use while debugging their programs.

 By convention, this variable is used for operations on the window that the mouse was pointing to when the System menu was popped up. They are implemented with **:window-op** menu items. (See paragraph 14.2.1, Menu Items.)
- w:default-window-types-item-list** Variable
 A menu item list that is used by the System menu Create option and by Create in the screen editor when it is operating on a screen.
- w:add-window-type** *string flavor documentation* Function
 Adds a new window type to the **w:default-window-types-item-list** variable. *string* is the string that appears in the System menu Create option; *flavor* is the flavor of the window to create; *documentation* is displayed in the mouse documentation window.

Window Resources 18.7 A *resource* is a pool of interchangeable objects that are available to be used temporarily and then returned to the pool.

Resources whose objects are windows are often useful. For example, a resource of windows serves as the System menu; when you invoke the System menu, a window is allocated from the resource and returned to the resource's pool when it is deactivated.

You usually define a resource with `defresource`. However, if you are defining windows in the resource you should use the `w:defwindow-resource` macro. Allocating windows from resources, and returning them, is exactly like working with any other resources and is documented in the *Explorer Programming Concepts* manual.

All the symbols described as resources in this manual are defined in this way.

`w:defwindow-resource` *name parameters &rest options* Macro

Defines a resource of windows.

Arguments: *name* — The name `w:defwindow-resource` gives this resource.

parameters — The parameters on which the object can depend. In addition, the window's superior is always defined. When you allocate a window from the resource, this parameter defaults to `w:mouse-sheet`.

options — A list of alternating keywords and values. Neither the keywords nor the values are evaluated at the time `w:defwindow-resource` is executed, but sometimes the value becomes part of an expression that is executed later (when a window is allocated from the resource).

The *options* argument can be `:initial-copies`, `:constructor`, `:make-window`, and `:reusable-when`:

- `:initial-copies` — The number of windows to create in the resource when the resource is defined. The default is 1. The initial copies are made inferiors of `w:default-screen`. Creating an initial copy is simply a way of saving time the first time a window needs to be allocated from the resource.
- `:constructor` — Whether to use the default creation function is not specified, `w:defwindow-resource` provides a default, which calls `make-instance` with arguments taken from the `:make-window` option. (`defresource` is described in detail in the *Explorer Lisp Reference* manual.)
- `:make-window` — A list consisting of a flavor name followed by keyword arguments. This list is consed into a `:make-window` form to produce the constructor for the resource.
- `:reusable-when` — Either `:deexposed` or `:deactivated`. If this keyword is not specified, then windows of the resource can be allocated to requesters if they have been explicitly returned to the pool and are not locked. The `:deexposed` keyword means that any window that is not exposed is considered to have been returned to the pool. The `:deactivated` keyword means that any window that is not active is considered to have been returned to the pool.

The following code creates a pop-up window similar in appearance to a Zmacs minibuffer:

```
;Resource of "fake" minibuffers
(defwindow-resource fake-minibuffer-window ()
  :make-window (fake-minibuffer-window
                :activate-p t)
  :reusable-when :deactivated
  :initial-copies 0)
```

w:window-resource-names

Variable

A list of the names of all window resources defined with `w:defwindow-resource`.

For example, a typical list is displayed as follows:

```
(meter::call-tree-inspector-resource gloss::gloss-pop-up-keystrokes-window
zwei::converse-simple-reply-window eh::debugger-frame w::inspect-frame-resource
zwei::suggestions-pop-up-standalone-editor-frame zwei::small-pop-up-standalone-editor-frame
zwei::suggestions-pop-up-notification-window w::pop-up-keystrokes-window
zwei::pop-up-standalone-editor-frame zwei::split-screen-menu zwei::background-typeout-windows
zwei::temporary-mode-line-window-with-borders-resource zwei::menu-command-menu
ucl::command-editor-window ucl::get-keystroke-window w::trace-pop-up-menu
w::temporary-choose-variable-values-window w::temporary-multiple-choice-window
w::screen-editor-menu w::split-screen-layout-window w::pop-up-text-window-without-more
w::pop-up-text-window w::screen-layout-menu w::split-screen-choose-values w::split-screen-menu
w::confirmation-window w::selectable-windows-menu w::window-type-menu w::system-menu
w::sequence-selection-multiple-menu w::momentary-multiple-menu
w::dynamic-multicolumn-momentary-menu w::momentary-menu w::pop-up-finger-window
w::pop-up-notification-window)
```

Finding Windows

18.8 The system includes a function for finding a window whose name you know.

find-window &optional (*substring* "")

Function

Finds a window whose name has *substring* in it and returns the flavor of the instance of the window. If more than one window matches *substring*, the function displays a menu that contains the possible matches.

Several other functions can be used to find the symbol associated with a window. These are briefly mentioned here, and they are described in detail in the *Explorer Tools and Utilities* manual. For all three of these functions, the value of `:dont-print` determines whether the function returns a list of the symbols found or whether the list is printed. A value of `t` causes the function to return the list; any other value causes the function to display the list.

apropos-resource *substring* &key :predicate :dont-print

Function

Finds all the resources whose names contain *substring*. If `:predicate` is non-`nil`, it is a function to be called with a flavor-name as an argument; only flavors for which the predicate returns non-`nil` are returned.

apropos-flavor *substring* &key :predicate :dont-print

Function

Finds all flavors whose names contain *substring*. If `:predicate` is non-`nil`, it is a function to be called with a flavor-name as an argument; only flavors for which the predicate returns non-`nil` are returned.

apropos-method *substring flavor-instance* Function
&key :predicate :dont-print

Finds all methods of a flavor whose names contain a substring. If **:predicate** is non-nil, it is a function to be called with a method name as an argument; only methods for which the predicate returns non-nil are returned.

Introduction

19.1 In place of the black-and-white monochrome monitor, you can use an optional high-resolution color monitor. Any code that runs on a monochrome system will also run on a color system. The following paragraphs discuss using color with the Explorer window system, including these topics:

- Requirements for coding in color
- How color works
- Initialization options and methods used with color
- Functions that manipulate the color map
- Color ALU functions
- Profile variables for color
- Color and texture in graphic output
- Printing a color screen on a monochrome printer
- Advanced, hardware-specific topics

Coding Requirements for Color

19.2 Except in a few cases, an application developed for use with a monochrome monitor will run on a color system without modifications. Exceptions are discussed in Appendix B, *Converting Applications to Color*.

In standard configuration, two contrasting colors represent the “black and white” of the monochrome monitor. These colors are called the *foreground* and the *background* colors. Specifically, the foreground color is the color of text displayed in a window; background color is the color of the window areas that seem to appear behind the text, hence the name background. Thus, even when you are using a color monitor, you may still be using a black-and-white (that is, a two-color) environment rather than a full color environment.

If you need to determine whether a particular system is a color or a monochrome system, you can use the `w:color-system-p` macro. A similar function, `w:get-display-type`, provides more specific information about the type of monitor that is present on the system.

`w:color-system-p` *window-or-screen*

Macro

Returns `nil` if *window-or-screen* is on a monochrome system; otherwise, the macro returns the type of system, for example `:8-bit-color`. For example, the following form returns `nil` if `*terminal-io*` is on monochrome system:

```
(w:color-system-p *terminal-io*)
```

`w:get-display-type window-or-screen`

Function

Returns a keyword that describes the type of window specified by *window-or-screen*. Currently, this function returns one of the following values:

- `:monochrome` for a monochrome window
- `:8-bit-color` for a color window

As different display types are developed, this function will return additional values.

Certain precautions must be taken in order for applications to work correctly on both monochrome and color systems. Note the following areas of caution:

- Specifying ALUs.
- Directly accessing the contents of screen or bit-save arrays. Do not assume these arrays are always 1-bit arrays because they will become 8-bit arrays on color systems.
- Forcing screen or bit-save arrays to be 1-bit arrays.
- Assuming arrays will be 1-bit arrays when displacing them to other arrays or physical addresses.
- Making use of color-specific instance variables for purposes other than those intended (for example, making use of these instance variables on monochrome systems).
- Accessing hardware directly on monochrome or color systems. Avoid bypassing of the window system functions.

How Color Works on a Monitor

19.3 On a monochrome monitor, each picture element (pixel) can have only two values, either on or off. On a color system, each pixel can have one of 256 values, ranging from 0 to 255. Thus, a pixel is no longer just on or off, black or white, 0 or 1, but rather it is an integer with a range of values.

The color controller actually generates 24 bits of color information per 8-bit pixel to drive the color monitor. These 24 bits are composed of 8 bits for RED, 8 bits for GREEN, and 8 bits for BLUE.* So, each of the three colors (R, G, and B) can take on one of 256 values, with 0 being completely off and 255 being full intensity. Thus, taken in all possible combinations, there are over 16 million possible colors.

Since each pixel can take on one of 256 possible values, each 8-bit pixel value is translated by the hardware into a 24-bit RGB value by indexing into a table called the Color Look-Up Table (LUT). Thus, up to 256 different colors can be displayed on the color monitor.

* The RGB model for representing color is based on the three color photoreceptors in the human eye, which detect either red, green, or blue. The RGB model is *not* based on the three primary colors red, blue, and yellow as many people might expect.

Note that the color associated with an index may change as the contents of the LUT change. This index value is referred to as either the *pixel value* or the *logical color* of a pixel. The value generated by the LUT that corresponds to the logical color is the *physical color*.

Foreground and Background Colors

19.3.1 For many operations such as writing text or drawing a simple line, only two colors are of interest: the foreground color and the background color. When the screen is cleared, all pixels are off (actually set to their background colors). When text appears, these pixels are on (set to their foreground colors). The following table lists the general operations in a monochrome environment and their equivalents in a color environment.

Operation in a Monochrome Environment	Equivalent Operation in a Color Environment
Turn a pixel <i>on</i>	Set a pixel to the window's <i>foreground</i> color
Turn a pixel <i>off</i>	Set a pixel to the window's <i>background</i> color

Each window has its own foreground and background colors. In the simplest case, every window has the same foreground and background colors. You can change these colors using window accessor methods or by setting variables in the Profile utility. (Note that changing a Profile variable only changes the default values to be used when new windows are created and does not change the color of any existing window.) By default, any text or graphics are drawn using the window's foreground color. Thus, any application transferred from a monochrome system runs with black and white replaced by the window's current foreground and background color.

LUTs and the Color Map

19.3.2 As the image stored in memory is transferred to the screen, the Explorer color system uses the values in the LUT to translate 8-bit pixel values into 24 bits of physical color information. The LUT is a single table in the hardware; it cannot be directly read from or written to by the software. Instead, the hardware contains two LUT buffers of its own that can be manipulated directly. The LUT is copied from an LUT buffer on command from the window system.

Although the hardware supports only two LUT buffers, there are no limits on the number of LUTs maintained by the software. Conceivably, each window could require a different set of color definitions to be loaded in the color LUT. Therefore, in a color environment, each window has a data structure called a *color map* associated with it. Although a programmer can read the contents of either buffer or change entries in the LUT (even load an entirely new table), a programmer should *always* work with a window's color map rather than with the LUT buffers.

When a window is created in a color environment, the window system assigns by default a pointer to the superior window's color map. This default action is important for some applications, particularly those applications that use constraint frames. Logically, the panes in a constraint frame share the same color map.

You can assign a specific color map when a window is created by using an initialization option. One option is to use a copy of the superior window's color map rather than use a pointer to it. This allows the application to alter the color map without those changes propagating through the window hierarchy and affecting other windows or applications.

You can change all or part of a window's color map by programming (using functions, methods) or by using the Color Map editor.

Whenever a window is exposed or selected, its color map is automatically loaded into one of the LUT buffers. The LUT is then loaded from the freshly updated buffer. So, as different windows are exposed or selected, the hardware is always using a copy of the exposed or selected window's color map. At the same time that color map is loaded into the LUT, values associated with it are placed where microcode can access them during various operations.

Contents of the Color Map

19.3.3 A *color map* is a data structure (specifically, a **defstruct**) that contains, among other things, 256 locations for logical colors. When loaded in the LUT and accessed, they define the physical colors. Each window can have a separate color map, but many windows simply inherit a color map from a superior. The color map used by system-defined Explorer software is contained in the `w:*default-color-map*` variable. The colors are actually defined in the `w:*default-initial-colors*` variable, which contains the RGB values for each system color.

Specifically, the color map contains the elements described in Table 19-2:

Table 19-1 Elements in the Color Map defstruct

Element Name	Default Value	Type	Description
reserved-slots	0 through 31	list	A list of reserved colors, as explained in Table 19-2.
name	"NAME"	string	The name of this color map.
system-version	"Release-x"	string	The version of the Explorer system software that was running when this color map was created.
CME-version	"0.0"	string	The version of the Color Map editor that was running when this color map was created.
saturate	255	integer	The saturate value, used as an upper bound in certain ALU operations.
clamp	0	integer	The clamp value, used as a lower bound in certain ALU operations.
table	a pointer to an array	array	The pointer to the color table data structure.

`color-map-xxx` *color-map-name*

Function

Returns the value of element *xxx* of the color map *color-map-name*. Color maps are created by using either the `w:make-color-map` or the `w:create-color-map` functions.

For example, the following form returns the value of the `name` element of the color map called `rainbow-map`:

```
(color-map-name rainbow-map)
```

You can use `setf` to set the element values, for example:

```
(setf (color-map-name rainbow-map) "New Name")
```

Reserved Colors

19.3.4 The reserved colors are the first 32 slots in the color map table and are listed by index number in the `reserved-slots` part of the color map. All system software is defined to use only these reserved colors for such things as the default foreground and background colors of a window, border colors, highlighting colors, and so on. By confining the specialized colors you need for your application to the rest of the color map table, you can ensure that all your windows maintain a consistent appearance.

For example, suppose your application requires a set of pastel colors, some shades of brown, and several shades of green. Before your program exposes your application window, it could use the color map functions defined on the following pages to define the pastel colors in locations (for example) 40 through 64, the shades of brown in 70 through 90, and the shades of green in locations 200 to 215.

NOTE: Although you can modify any location in the color map table, it is recommended that you not alter the contents of any of the reserved slots unless absolutely necessary. You may create odd-looking displays when your color map table is loaded and the user is working with system-defined applications. For example, suppose the user is running your application which uses a set of colors that gradually change through the spectrum. If the user invokes the System menu, the System menu also uses the application's colors—which, because of the gradual shading, means that the System menu is now displayed as, say, a medium shade of blue on a very light shade of blue. The System menu will be virtually unreadable.

Also, if the user is running your application in a split screen with a system application, and your application has redefined the reserved colors, all the various applications change color each time the user selects a different application.

The reserved 32 slots, listed in Table 19-2, contain common named colors plus eight shades of gray for compatibility with the eight gray or stipple patterns in the monochrome environment.

Naming Colors

19.3.5 A color name is associated with a specific color index by the `w:color-alist` variable. All these indexes refer to reserved slots in the default color map table. If any of the reserved locations are modified, either under program control or through the Color Map editor, these names become meaningless (that is, the physical color no longer bears the same relationship to the color name).

The remaining locations of the color map table are loaded with values for a red color ramp, a green color ramp, and a blue color ramp. (A *ramp* is a smoothly varying sequence of colors. Each of the default ramps vary in intensity from very dark shades to very light shades of the same color.) The last location is loaded with black because it is the opposite of color 0, which is white. The order of the colors was chosen to give good contrast between the background colors and blinkers, since blinkers, by default, are set equal to the background color plus an offset of 9. The colors are actually defined by the value of the `w:*default-initial-colors*` variable.

If you want to associate a specific color name to appear in any menu that contains all the color names, add the color name and its index number to the association list in the `w:color-alist` variable. For example, the following code adds the color name `dull-red` to the list of displayed colors:

```
(setq w:color-alist (append w:color-alist `(("dull-red" . 95))))
```

Table 19-2

Named Colors in the Default Color Map Table

Index Number	Color Name
0	w:white
1	w:12%-gray-color
2	w:25%-gray-color
3	w:33%-gray-color
4	w:50%-gray-color
5	w:66%-gray-color
6	w:75%-gray-color
7	w:88%-gray-color
8	w:black
9	w:dark-green
10	w:blue
11	w:red
12	w:orange
13	w:purple
14	w:pink
15	w:cyan
16	w:magenta
17	w:yellow
18	w:dark-blue
19	w:green
20	w:dark-brown
21	w:blue-green
22	w:light-brown
23	w:red-purple

NOTE:

Locations 24 through 31 are reserved for future use.

Initialization Options and Methods Used With Color Windows

19.4 The following initialization options and methods are all that most people need to manipulate color in windows on the Explorer system. Examples of how to specify color in menu items are given with other examples of menus in Section 14, Choice Facilities. If you want to set up your own color map, see paragraph 19.5, Functions That Manipulate the Color Map.

In addition to the following initialization options and methods, you can also set the foreground and background colors for a label by specifying those colors in a keyword-argument list as part of the label specification, as described in paragraph 3.4, Labels. An example of this specification is given at the end of this numbered paragraph.

If these initialization options are not specified when you create a window, the Explorer system uses the values in the `w:*default-background*` (initially `w:12%-gray-color`) and in the `w:*default-foreground*` (initially `w:black`) variables. These variables are Profile variables and can be set by the user.

<code>:background-color</code>	<i>value</i>	Initialization Option of <i>windows</i>
<code>:background-color</code>		Method of <i>windows</i>
<code>:set-background-color</code>	<i>value sheet-or-window</i>	Method of <i>windows</i>

Initializes the `w:background-color` instance variable of the `w:sheet` flavor for *sheet-or-window*.

For example, the following forms return and set, respectively, the background color of the window named `my-window`:

```
; Assumes my-window has already been created.
(format my-window "The screen background color is -D"
 (send my-window :background-color))

(send my-window :set-background-color w:dark-green)
```

<code>:foreground-color</code>	<i>value</i>	Initialization Option of <i>windows</i>
<code>:foreground-color</code>		Method of <i>windows</i>
<code>:set-foreground-color</code>	<i>value sheet-or-window</i>	Method of <i>windows</i>

Initializes the `w:foreground-color` instance variable of the `w:sheet` flavor for *sheet-or-window*.

For example, the following forms return and set, respectively, the foreground color of the window named `my-window`.

```
; Assumes my-window has already been created.
(format my-window "The screen foreground color is -D"
 (send my-window :foreground-color))

(send my-window :set-foreground-color w:red)
```

<code>:complement-bow-mode</code>		Method of <i>windows</i>
-----------------------------------	--	--------------------------

Transposes the values of the foreground and background colors in the window. This method has no effect on a monochrome system because monochrome systems do not use foreground and background colors. Note that the `w:complement-bow-mode` function can be used only in a monochrome system.

:blinker-offset *value* Initialization Option of *blinkers*
:blinker-offset Method of *blinkers*
:set-blinker-offset *value* Method of *blinkers*

Initializes the **w:blinker-offset** instance variable of the **w:blinkers-mixin** flavor. The blinker offset is the value added to the value of the color on the screen to produce the color of the blinker. To erase the blinker (that is, to make it blink), the Explorer system subtracts the value of **w:blinker-offset** from the color of the blinker, which produces a blinker that is invisible or that has blinked off. By default, **w:blinker-offset** is 9.

:border-color *value* Initialization Option of **w:borders-mixin**
:border-color Method of **w:borders-mixin**
:set-border-color *value* Method of **w:borders-mixin**

Initializes the **w:border-color** instance variable of the **w:borders-mixin** flavor for *sheet-or-window*. The border of the window is displayed in the color specified by *value*. Most windows include **w:borders-mixin** as a component flavor.

:label-color *value* Initialization Option of **w:label-mixin**
:label-color Method of **w:label-mixin**
:set-label-color *value* Method of **w:label-mixin**

Initializes the **w:label-color** instance variable of the **w:label-mixin** flavor for *sheet-or-window*. The label of the window is displayed in the color specified by *value*. Most windows include **w:label-mixin** as a component flavor.

:label-background *value* Initialization Option of **w:label-mixin**
:label-background Method of **w:label-mixin**
:set-label-background *value* Method of **w:label-mixin**

Initializes the **w:label-background** instance variable of the **w:label-mixin** flavor for *sheet-or-window*. The background of the label of the window is displayed in the color specified by *value*. Most windows include **w:label-mixin** as a component flavor.

The following code shows how to create a window using these initialization options.

```
(setq my-window (w:make-window 'w:window
                             :foreground-color w:white
                             :background-color w:dark-green
                             :border-color w:green
                             :label `(:string "My Window" ; note backquote
                                       :color ,w:black ; note comma
                                       :background ,50%-gray-color)
                             ))
(send my-window :expose)
```

Functions That Manipulate the Color Map

19.5 The various color map functions provide tools for the programmer to create and modify color maps, to load color maps into the hardware buffers, and to copy the contents of the hardware buffers into a color map structure.

The following example uses several of the functions that manipulate the color map. The example creates a color map called `rainbow` by using the **w:make-color-map** function, writes the color map, and then displays a window with a rainbow circle.

```
(defun make-rainbow ()
  (let ((raw-map (w:make-color-map))
        )
    (fill-colors raw-map) ; Create the rainbow colors
    (setf (tv:color-map-name raw-map) "Rainbow") ; Change the name
    raw-map)
  )
```

```
(defun fill-colors (color-map &optional (function #'merge-colors-1))
  "This uses the function merge-colors to fill the color-map"
  (loop for list in (funcall function)
        for index from 1. to 254.
        doing (apply #'w:write-color-map color-map
                     index list)
        finally (w:write-color-map color-map 0. 0. 0. 0.)
               (w:write-color-map color-map 255. 255. 255. 255.))
  )
```

```
(defun merge-colors-1 ()
  "This provides a fairly uniform rainbow from 1 through 254, with
  black at 0 and white at 255."
  (append (loop for iloop from 1. to 21 ;;pink to red = 21
               collecting (list 252.
                               (- 126. (* iloop 6.))
                               (- 126. (* iloop 6.))))
          (loop for iloop from 1. to 42. ;;red to orange = 63
               collecting (list 252.
                               (round (* iloop 4)
                                       0.))
               )
          (loop for iloop from 1. to 42. ;;orange to yellow = 105
               collecting (list 252.
                               (+ 168. (round (* iloop 2))
                                       0.))
               )
          (loop for iloop from 1. to 36. ;;yellow to green = 141
               collecting (list (round (- 252. (* iloop 7.))
                                       252.
                                       0.))
               )
          (loop for iloop from 1. to 36. ;;green to blue = 177
               collecting (list 0.
                               (round (- 252. (* iloop 7.))
                                       (* iloop 7.))
               )
          (loop for iloop from 1. to 36. ;;blue to violet = 213
               collecting (list (round (* iloop 7.)
                                       0.
                                       252.))
               )
          (loop for iloop from 1. to 42. ;;violet to white = 255
               collecting (list 252.
                               (round (* iloop 6.)
                                       252.))
               )
  )
  )
```

```
;;; This function saves the old color map, loads a new one, and then creates a
;;; rainbow circle.
```

```
(defun demo-rainbow (window)
  (let ((old (w:copy-color-map (send window :color-map)))
        (new (make-rainbow))
        )
    (send window :set-color-map new) ; sets the color map
    ; When a window is selected, its color map is loaded into the hardware LUT
    (send window :select)
    (dotimes (i 255) (send window :draw-filled-circle
                               500 400 (- 265 i) +1))
    (dotimes (i 512) (send window :draw-filled-circle
                               500 400 265 1 w:alu-add))
  )
```

```

        (send window :set-color-map old)           ; restores the old color map
        (send window :select)
      )

```

;;; Enter the following form in a Lisp Listener to execute the example:

```

(demo-rainbow *terminal-io*)

```

General Functions 19.5.1 The following functions manipulate the color map for any type of hardware.

w:sheet-color-map *sheet-or-window* Function
:color-map Method of *windows*

Returns the color map for *sheet-or-window*.

For example, the following returns the color map for the **terminal-io** stream.

```

(w:sheet-color-map *terminal-io*)

```

:set-color-map *new-map* &optional (*inherit nil*) Method of *w:sheet*

Sets the color map of the window to the new color map. If *inherit* is *t*, this method forces inferiors to point to the same color map, unless the inferior had its own copy originally.

w:make-color-map Function

Allocates the color map structure but does not initialize it.

w:create-color-map &optional *init-plist* Function

Calls **w:make-color-map**, then initializes the color map table either to defaults or to values supplied in the *init-plist* argument. *init-plist* is a list of keyword-value pairs. Allowable keywords include any of the slot names of a color map structure except for *table*. The arguments for each of these keywords are the contents of the color map structure. To assign colors to the color map, use the **:initial-colors** keyword which takes a list as an argument. This list is composed of sublists of the form:

```

(index-number red-value green-value blue-value)

```

where each of these elements is an integer between 0 and 255. The *index-number* argument is the number of the slot in the color map table. The other values represent the red, green, and blue values for the color in the *index-number* slot.

For example, the following code creates a list of colors in *x* and then creates a color map named *special* that is accessed through *z*.

```

(setq x (append w:*default-initial-colors*
               (list '(33 50 60 70) '(34 100 100 100) '(35 100 200 200)
                     '(46 100 0 255) '(40 0 0 255))
           ))

(setq z (w:create-color-map '(:saturate 47
                             :clamp 9
                             :name "Special"
                             :initial-colors x))

```


w:copy-color-map *source-color-map* Function
Returns a color map that is an exact copy of the source color map, including contents.

w:read-color-map *color-map index* Function
Returns the values of red, green, and blue for *color-map* at the specified *index*. *index* can be an integer from 0 through 255, inclusive.

w:write-color-map *color-map index red green blue* Function
Writes the *red*, *green*, and *blue* values into the location specified by *index* of *color-map*. *index* can be an integer from 0 through 255, inclusive.

For example, the following code changes the color in index position 33 of the color map table named `special` (created in the previous example) to white, which has RGB values of (0 0 0).

```
(w:write-color-map z 33 0 0 0)
```

w:write-color-map-file *color-map* Function
Saves the color map specified by *color-map* to a file that defaults to LM:COLOR-MAPS; *name-of-map*.XLD. When you are prompted for the name of the color map, the symbol is set to the color map and the name element of the map is set to the symbol (a string).

NOTE: This function is available only if the Color Map editor software is loaded. Refer to the Color Map Editor section in the *Explorer Tools and Utilities* manual.

w:select-color-with-mouse &optional (*color-map w:*default-color-map**) Function
Displays 256 rectangles in the colors of the *color-map*. When you click on any of the rectangles, the function returns the index value of that color in the color map. During the execution of this function, *color-map* becomes the active color map. The following example allows you to select a color from the default color map:

```
(defun choose-a-color ()
  (let ((my-color (w:select-color-with-mouse)))
    (send w:selected-window :draw-filled-arc 500 500 400 400 360
      my-color))
  )
```

Hardware-Specific Functions

19.5.2 The following functions manipulate the color map for only hardware supported by the current release.

CAUTION: These functions may change incompatibly without notice. **DO NOT** use these functions unless absolutely necessary. Instead, use the functions listed in the preceding paragraph, titled **General Functions**. **DO NOT** redefine these functions; to do so may cause your system to crash.

- w:download-color-lut-buffer** *color-map* &optional *buffer* Function
 Loads the color map table specified by *color-map* into the LUT buffer specified by *buffer*. The value of *buffer* can be either 0 or 1. If *buffer* is the current buffer, the **w:transfer-color-lut-buffer** function is executed automatically.
- w:get-color-lut-buffer** *color-map* &optional *buffer* Function
 Copies the color LUT buffer specified by *buffer* into the color map table specified by *color-map*. The value of *buffer* can be either 0 or 1.
- w:transfer-color-lut-buffer** &optional *buffer* Function
 Transfers the LUT buffer specified by *buffer* to the LUT. The value of *buffer* can be either 0 or 1.
- w:current-color-lut-buffer** Function
 Returns the number of the buffer that was last transferred to the LUT.
- w:read-color-lut-buffer** *index* &optional *buffer* Function
 Returns three values: the red, green, and blue values for the specified LUT buffer at the specified *index*. The value of *index* can be an integer from 0 through 255, inclusive. The value of *buffer* can be either 0 or 1.
- w:write-color-lut-buffer** *index red green blue* &optional *buffer* Function
 Writes the *red*, *green*, and *blue* values into the location of the specified color LUT buffer at the specified *index*. The value of *buffer* can be either 0 or 1. *index* can be an integer from 0 through 255, inclusive.

Color ALU Functions

19.6 To better understand the capabilities provided by the color Explorer, specifically the new ALU operations, a good knowledge of how drawing operations work on an Explorer is necessary. The following lists several ways in which drawing can occur on the Explorer screen:

- Character and string output methods and functions
- Graphics (lines, circles, and so on) output methods and functions
- Image transfer functions (for example, **bitblt**)

In all cases, drawing operations make use of a *source* pattern (array) and a *destination* array. In some operations the source is specified explicitly (**bitblt**), in some the source is specified implicitly (**:draw-line**), and in others the source is specified indirectly (**:draw-character**).

When drawing a character, a source pattern containing 0s and 1s is used. On a monochrome system, pixels are either on or off, thus only the 1s and 0s are needed. On a color system, in order to allow characters to appear in color, a *foreground* color (described later) and a *background* color must be used. In the most general case, 1s in the source are drawn in the foreground color, and the 0s are drawn in the background color.

When drawing graphics (a circle, line, rectangle, and so on), a 1-bit source pattern is used by the microcode. On a monochrome system, the source pattern can contain 1s and 0s in order to simulate some gray shade. On a color system, the source pattern contains all 1s because a foreground color is used for each pixel. Again, in the most general case, 1s are drawn in the foreground color.

In either of these examples, the results (destination) are affected by the ALU operation specified.

On color systems, the concept of using foreground and background colors when drawing 1-bit source patterns is called *color expansion*. Expansion is used when the source pattern is a 1-bit array and the destination array is an 8-bit array. On color systems, you can think of the display screen as being a 8-bit array. Expansion is needed so that color will appear on the screen even though 1-bit source patterns are being used. The 1s in the source pattern are expanded to the foreground color, and 0s are expanded to the background color.

On both monochrome and color systems, ALUs are used to determine how to combine the source and the destination. Additional ALUs have been added for color systems. In all cases, when a 1-bit source and a 8-bit destination are being used, the source is expanded before the ALU operation is applied. The additional color ALUs are described in the following paragraphs. An important point to understand is that the color ALUs produce the same results as the standard boolean ALUs when used on monochrome systems. However, certain boolean ALUs do not produce the desired effect when used on color systems. For example, the add-color ALU produces the same effect as exclusive-or (`w:alu-xor`) when used on a monochrome system. However, `w:alu-xor` does not produce a predictable effect on color systems. Instead, unexpected colors appear when using `w:alu-xor` on color systems. Thus, to ensure an application works correctly on both monochrome and color systems, color ALUs should be used whenever possible.

The color ALU operations combine the (color) value of the pixel already drawn on the window (destination) with the (color) value of the pixel to be drawn (source). In general, color ALU operations take the two color values, apply an ALU operation to them, then use the resulting value as an index to the color map table. Thus, in a color environment, the same values combined using the same ALUs will produce different colors according to the color map.

The default generic ALUs are defined the same in both monochrome and color environments. Specifically, `w:char-aluf` uses `w:alu-transp`, and `w:erase-aluf` uses `w:alu-back`. Thus, in a color environment, anything drawn on the window using the general ALU arguments writes in the foreground color and erases to the background color. If an existing application uses the generic ALUs, in most cases you need not update them. However, if an existing application uses `w:alu-xor` to draw output to the window, you should use a color ALU function while running in a color environment. Any new code you write should use the new color ALU functions even if it is to run on the monochrome system.

Color ALU operations are add, subtract, max, min, add with saturation, subtract with clamping, transparency, and background. Each of these operations is described in detail in the following pages and are summarized in Table 19-3.

Add and Subtract

19.6.1 The add and subtract color ALUs are straightforward operations performed modulo 256. That is, if a pixel's current value is 254 and a value of 4 is added, the result is $254 + 4 = 258$. However, 258 is greater than the largest available value, so the result is $258 - 256 = 2$. You can also think of the addition as wrapping around to the front of the table. Similarly, a value of 3 minus a value of 7 yields a pixel value of 252.

The color associated with these values depends on the colors in the color map table. For example, if the color map table contains shades of blue from 240 through 255 and shades of gray from 0 through 7, then—using the $254 + 4 = 258$ example from the previous paragraph—the color of the pixel changes from 254 (a shade of blue) to 4 (a shade of gray).

One system application for the add and subtract ALUs is for drawing blinkers. In a monochrome environment, blinkers are drawn on the screen using XOR and then removed using a second XOR. For the color environment, the blinker is added to make it appear and subtracted to make it disappear. The wraparound feature is important because adding an offset of, say, 9, to a color 255, must still yield a valid color. In addition, the value must return to 255 when 9 is subtracted from the new value so that the contents of the screen are restored correctly.

For example, the following code continually changes color until aborted.

```
(loop
  (send w:selected-window
    :draw-filled-arc 500 500      ; center of the arc
                    400 400      ; starting point
                    360           ; degrees of arc
                    w:red         ; initial color if background is 0
                    w:alu-add)    ; alu function
  )
```

**Add With Saturate
and Subtract With
Clamping**

19.6.2 For some applications, you do not want the colors to wrap around the color map table. For example, suppose you want to use one of the shades of green that are contained between 33 and 55; color 56 starts the shades of, say, red. To ensure that the values stayed within the range of green shades, you would set the clamping value to 33 and the saturation value to 55. Then, $40 + 10$ yields 50, which is in the range of allowed values. However, $40 + 20$ yields 55 because the result cannot exceed the saturation value. Similarly, $40 - 5$ yields 35, but $40 - 10$ yields 33.

The saturation and clamping values are set for each color map and can be changed using the `setf` function, as described in paragraph 19.3.3, Contents of the Color Map.

**Minimum,
Maximum,
and Average**

19.6.3 The minimum, maximum, and average operations are straightforward arithmetic operations. In the averaging operation, any fractions are truncated. For example, $(\text{max } 20 \ 30)$ yields 30; $(\text{min } 20 \ 30)$ yields 20; $(\text{average } 20 \ 30) = (20 + 30)/2 = 25$; and $(\text{average } 2 \ 3) = (2 + 3)/2 = 2$.

Of course, the results of these operations make sense only if the color map table is properly prepared.

**Background
and Transparency**

19.6.4 The background ALU operation forces the destination pixel to the window's current background color regardless of the value of the source or destination pixel value.

With the transparency ALU, output is drawn on a window so that existing colors seem to show through a pattern (such as text) that is being drawn on the display. Each pattern is composed of on and/or off bits. When drawn to the screen in a color environment, in general, on bits result in pixels being set to the foreground color and off bits leave current pixels unchanged. With a pattern such as text, the system should do nothing with the off bits. Thus, existing screen contents show through the pattern, and the gaps in the pattern itself appear transparent.

The rules of transparency (for **w:char-aluf**, **w:combine**, and **w:alu-transp**) are as follows. (Note these rules apply only to color systems.)

1. If the source is 1-bit (as in drawing text, graphics, or 1-bit patterns) and the destination is 8-bit (as is the display screen), source values of 0 (zero) are ignored (leaving the destination as is), and source values of 1 are translated to the foreground color and written to the destination. The concept of translating a 1-bit source into an 8-bit value (destination) is referred to as *expansion*.
2. If the source is 8-bit (as in 8-bit texture patterns), and the destination is 8-bit, source values (colors) equal to the background (transparency) color are ignored, leaving the destination as is. If the source value does not equal the destination, the source is written to the destination.

NOTE: If the source is 1-bit and the destination is 1-bit, transparency has no effect. What actually happens is an inclusive-or (**w:alu-ior**).

Also note that it is invalid to have an 8-bit source and a 1-bit destination.

Table 19-3 Color ALU Operations

Operation	Description
<code>w:alu-add</code>	The source and destination are added, and the result is allowed to wrap around modulo 255.
<code>w:alu-adds</code>	The result is the smaller of the saturation value or the sum of the source and destination.
<code>w:alu-avg</code>	The source and destination are averaged. Any fractions are truncated.
<code>w:alu-back</code>	The result is the background color regardless of the value of the source or destination.
<code>w:alu-max</code>	The result is the larger of the source or destination.
<code>w:alu-min</code>	The result is the smaller of the source or destination.
<code>w:alu-sub</code>	The source and destination are subtracted, and the result is allowed to wrap around 0.
<code>w:alu-subc</code>	The result is the larger of the clamp value or the difference between the source and destination.
<code>w:alu-transp</code>	The result combines the source and destination according to the rules of transparency.

Color Versus Monochrome ALU Functions

19.6.5 Any new code you write should use the new color ALU functions because, when the code is run on a monochrome system, the ALU functions produce suitable results equivalent to the old monochrome ALU functions. The mappings are shown in Table 19-4, and the equivalent truth tables are shown in Table 19-5. Note that some monochrome ALU functions do not have equivalent color ALU functions.

Table 19-4

Monochrome ALU Functions Used by the Color ALU Functions on a Monochrome System

Color ALU Function	Equivalent Monochrome ALU Function
<code>w:alu-add</code>	<code>w:alu-xor</code>
<code>w:alu-adds</code>	<code>w:alu-ior</code>
<code>w:alu-avg</code>	<code>w:alu-ior</code>
<code>w:alu-back*</code>	<code>w:alu-andca</code>
<code>w:alu-max</code>	<code>w:alu-ior</code>
<code>w:alu-min</code>	<code>w:alu-and</code>
<code>w:alu-sub</code>	<code>w:alu-xor</code>
<code>w:alu-subc</code>	<code>w:alu-andca</code>
<code>w:alu-transp*</code>	<code>w:alu-ior</code>

Note:

* It is highly recommended that you use the `w:erase-aluf` instance variable (preferably) or `w:erase` instead of `w:alu-back` and that you use the `w:char-aluf` instance variable (preferably) or `w:combine` instead of `w:alu-transp` because this will help to remain compatible with future Explorer systems.

Table 19-5

Truth Tables for Color ALU Functions on Monochrome Displays

Input		Results				
Source	Screen	min and	adds transp max ior	add sub xor	subc andca	*
						**
0	0	0	0	0	0	
0	1	0	1	1	0	
1	0	0	1	1	1	
1	1	1	1	0	0	

NOTES:

* 8-bit color ALU functions on a monochrome (1-bit) system

** monochrome ALU functions

Profile Variables for Color

19.7 Table 19-6 lists the Profile variables that can be set using the Color Display option of the Profile utility. Each of these variables except `w:*default-blinker-offset*` takes a color as its value. A color can be either an integer in the range 0 through 255, or the name of a color such as `w:blue`. The `w:*default-blinker-offset*` variable takes an integer that is added to the foreground color of the window to produce the color of the blinker.

Table 19-6 Profile Variables for Color

Variable	Default Value	Description
<code>w:*default-foreground*</code>	<code>w:black</code>	Default foreground color for the window.
<code>w:*default-background*</code>	<code>w:12%-gray-color</code>	Default background color for the window.
<code>w:*default-border-color*</code>	<code>w:black</code>	Default color for the border of the window.
<code>w:*default-menu-foreground*</code>	<code>w:black</code>	Default foreground color for menus.
<code>w:*default-menu-background*</code>	<code>w:25%-gray-color</code>	Default background color for menus.
<code>w:*default-menu-label-foreground*</code>	<code>w:white</code>	Default foreground color for the labels of menus.
<code>w:*default-menu-label-background*</code>	<code>w:50%-gray-color</code>	Default background color for the labels of menus.
<code>w:*default-documentation-foreground*</code>	<code>w:black</code>	Default foreground color for the mouse documentation window.
<code>w:*default-documentation-background*</code>	<code>w:33%-gray-color</code>	Default background color for the mouse documentation window.
<code>w:*default-status-foreground*</code>	<code>w:12%-gray-color</code>	Default foreground color for the windows in the status line.
<code>w:*default-status-background*</code>	<code>w:black</code>	Default background color for the windows in the status line.
<code>w:*default-scroll-bar-color*</code>	<code>w:50%-gray-color</code>	Default color for the shaded area of scroll bars.
<code>w:*default-blinker-offset*</code>	9	Default value added to the colors appearing on the screen to produce a contrasting blinker.
<code>w:*default-label-background*</code>	<code>w:25%-gray-color</code>	Default background color for labels.
<code>w:*default-label-foreground*</code>	<code>w:black</code>	Default foreground color for labels.

Color and Texture in Graphic Output

19.8 The `w:graphics-mixin` methods for drawing have an argument named `color` as well as an argument named `texture`. The `color` argument is an integer pixel value which, in a color environment, represents an index into a color map table. In a monochrome environment, this integer is mapped to a texture, or stipple, pattern that represents a shade of gray.

The optional `texture` argument applies to both a color and a monochrome environment. If `texture` is applied in a color environment, the texture is drawn in the color specified. If `texture` is applied in a monochrome environment, the `texture` argument specifies the texture of the output, and the `color` argument is ignored.

The value of `texture` is an array suitable for `bitblt`ing to the screen. The type of array determines how the texture appears:

- If the array is a one-bit array, it is expanded using the foreground and background colors of the window.
- If the array is an eight-bit array, each eight-bit element of the array is treated as a color, and the `color` argument itself is ignored.

In both cases, the resultant appearance on the screen is a produced by a combination of the contents of the texture array, current pixel values, and the specified ALU operation.

The Explorer system provides a set of system-defined 1-bit textures in the `w:*textures*` variable.

`w:*textures*`

Variable

An array of the system-defined texture patterns. You can view the choices using the `w:select-texture-with-mouse` function.

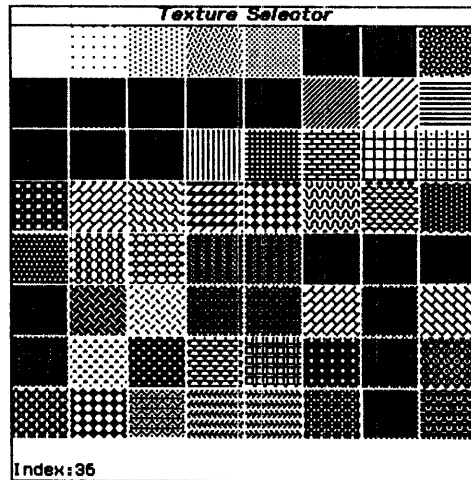
`w:select-texture-with-mouse` &optional (`color w:black`)

Function

Displays the various system-defined textures in the color specified by `color`. Use the mouse to select a texture, and the function returns an integer that is the index value from the `w:*textures*` array. NO TAG shows a typical display from this function.

Figure 19-1

Produced by w:select-texture-with-mouse



For example, the following code, when evaluated in the Lisp Listener, allows you to select a texture for a filled rectangle:

```
(send w:selected-window
      :draw-filled-rectangle
      100 100           ; upper left corner
      200              ; width
      250              ; length
      w:green          ; initial color
      w:normal         ; alu function
      t                ; draw the edge
      ;; displays a menu from which you can select a texture pattern
      (aref w:*textures* (w:select-texture-with-mouse)))
```

Alternatively, instead of invoking the textures displayed with `w:select-texture-with-mouse`, you can specify a particular texture by using the index, a number from 0 through 63, as follows:

```
(aref w:*textures* 43)
```

You can also use any of the shades of gray used on a monochrome monitor by using their names. These shades of gray are listed in Table 12-9. Note that these shades of gray, while similar to the textures displayed by `w:select-texture-with-mouse`, are not identical. Also note that a variable with a name such as `w:88%-gray` points to an array; a variable with a name such as `w:88%-gray-color` contains an integer that is an index value from a color map table.

For example, the following code, when evaluated in the Lisp Listener, produces a filled circle that includes a texture:

```
(send w:selected-window
      :draw-filled-arc 500 500   ; center of the arc
                        400 400   ; starting point
                        360        ; degrees of arc
                        w:red      ; initial color
                        w:alu-transp ; alu function
                        29         ; number of points on edge
                        t          ; draw the edge
                        w:88%-gray ; texture pattern)
```

You may also specify your own texture. For example, the following code creates and fills an eight-bit array. The last form draws a rectangle on **terminal-io** using that array.

```
(setq 8b-array
  (make-array '(32 32) :element-type '(mod 256)
    :initial-contents
    (circular-list
      (list 10 10 10 10 15 15 15 15 12 12 12 12 15 15 15 15
        15 15 15 15 13 13 13 13 15 15 15 15 10 10 10 10))
    ))
(send *terminal-io*
  :draw-filled-rectangle 20 300 1000 150 w:red w:normal t 8b-array)
```

Printing a Color Screen on a Monochrome Printer

19.9 If your color screens contain only foreground and background colors, printing color screens to monochrome printers presents no problem at all. In this case, each pixel in the foreground color is translated to black (value of 1), and each background color pixel is translated to white (value of 0). However, usually your screen contains more than just foreground and background colors. A translation is necessary to represent the remaining colors.

On a color screen, each pixel is represented by a byte value, while a dot on a monochrome printer can have only a one-bit value. Because of this disparity, the most efficient solution for printing is to translate groups of color pixels (color patterns) into various gray patterns. This kind of translation results in a certain amount of contrast reduction (or data compression).

For example, if you display 100 different colors in a 10x10 color pixel array, it is impossible to contrast 100 different gray shades in a 10x10 monochrome array. Most gray shades require a 4x4 bit area to make their pattern noticeable. Therefore, a color-to-monochrome translation works best when there are about 16 pixels of the same color grouped together. Depending on your application, this grouping may be a problem; however, for most cases, the default contrast is enough to be apparent.

Another problem is that there are not nearly enough gray shades in the default system to provide a one-to-one mapping for each of the possible 256 currently displayable colors. Therefore, several colors map to the same gray shade. You can adjust for this problem in two ways:

- Create more gray patterns and enter them into the **printer:color-to-gray-scale-table** variable. (Refer to the **w:make-gray** function in paragraph 12.4.2, Bit Block Transferring, for information on creating gray patterns).
- Modify the entries in the **printer:color-to-gray-scale-table** variable to ensure that the colors you care about do not map to the same gray shade.

If your application needs to make a color-to-monochrome translation, or if you merely want to modify the existing algorithm, the following variable and functions are helpful.

printer:color-to-gray-scale-table

Variable

An array of length 256. The print screen utilities index into this array with a color value to find the appropriate gray shade to use for the color. This table is initialized with 9 gray patterns with specific mappings for the 24 named colors; for other colors, the 9 gray patterns repeat to fill the array. Table 19-7 lists the gray patterns and which colors are mapped to them. If you plan to use several colors and expect to use monochrome screen dumps, you should choose the colors carefully so they contrast on both the screen and the hard copy.

Notice that in the default case, the **printer:translate-color-array** function does not access this table for foreground and background colors because they are immediately translated to ones and zeros, respectively.

Table 19-7

Gray Patterns Used for Printing the Named Colors

Gray Pattern	Colors Printed With That Pattern
w:white	
w:12%-gray-color	w:cyan, w:yellow
w:25%-gray-color	w:pink, w:magenta
w:33%-gray-color	w:red-purple
w:50%-gray-color	w:red, w:green
w:66%-gray-color	w:blue, w:orange
w:75%-gray-color	w:purple, w:light-brown
w:88%-gray-color	w:dark-green, w:blue-green
w:black	w:dark-blue, w:dark-brown

printer:get-gray-scale-value *x y gray-scale*

Function

Returns a 1 or 0 depending on whether the monochrome value at this location should be set at location *x,y* according to *gray-scale*. The *x* and *y* values should be nonnegative integers, and *gray-scale* should be a gray-scale array as defined by the function **w:make-gray**. The *x* and *y* coordinates are converted by using modulo arithmetic to indices into the *gray-scale* array.

printer:translate-color-array *color-array monochrome-array*

Function

&optional *foreground-color background-color color-start-x color-start-y monochrome-start-x monochrome-start-y color-end-x color-end-y*

Translates the pixel values from *color-array* to bit values stored in *monochrome-array*. Every element in *color-array* should be a number from 0 to 255. For each location in *color-array* that is **eql** to *foreground-color*, the corresponding location in the *monochrome-array* is set to 1, while *background-color* locations are set to 0. The defaults for *foreground-color* and *background-color* are **w:*default-foreground*** and **w:*default-background***, respectively. Color values that are neither foreground nor background translate to the following:

```
(printer:get-gray-scale-value x-coordinate y-coordinate
 (printer:color-to-gray-scale-table color-value-at-x-y))
```

The *color-start-x*, *monochrome-start-x*, *color-start-y*, and *monochrome-start-y* arguments indicate where the first elements are read from or written to for their respective arrays. The default value for each of these arguments is 0. The *color-end-x* and *color-end-y* values indicate the limits in the x and y directions for the translation relative to *color-array*. The default for these values is the respective dimension limit in *color-array*. The specified portion of the *monochrome-array* must be large enough to contain the translated values of the specified portion of *color-array*.

Plane Masks

19.10 A *plane mask* is a feature of the color window system that is typically used for only highly specialized applications. The screen array for a color window can be thought of as a three-dimensional array that is eight bits deep and as wide and high as the monitor screen. (In the case of the Explorer color monitor, this is 1048 x 808 pixels.) The depth of the array can be considered as a single array of eight-bit values, or as eight planes of information, each one-bit thick.

The plane mask is an eight-bit value that acts as a mask, specifying which plane(s) can be modified by any drawing operations. For example, if the plane mask is hexadecimal BF, then all planes except plane 6 can be written to. Each window has its own plane mask specified as an instance variable of *w:sheet*.

CAUTION: These methods are hardware-specific and the exact format could change in future software releases.

:plane-mask *register-block*

Method of *windows*

:set-plane-mask *value* *register-block*

Method of *windows*

Returns or sets, respectively, the hardware plane mask. The *value* argument is the new value of the plane mask; *register-block* should be 0.

For example, the following code sets the plane mask so that only half of the planes can be changed.

```
(send w:selected-window :set-plane-mask #x0F 0) ; mask off half the planes
```

Methods to Control Monitors Directly

19.11 The color hardware can control a color monitor and a monochrome monitor at the same time; both would show the same display, one in color and one in monochrome. The various hardware messages available are methods of the *w:control-register* flavor and should be sent to the *w:*control-register** variable, which is automatically set to point to the hardware attached to the Explorer system.

CAUTION: These methods may change incompatibly, without notice. DO NOT use these methods unless absolutely necessary.

w:*control-register* Variable
 The hardware attached to the Explorer system. This variable is automatically set by the system software and should not be changed.

:color-blanking Method of w:control-register
:set-color-blanking *value* Method of w:control-register

Returns or sets, respectively, the state of color blanking via the Video Attribute Register. *value* can be either `:on` or `:off` (or `t` or `nil`), indicating that the video signal for the color monitor is either on or off, respectively. When you blank a color monitor, you turn the video signal off.

For example, the following code completely turns off the video and then turns it back on:

```
(progn
  (send w:*control-register* :set-color-blanking :off)
  (sleep 2)
  (beep)
  (send w:*control-register* :set-color-blanking :on)
  (send w:selected-window :string-out
    "Your screen is not blanked." 0 nil w:red)
)
```

The following code displays whether the monitor is blanked or not blanked:

```
(setq save-data-stream (open "data" :direction :output))
(format save-data-stream "The color monitor is -A"
  (if (send w:*control-register* :color-blanking)
      "not blanked"
      "blanked"))
```

:monochrome-blanking Method of w:control-register
:set-monochrome-blanking *value* Method of w:control-register

Returns or sets, respectively, the state of blanking on a monochrome monitor via the Video Attribute Register. *value* can be either `:on` or `:off` (or `t` or `nil`), indicating that the video signal for the monochrome monitor is either on or off, respectively. These methods are exactly like the color blanking methods.

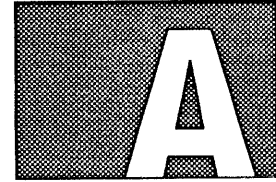
:monochrome-polarity Method of w:control-register
:set-monochrome-polarity *value* Method of w:control-register

Returns or sets, respectively, the monochrome polarity bit in the video control register. Toggling the value of this bit reverses the meaning of white and black on a monochrome monitor. In most cases, you should use the `:set-reverse-video` method rather than one of these methods to achieve a reverse video display.

For example, the following code toggles the value of the monochrome polarity:

```
(send w:*control-register* :set-monochrome-polarity
  (- 1 (send w:*control-register* :monochrome-polarity)))
```

OBSOLETE SYMBOLS



Introduction

A.1 The code described in this appendix is from previous releases; it is still available for use, but code marked as obsolete in this manual (that is, symbols with *[o]* in the syntax description line) will be phased out in future releases. This code should not be used for writing new programs.

See the *Release Information, Explorer System Software Release 3.0.0* for instructions on how to upgrade old programs.

Scrolling

A.2 Earlier Explorer releases included several ways the mouse could be used to scroll a window, each implemented by a mixin. In Release 3.0, many of these special scrolling mixins have been combined into a single, simpler mixin that gives a better user interface, a better programmer interface, and better code. This mixin is described in paragraph 11.8.1, Scroll Bars.

tv:basic-scroll-bar

[o] Flavor

Gives a window the ability to have a scroll bar. This flavor defines the following instance variables:

- **tv:scroll-bar** is a list describing the rectangle allocated for the scroll bar when the window provides margin space for a scroll bar. Otherwise, **tv:scroll-bar** is *nil*.
- **tv:scroll-bar-always-displayed** is non-*nil* if the scroll bar is displayed whenever margin space is provided for it, even if the mouse is not there.
- **tv:scroll-bar-in** is non-*nil* when the mouse is actually in this window's scroll bar.

tv:scroll-bar

[o] Instance Variable of **tv:basic-scroll-bar**

littable, gettable, settable.

A list that specifies whether to have a scroll bar, how big to make it, and where to place it. The argument to the initialization option or **:set-scroll-bar** can be *nil* for no scroll bar, *t* for a default scroll bar, or a small positive number, which requests a scroll bar of that width. The scroll bar occupies space in the margins of the window.

The method can change the inside size of the window because it can change the amount of space needed in the margin.

:enable-scrolling-p

[o] Method of **tv:basic-scroll-bar**

Returns *t* when the window has a scroll bar.

tv:scroll-bar-always-displayed

[o] Instance Variable of **tv:basic-scroll-bar**

littable, gettable, settable.

Specifies whether the scroll bar should always appear in the window. If the variable is non-*nil*, the bar of the scroll bar appears on the screen all the time, not only when the mouse is in it. Using the method updates the screen.

tv:scroll-bar-in [o] Instance Variable of **tv:basic-scroll-bar**

A Boolean value that indicates whether the mouse is actually in this window's scroll bar.

:mouse-buttons-scroll *mouse-char x y* [o] Method of **tv:basic-scroll-bar**

Invoked when the mouse is clicked in the scroll bar. The default definition—which you can redefine—provides the standard scrolling commands.

Arguments: *mouse-char* — A character with its mouse bit set, identifying the button clicked and how many times.

x, y — The position of the mouse cursor, relative to this window's outside edges, at the time of the click.

tv:flashy-scrolling-mixin [o] Flavor

Provides the ability to scroll the window a line at a time by pushing the mouse against the top or bottom edge. The mouse blinker changes to a thick up-or-down arrow when it is in the scroll bar.

This sort of scrolling is provided in the Zmacs editor and the Inspector. This flavor does not cause the words `More` above to appear, the way it does in the Inspector; that is done by **tv:margin-scroll-mixin**.

:flashy-scrolling-region *spec* [o] Initialization Option of **tv:flashy-scrolling-mixin**

Initializes the window for scrolling regions. *spec* specifies where the window scrolling regions should be, and it has this form:

((*top-height top-left top-right*)
(*bottom-height bottom-left bottom-right*))

where:

Each region always abuts the top or bottom edge of the window, overlapping the window's margin, but possibly extending into the inside of the window.

top-height and *bottom-height* indicate the number of pixels in height for the specified region.

top-left, *top-right*, *bottom-left*, and *bottom-right* give the sides of the region. These can be fixnums (positions relative to the window's left edge), flonums (fractions of the width of the window, with 0 at the left), or `:left` for the left edge or `:right` for the right edge.

tv:margin-scroll-mixin [o] Flavor

Provides mouse-sensitive regions in the top and bottom margins that display `More` below or `More` above if all of the text cannot be shown on the window. A mouse-click on the region scrolls an entire window of text. The **tv:margin-scroll-mixin** flavor requires **tv:margin-region-mixin** as well.

:margin-scroll-regions *region-list* [o] Initialization Option of **tv:margin-scroll-mixin**

Initializes a window for scrolling. Each element of *region-list* describes what to do with one of the two scrolling regions. An element of the list has the following form:

(keyword at-end-message more-message font-specifier)

where:

keyword is **:top** or **:bottom** and indicates which region this element describes.

at-end-message is an expression evaluated to produce the string to display in the region when you have reached the end of the object while scrolling in that direction. If **nil** or omitted, it defaults to **:top** or **:bottom** keywords.

more-message is another expression that is supposed to evaluate to a string to print when you have reached the end of the object while scrolling in that direction. **More above** and **More below** are the default strings.

font-specifier specifies the font to use. It defaults to **tr10i** if it is **nil** or omitted.

The *at-end-message* and *more-message* elements are the most commonly used.

tv:flashy-margin-scrolling-mixin [o] Flavor

Provides both flashy scrolling and margin scrolling, with the flashy scrolling areas overlaying the margin scrolling regions. You do not need anything else to use this flavor except **tv:basic-scroll-bar**.

tv:scroll-stuff-on-off-mixin [o] Flavor

Provides a scroll bar, flashy scrolling, and margin scrolling, and then makes them appear or disappear according to the value returned by the **:enable-scrolling-p** method.

:decide-if-scrolling-necessary [o] Method of **tv:scroll-stuff-on-off-mixin**

Makes the scroll bar and margin regions appear or disappear, if appropriate, using the **:enable-scrolling-p** method to decide whether they should be present. This method avoids displaying scrolling features, and using up screen space for them, when there is no place to scroll to.

The **:decide-if-scrolling-necessary** method is invoked automatically at certain times. The **:decide-if-scrolling-necessary** method should be invoked whenever the number of lines to scroll through has been changed, but before any associated redisplay is performed (because the redisplay to be performed may be different after this method finishes).

If the scroll bar and margin regions must be added or removed, then either the inside size or the outside size of the window must change. The **:adjustable-size-p** method is used to decide which size must be changed. If **:adjustable-size-p** returns non-**nil**, the inside size is preserved and the outside size is changed; otherwise, the outside size is preserved.

Changing the inside size can affect the window's redisplay calculations, and, for some windows, it may cause a redisplay within this method. You may want to invoke it inside of a `tv:with-sheet-deexposed` macro to avoid letting the user see gratuitous double redisplays, or to suppress the redisplay entirely if there is no bit-save array.

If the outside size is to be changed, and if changing the number of displayable items changes the height of the window, then these changes should be completed before invoking `:decide-if-scrolling-necessary`.

:adjustable-size-p [o] Method of `tv:scroll-stuff-on-off-mixin`

Determines how to adjust the window margin size. If `:adjustable-size-p` returns non-nil, the inside size is preserved; otherwise, the outside size is preserved.

Although the `tv:scroll-stuff-on-off-mixin` flavor does not define this method, the flavor requires users to define it.

tv:menu-margin-choice-mixin [o] Flavor

Required flavor: `tv:basic-menu`

Gives a menu the ability to have choice boxes in the margin. The `tv:menu-margin-choice-mixin` flavor is used in multiple menus.

Choice boxes appear in a single line in the bottom margin of the menu. Each choice box consists of a name followed by a little square or box. Clicking on the box activates the choice.

The `tv:menu-margin-choice-mixin` flavor adapts `tv:margin-choice-mixin` for use in menus.

:menu-margin-choices *items* [o] Initialization Option of `tv:menu-margin-choice-mixin`
Settable.

Sets the list of choice box items. *items* is the list of menu items. The items look and work exactly like menu items, and clicking on one has the same effect. The difference is in how and where they display.

The following code shows the default value, which provides a single choice box and implements the values returned by `tv:multiple-menu-choose`.

```
(( "Do It" :eval (values (funcall-self :highlighted-values) t)))
```

tv:margin-choice-menu [o] Flavor

An instantiable menu flavor that also allows margin choices.

tv:momentary-margin-choice-menu [o] Flavor

An instantiable momentary menu flavor that also allows margin choices. Menus produced by the `tv:momentary-margin-choice-menu` flavor disappear when the user is done with the menu.

Graphics

A.3 The following symbols are obsolete symbols used to draw graphics.

tv:graphics-mixin A.3.1 The following paragraphs describe the methods defined by **tv:graphics-mixin** that draw graphics on a window. These methods are obsolete and should not be used; instead, you should use the equivalent methods of **w:graphics-mixin**. In most cases, the names are identical. For example, the **:draw-point** method of **w:graphics-mixin** is functionally equivalent to the **:draw-point** method of **tv:graphics-mixin**.

tv:graphics-mixin [o] Flavor
 Required flavors: **tv:sheet**, **tv:essential-window**
 Provides graphics output operations for windows. For new code, use **w:graphics-mixin** instead.

:point x y [o] Method of **tv:graphics-mixin**
 Returns the value of the pixel located at the window coordinates specified by the *x* and *y* arguments. If the coordinates are outside the interior of the window, the returned value is 0; in other words, clipping causes a pixel to have a value of 0.

:draw-point x y &optional (alu tv:char-aluf) [o] Method of **tv:graphics-mixin**
 (*value -1*)
 Draws a pixel at the window coordinates specified by *x* and *y*. **:draw-point** combines the new value with the existing window pixel value at the specified coordinates according to the specified ALU function. The default for *value*, *-1*, is a pixel with all its bits set to 1.

:draw-line x1 y1 x2 y2 [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf draw-end-point*)
 Draws a line on the window from (*x1*, *y1*) to (*x2*, *y2*) using *alu*. *draw-end-point* and determines whether the end point, *x2,y2*, is drawn. If several connecting lines are to be drawn, specifying a value of **nil** draws the connecting points correctly.

:draw-lines alu x0 y0 x1 y1... xn yn [o] Method of **tv:graphics-mixin**
 Draws a line connecting each pair of coordinates. For example, a line connects *x0,y0* to *x1,y1* and another line connects *x1,y1* to *x2,y2*. This process continues until a line is drawn from *-xn -1,yn -1* to *xn,yn*. Each connecting point is drawn only once, and the point specified by *xn,yn* is not drawn.

:draw-dashed-line x0 y0 x1 y1 [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf*) (*dash-spacing 20*)
 (*space-literally-p nil*) (*offset 0*) (*dash-length* (**floor** *dash-spacing 2*))
 Draws a dashed line on the window.

Arguments: *x0*, *y0* — The window coordinates where **:draw-dashed-line** begins drawing the line.
x1, *y1* — The window coordinates where **:draw-dashed-line** stops drawing the line.

alu — The **:draw-dashed-line** method draws the line on the window according to the *alu* specified; that is, each pixel on the line is turned on or off as determined by the *alu* and the pixels at each window coordinate.

dash-spacing — The distance, in pixels, from the beginning of a dash to the end of the space following the dash.

space-literally-p — Determines whether to adjust the spacing between the dashes. If *space-literally-p* is nil, the spacing is adjusted so that the dashes fit evenly into the length of the line. If *space-literally-p* is non-nil, the dashes are spaced exactly as specified, even though the line may end in the spacing between dashes.

offset — How far, in pixels, from *x0,y0* the first dash starts and how soon before *x1,y1* the last dash ends. If *offset* is 0, the first dash starts on *x0,y0* and the last dash ends on *x1,y1*. *offset* is specified as the number of pixels.

dash-length — The actual length of the dash. If *dash-length* is not specified, the default dash length is calculated as half the value specified by *dash-spacing*.

:draw-curve *x-array y-array* [o] Method of tv:graphics-mixin
&optional *end (alu tv:char-aluf) closed-p*

Draws a sequence of line segments on the window.

The **:draw-curve** method is slightly different from the **:draw-lines** method in that the arguments specifying the coordinates are in two arrays rather than in an argument list. The **:draw-curve** method is also more general because of the flexibility allowed by the *end* and *closed-p* arguments.

Arguments: *x-array, y-array* — The window coordinates connecting all the points. **:draw-curve** draws a line connecting each pair of coordinates; for example, a line connects the first pair of *x-array,y-array* coordinates to the second pair of *x-array,y-array* coordinates, and another line connects the second pair of *x-array, y-array* coordinates to the third pair of *x-array,y-array* coordinates. This process continues until a line is drawn from the next-to-last pair of *x-array,y-array* coordinates to the last pair of *x-array,y-array* coordinates. Each connecting point is drawn only once, and the point specified by the last *x-array,y-array* pair is not drawn.

end — The number of line segments drawn.

alu — The **:draw-curve** method draws the line on the window according to the *alu* specified; that is, each pixel on the line is turned on or off as determined by the *alu* and the pixels at each window coordinate.

closed-p — If the value of the *closed-p* argument is non-nil, then a line is drawn connecting the last point to the first point.

:draw-wide-curve *x-array y-array width* [o] Method of tv:graphics-mixin
&optional *end alu closed-p*

Identical to **:draw-curve**, except that *width* is added. *width* specifies how wide, in pixels, the lines are when drawn.

:draw-triangle *x1 y1 x2 y2 x3 y3* [o] Method of tv:graphics-mixin
&optional *(alu tv:char-aluf)*

Draws a filled triangle whose vertices are at the window coordinates specified by the *x1, y1, x2, y2, and x3, y3* arguments.

:draw-circle *center-x center-y radius* [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf*)
 Draws a hollow circle with its center at *center-x* and *center-y*, with a radius of *radius*.

:draw-filled-in-circle *center-x center-y radius* [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf*)
 Like **:draw-circle** but **:draw-filled-in-circle** also fills in the circle after the circle is drawn.

:draw-circular-arc *center-x center-y radius start-theta end-theta* [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf*)
 Draws a circular arc (a portion of a hollow circle) on a window. *start-theta* and *end-theta* determine where the arc is drawn and must be specified in radians.

The point specified by 0 radians is the rightmost part of the arc with the radians increasing as the arc is drawn in a counterclockwise direction. If *start-theta* is larger than *end-theta*, the arc passes through 0 to *end-theta*.

:draw-filled-in-sector *center-x center-y radius start-theta end-theta* [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf*)
 Operates in the same way as **:draw-circular-arc**, except that **:draw-filled-in-sector** draws the arc and fills in the circular sector defined by the arc.

:draw-regular-polygon *x1 y1 x2 y2 n* [o] Method of **tv:graphics-mixin**
 &optional (*alu tv:char-aluf*)
 Draws a regular, filled polygon on a window.

Arguments: *x1, y1, x2, y2* — Window coordinates that specify one side of the polygon. Because the polygon is a regular object, all sides are the same length, and all interior angles are equal.

n — The number of sides for the polygon. *n* also determines whether the polygon is drawn in a clockwise or a counterclockwise direction. If the value of *n* is positive, **:draw-regular-polygon** draws the polygon in a clockwise direction. If the value of *n* is negative, **:draw-regular-polygon** draws the polygon in a counterclockwise direction.

:draw-cubic-spline *px py z* &optional *curve-width* [o] Method of **tv:graphics-mixin**
 (*alu tv:char-aluf*) (*c1 :relaxed*) *c2*
p1-prime-x p1-prime-y pn-prime-x pn-prime-y

Draws a cubic spline curve that passes through a sequence of points. The **:draw-cubic-spline** method draws a smooth-line curve through the points; this method is different from **:draw-curve**, which draws lines to connect the points.

Arguments: *px, py* — Arrays that contain the window coordinates for the points the curve connects. The number of points is determined by the length of *px*.

The points are computed so that they match the position and first derivative (slope) of each point.

Because there are no derivatives for the end points, you can specify the boundary conditions for these points. *c1* and *c2* determine the boundary conditions for the starting and ending points, respectively.

z — the number of points to use when drawing the curved line between each *px,py* pair. The **:draw-cubic-spline** method uses **:draw-curve** to draw the curved line between each *px,py* point.

curve-width — If the *curve-width* argument is specified, **:draw-cubic-spline** uses the **:draw-wide-curve** method to draw the curved line.

alu — The default value for the *alu* argument is **tv:char-aluf**.

c1, c2 — The *c1* and *c2* arguments specify how the **:draw-cubic-spline** method specifies the derivative at the end points. The possible values for *c1* and *c2* are **:relaxed**, **:clamped**, **:cyclic**, and **:anticyclic**:

- The **:relaxed** value makes the derivative at this point 0.
- **:clamped** allows the caller to specify the derivative. The arguments *p1-prime-x* and *p1-prime-y* specify the derivative at the starting point and are used only when *c1* is **:clamped**. The *pn-prime-x* and *pn-prime-y* arguments specify the derivative at the ending point and are used only if *c2* is **:clamped**.
- **:cyclic** specifies that the derivatives for both end points are equal. The *p1-prime-x* and *p1-prime-y* arguments determine the derivatives. If *c1* is specified as **:cyclic**, then *c2* is ignored. If you use **:cyclic** to draw a closed curve through *n* points, then the *px* and *py* arrays must contain *n + 1* points. The first and last points in the *px* and *py* arrays must be equal.
- **:anticyclic** makes the derivative of the starting point the negative of the derivative of the ending point. The *p1-prime-x* and *p1-prime-y* arguments determine the derivatives. If *c1* is specified as **:anticyclic**, then *c2* is ignored.

p1-prime-x, p1-prime-y — When **:clamped** is specified for the *c1* argument, values passed to *p1-prime-x* and *p1-prime-y* specify the derivative for the starting point. If **:clamped** is not specified for *c1*, *p1-prime-x* and *p1-prime-y* are ignored.

p1-prime-x and *p1-prime-y* define a vector that pulls the first part of the spline in a certain direction when the spline is drawn. The vector is the same as that of a line segment between the window coordinates 0,0 and *p1-prime-x,p1-prime-y*. The direction of the vector is the same as that of the line segment just described. The first part of the spline is pulled in the direction of the vector, with the magnitude of the pull relative to the length of the vector. The larger the vector, the farther the spline is pulled.

pn-prime-x, pn-prime-y — When **:clamped** is specified for *c2*, *pn-prime-x* and *pn-prime-y* specify the derivative for the ending point. If **:clamped** is not specified for *c2*, *pn-prime-x* and *pn-prime-y* are ignored.

The effects of *pn-prime-x* and *pn-prime-y* are the same as the effects of *p1-prime-x* and *p1-prime-y*, except that *pn-prime-x* and *pn-prime-y* affect the last part of the spline.

tv:stream-mixin A.3.2 You should use **w:graphics-mixin** instead of **tv:stream-mixin**.

:draw-char *char font x y* &optional (*alu tv:char-aluf*) Method of **tv:stream-mixin**
 Draws *char* of *font* using *alu* where the upper left corner of *char* is at window coordinates (*x,y*).

:draw-rectangle *width height x y* &optional (*alu tv:char-aluf*) Method of **tv:stream-mixin**
 Draws a filled rectangle of size *width* by *height* whose upper left corner is placed at the window coordinates (*x,y*).

gwin:draw-mixin A.3.3 You should use **w:graphics-mixin** instead of **gwin:draw-mixin**.

gwin:draw-mixin [o] Flavor
 Required flavor: **tv:minimum-window**
 Required methods: **:transform**, **:transform-point**, **:untransform-point**
 Contains the methods used to draw graphics objects in a window.

:allow-interrupts? *t-or-nil* [o] Initialization Option of **gwin:draw-mixin**
Gettable, settable. Default: **nil**
 Sets a flag that allows the drawing of a picture list to be interruptable. **nil** specifies that the entire picture list is to be drawn.

:draw-arc *x-center y-center x-start y-start* [o] Method of **gwin:draw-mixin**
 &optional (*arc-angle 360*) (*thickness 1.*) (*color black*) (*alu normal*)
 (*num-points 29*)
 Draws a hollow arc.

:draw-circle *x-center y-center radius* [o] Method of **gwin:draw-mixin**
 &optional (*thickness 1.*) (*color black*) (*alu normal*) (*num-points 29*)
 Draws a hollow circle.

:draw-filled-arc *x-center y-center x-start y-start* [o] Method of **gwin:draw-mixin**
 &optional (*arc-angle 360*) (*color black*) (*alu normal*) (*num-points 29*)
 (*draw-edge t*)
 Draws a solid arc.

:draw-filled-circle *x-center y-center radius* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*alu normal*) (*num-points 29*) (*draw-edge t*)
 Draws a solid circle.

:draw-filled-rectangle *left top width height* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*alu normal*) (*draw-edge t*)
 Draws a solid rectangle.

:draw-filled-triangle *x1 y1 x2 y2 x3 y3* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*alu normal*) (*draw-third-edge nil*)
 (*draw-second-edge t*) (*draw-first-edge t*)
 Draws a solid triangle.

- :draw-filled-triangle-list** *triangle-list* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*alu combine*)
 Draws a set of solid triangles. The vertices are specified as six-element lists that are the elements of the *triangle-list* parameter.
- :draw-line** *from-x from-y to-x to-y* [o] Method of **gwin:draw-mixin**
 &optional (*thickness 1.*) (*color black*) (*alu normal*)
 Draws a line.
- :draw-picture-list** *items* &optional *world* [o] Method of **gwin:draw-mixin**
 Draws a list of graphics entities in the window. Clipping is performed on each item. A check is performed to see if the item is too small to draw in detail or if the item is so small that it will not be drawn. A Boolean value is returned; it indicates whether the entities were drawn without interruption.
- :draw-polyline** *x-points y-points* [o] Method of **gwin:draw-mixin**
 &optional (*thickness 1.*) (*color black*) (*num-points* the minimum of the number of active array elements of *x-pints* and *y-points*) (*alu normal*)
 Draws a polyline.
- :draw-raster** *x y raster start-x start-y width height* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*alu normal*)
 Draws a raster image in the window by copying the pixel data. This is similar to a bitblt operation except that the starting location is in world coordinates and the image is clipped.
- :draw-rect** *left top width height* [o] Method of **gwin:draw-mixin**
 &optional (*thickness 1.*) (*color black*) (*alu normal*)
 Draws a hollow rectangle.
- :draw-solid-polygon** *x-center y-center x-points y-points* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*num-points* array elements of *x-points*) (*alu normal*)
 Draws a filled, concave polygon. This polygon has no inward-pointing vertices.
- :draw-string** *font text x y* [o] Method of **gwin:draw-mixin**
 &optional (*color black*) (*left 0.*) (*tab 8.*) (*scale 1.*) (*alu combine*)
 Draws a string of text in the specified font. The returned values are the x and y coordinates of the cursor after the string is drawn.
- :draw-triangle** *x1 y1 x2 y2 x3 y3* [o] Method of **gwin:draw-mixin**
 &optional (*thickness 1.*) (*color black*) (*alu normal*)
 Draws a hollow triangle.
- gwin:min-dot-delta** [o] Instance Variable of **gwin:draw-mixin**
Inittable, gettable, settable. Default: 6.
 Minimum size that an object can be and still be drawn in detail.
- gwin:min-nil-delta** [o] Instance Variable of **gwin:draw-mixin**
Inittable, gettable, settable. Default: 2.
 The minimum size that an object can be and still be drawn.

:undraw-picture-list *items* [o] Method of **gwin:draw-mixin**
 Undraws a list of graphics entities in the window. Clipping is performed on each item. A check is performed to see if any item was so small that it was not drawn at all. A Boolean value is returned; it indicates whether the entities were removed without interruption.

gwin:create-gwin-fonts &optional (*fonts-to-convert* ***default-gwin-fonts***) Function
 Converts a Lisp machine font to a raster font for the graphics window system. The argument can be one font description or a list of font descriptions. A font description has the following form:

(**fonts:symbol-name** *w:new-symbol-name printable-name-string*)

For example, the following is a font description:

(**fonts:cmr18** *w:cmr18-font* "Roman 18")

Primitives A.3.4

CAUTION: The subprimitives discussed in the following paragraphs do not perform error checking.

Some of the following subprimitives can accept an array or a sheet array. In window system applications, the argument is usually a sheet, but any suitable two-dimensional numeric array will do.

tv:draw-char *font char x y alu sheet-or-array* [o] Function
 Draws *char* of *font* on *sheet-or-array* using *alu*. The upper left corner of *char* begins at (*x,y*). **tv:draw-char** does not perform clipping.

sys:%draw-char *font char x y alu sheet-or-array* [o] Function
 Like **tv:draw-char**, except that **sys:%draw-char** is the actual microcoded primitive. **sys:%draw-char** does not use the indexing table for a wide font, so when **sys:%draw-char** is used on a wide font, the character specified by *char* is not the character you want to draw. For most purposes, you should use **tv:draw-char** rather than **sys:%draw-char**.

sys:%draw-rectangle *width height x-bitpos y-bitpos alu sheet-or-array* [o] Function

tv:%draw-rectangle-clipped *width height x-bitpos y-bitpos alu sheet-or-array* [o] Function

tv:%draw-rectangle-inside-clipped *width height x-bitpos y-bitpos alu sheet-or-array* [o] Function

Draws a rectangle of size *width* by *height* using *alu* on *sheet-or-array*. The upper left corner of the rectangle corresponds to the coordinates *x-bitpos*, *y-bitpos* relative to the upper left corner of the window (or array).

sys:%draw-rectangle performs no clipping; **tv:%draw-rectangle-clipped** performs clipping on the outside edges of the sheet; **tv:%draw-rectangle-inside-clipped** performs clipping on the inside edges of sheet.

sys:%draw-line *x0 y0 x y alu draw-end-point-p sheet-or-array* [o] Function

Draws a line from the sheet coordinates *x0,y0* to *x,y* on *sheet-or-array* using *alu*. The end point, *x,y*, is drawn only if *draw-end-point-p* is non-nil. **sys:%draw-line** does not perform clipping.

sys:%draw-filled-raster-line *x0 y0 x y alu draw-end-point-p sheet-or-array* [o] Function

Draws a line from the sheet coordinates *x0,y0* to *x,y* on *sheet-or-array* using *alu*. The end point, *x,y*, is drawn only if *draw-end-point-p* is non-nil. **sys:%draw-line** does not perform clipping.

sys:%draw-triangle *x1 y1 x2 y2 x3 y3 alu sheet-or-array* [o] Function

sys:%draw-filled-triangle *x1 y1 x2 y2 x3 y3 alu sheet-or-array* [o] Function

Draws an unfilled or filled triangle, respectively, with the vertices at the specified coordinates. Pixels are considered as being between coordinate points, with the left pixel between *x* values 0 and 1 and between *y* values 0 and 1. **sys:%draw-triangle** does not perform clipping.

Menus

A.4 The following paragraphs describe obsolete symbols used to produce menus.

Flavors and Methods A.4.1

tv:menu-execute-mixin [o] Flavor

Defines the **:execute** method to process a menu item according to the rules described previously.

:execute *item* [o] Method of **tv:menu-execute-mixin**

Processes *item*, computing and returning the value according to the rules described previously. **:execute** determines everything about the meaning of a menu item, except where it affects displaying the menu. By redefining the **:execute** method, you can implement new types of menu items. Because the keywords **:no-select**, **:documentation**, and **:font** determine how the menu is displayed, the overall format of the **:execute** method must be as described.

:execute-no-side-effects *item* [o] Method of **tv:menu-execute-mixin**

Processes *item*, computing and returning the value, provided that this can be done without side effects. If computing the value might possibly have side effects (such as for item types **:eval**, **:funcall**, **:kbd**, **:window-op**, **:menu**, and **:menu-choose**), the value is not computed, and **nil** is returned.

This method is typically used to find the item in a given item list that returns a particular value if selected.

tv>window-hacking-menu-mixin [o] Flavor

Provides for the **:window-op** item type by implementing the **:execute-window-op** method. This implementation involves remembering the mouse position and the window under the mouse at the time the menu is exposed.

Functional Interface A.4.2 The `tv:menu-choose` function provides an easy interface to menus.

`tv:menu-choose` *item-list* &optional (*label* nil) (*near-mode* :mouse) [o] Function
 (*default-item* nil) (*superior* tv:mouse-sheet)

Pops up a menu and allows the user to make a choice with the mouse. When the choice is made, the menu disappears and the chosen item is executed. The value of that item is returned as the first value of `tv:menu-choose`, and the item itself is returned as the second value.

If the user moves the mouse blinker well outside of the menu, the menu disappears and `tv:menu-choose` returns `nil`.

NOTE: If the user moves the mouse blinker just outside the menu, the menu does not disappear. This facility prevents inadvertent deexposure of a menu.

- Arguments:*
- item-list* — A list of items as described in paragraph 14.2.1, Menu Items.
 - label* — A string to be displayed at the top of the menu, or `nil` to specify the absence of a label.
 - near-mode* — Where to put the menu. It must be an acceptable argument to `:expose-near` method.
 - default-item* — The item over which the mouse should be positioned initially. This argument allows the user to select that item without moving the mouse. If *default-item* is `nil`, the mouse is initially positioned in the center of the menu.
 - superior* — The sheet of which the menu should be an inferior.

Geometry A.4.3 The following initialization options and methods manipulate the geometry of a menu.

`:geometry` *list* [o] Initialization Option of `tv:menu`
`:set-geometry` &optional *columns rows inside-width* [o] Method of `tv:menu`
inside-height max-width max-height

Sets the geometry (the constraints) from the arguments. The `:geometry` initialization option uses a list of six elements. The `:set-geometry` method uses six separate arguments rather than a list of six elements; this arrangement allows you to omit some of the arguments. The menu may change its shape and redisplay when its geometry changes.

For `:set-geometry`, an explicit argument of `nil` makes the corresponding aspect of the geometry unconstrained. An omitted argument or an argument of `t` leaves the corresponding aspect of the geometry the way it is (if unconstrained, it remains so).

`:geometry` [o] Method of `tv:menu`
`:current-geometry` [o] Method of `tv:menu`

Returns a list of six items. For `:geometry`, these items are the constraints, with `nil` in unspecified positions. For `:current-geometry`, these items correspond to the actual current state of the menu. The first four elements, which

are sufficient to describe the current state, are never `nil`. The last two elements, which can be `nil`, are the constraint values for the maximum width and height.

`:rows` *n-rows* [o] Initialization Option of `tv:menu`
`:columns` *n-columns* [o] Initialization Option of `tv:menu`

Set the number of rows or columns, respectively, to the value specified by the argument.

`:fill-p` *t-or-nil* [o] Initialization Option of `tv:menu`
Gettable, settable.

Specifies whether to use a filled format (`t`) rather than a columnar format (`nil`).

`:default-font` *font* [o] Initialization Option of `tv:menu`
Settable. Default: the standard font for the `:menu` font-purpose keyword for the screen under the menu

Sets the default font and is used to display items that do not specify a font.

`:set-edges` *left top right bottom &optional option* [o] Method of `tv:menu`

Sets the current position and size of the menu by setting the edges of the menu.

Arguments: *left* — The position of the left edge of the menu, in pixels, from the left edge of the menu's superior.

top — The position of the top edge of the menu, in pixels, from the top edge of the menu's superior.

right — The position of the right edge of the menu, in pixels, from the left edge of the menu's superior.

bottom — The position of the bottom edge of the menu, in pixels, from the top edge of the menu's superior.

option — Makes the specified size a permanent constraint for the menu. The *option* argument can be `:temporary` or `:nil`. If *option* is `:temporary`, the menu is redisplayed with the specified edges for now, but once it is redisplayed for any reason, the permanent constraints (or lack of them) reemerge.

Ordinary Menus A.4.4 The following paragraphs discuss the basic and mixin flavors and their associated initialization options, instance variables, methods, and resources for the ordinary kinds of menus. These flavors cannot be instantiated themselves but are useful to know about. Other kinds of menus are discussed in later paragraphs.

`tv:basic-menu` [o] Flavor

The basic flavor on which all other menu flavors are built. All the methods documented as being of `tv:menu` are really defined by this flavor.

`tv:item-list` [o] Instance Variable of `tv:basic-menu`

The item list of the menu.

- tv:last-item** [o] Instance Variable of **tv:basic-menu**
 The last item selected with a mouse click in this menu, or **nil** if none has been selected yet. **tv:last-item** is used for positioning a pop-up menu.
- tv:current-item** [o] Instance Variable of **tv:basic-menu**
 The item at which the mouse is pointing, or **nil**.
- tv:chosen-item** [o] Instance Variable of **tv:basic-menu**
 Set for a particular item each time that item is selected. A program can be put into a wait state by setting **tv:chosen-item** to **nil** and waiting for it to become non-**nil**.
- tv:geometry** [o] Instance Variable of **tv:basic-menu**
 A list of six elements representing the geometry (constraints) of the menu.

NOTE: Remember that *momentary* is the obsolete name for a pop-up menu. The name was changed to make it more meaningful.

- tv:basic-momentary-menu** [o] Flavor
 Produces a menu that is often referred to as a pop-up menu, which is only momentarily on the screen. A **:choose** method on a menu of this flavor positions the menu near the mouse cursor. When the user selects an item in the menu or alternatively moves the mouse out of the menu, the menu disappears and deactivates. The mouse then returns to where it was when the menu appeared, and the **:choose** method returns the chosen item or **nil**.

The following flavors produce useful menus and are all built on the **tv:basic-menu** flavor.

- tv:menu** [o] Flavor
 The same as **tv:basic-menu** with borders and a label on top. **tv:menu** defaults to no label, but you can specify a label using the **:label** init-plist option or the **:set-label** method.
- tv:momentary-menu** [o] Flavor
tv:momentary-menu &optional (*superior tv:mouse-sheet*) [o] Resource
tv:basic-momentary-menu mixed with other flavors to make this window instantiable. The resource is a resource of momentary menus. *superior* specifies the superior of the **tv:momentary-menu** resource.

- tv:temporary-menu** [o] Flavor
 Produces a menu that is a temporary window. When a menu is exposed, **tv:temporary-menu** saves the bits of the windows underneath the menu. **tv:temporary-menu** does not produce a momentary menu, and therefore the menu is not exposed or deexposed automatically.

You use a temporary menu rather than a momentary menu when you want to pop up a menu and make several choices from it before deexposing the

menu, or if you do not want to allow the user the option of not choosing anything by moving the mouse out of the window.

tv:momentary-window-hacking-menu [o] Flavor

Produces a momentary menu using the **tv:window-hacking-menu-mixin** flavor.

The following methods and initialization options are useful on menus of any flavor. Methods and initialization options that specifically affect the shape of the menu and the arrangement of items are listed in paragraph A.4.3, Geometry.

:item-list *item-list* [o] Initialization Option of tv:menu
Gettable, settable.

Sets the list of items (choices) for a menu. Setting the item list with the method recomputes the geometry and redisplay the menu. *item-list* specifies the list of items for the menu. (See paragraph 14.2.1, Menu Items, for a discussion of how to specify menu items.)

:choose [o] Method of tv:menu

Exposes the menu if it is not already exposed, waits until the user selects an item with the mouse, and executes the selection. The resulting value is returned. A momentary menu returns **nil** from **:choose** if the mouse is moved out of it, and in any case pops down before returning.

:execute *item* [o] Method of tv:menu

Once a menu item is selected, performs the appropriate side effects and returns the appropriate value. For most kinds of menus, the **:execute** method is invoked automatically as part of the **:choose** method, but command menus require the user program to invoke **:execute** explicitly if this method is desired. (See paragraph A.4.5, Command Menus.)

:move-near-window *window* [o] Method of tv:menu

Exposes the menu above or below *window*, giving the menu the same width as the window. If the menu is too large to fit within *window*, scrolling can be enabled.

:center-around *x y* [o] Method of tv:menu

Implemented by all windows, but menus handle it a little differently. The window is positioned so that the last item chosen appears at the coordinates specified by the *x* and *y* arguments (in the superior), if possible. If displaying the menu as specified would cause part of the menu to be outside of its superior, the menu is offset slightly to keep it inside its superior. The actual coordinates of the center of the appropriate item are returned (you might want to put the mouse there). Momentary menus use this method to put the menu in such a place that the mouse blinker appears over the last item chosen.

:current-item [o] Method of tv:menu

Returns the item the mouse is currently pointing at, or **nil** if none.

- :chosen-item** [o] Method of **tv:menu**
 Returns the item that has been chosen by the mouse and is being communicated back to the controlling process.
- :last-item** [o] Method of **tv:menu**
Settable.
 Returns or sets the item that was chosen by the mouse the last time this menu was used. When a momentary menu is exposed near the mouse cursor by the **:choose** method, **:last-item** puts the mouse cursor over this item so that it is easy to choose the item again.
- :column-row-size** [o] Method of **tv:menu**
 Returns two values: the width of a column in bits and the height of a row in bits.
- :item-cursorpos** *item* [o] Method of **tv:menu**
 Returns two values, as **:read-cursorpos** does, giving the x and y coordinates of the center of the displayed representation of *item*. The result is **nil** if the item is scrolled off the display.
- :item-rectangle** *item* [o] Method of **tv:menu**
 Returns four values: the coordinates of the rectangle enclosing the displayed representation of the specified item. The result is **nil** if the item is scrolled off the display. The returned coordinates are the coordinates of the left, top, right, and bottom of the rectangle. They include a one-pixel margin around the item.
- :menu-draw** [o] Method of **tv:menu**
 Draws the menu's display. The system automatically invokes **:menu-draw** when required, and **:menu-draw** should not be used in application programs. However, user-defined menu flavors can redefine the **:menu-draw** method or add methods to it.
- :mouse-buttons-on-item** *buttons-down-mask* [o] Method of **tv:menu**
 With the default definition, records the chosen item and processes the item type **:buttons** when it is used. The mouse process invokes the **:mouse-buttons-on-item** method when the mouse blinker is positioned on an item and a mouse button is clicked. **:mouse-buttons-on-item** does everything that should be done by the mouse process when a mouse button is clicked.

 The **tv:current-item** instance variable or the **:current-item** method can be used to find out which item the mouse is on.

Command Menus

A.4.5 The menus described so far are driven by the **:choose** method; that is, the program decides when it is time for the user to choose something in the menu. In some applications, the user should decide when to choose something from a menu. For example, in Peek, the user can select a new mode with the menu at any time, but Peek cannot spend all its time waiting for the user to do this.

The command menu is designed for such applications. When an item in a command menu is chosen, the menu puts a blip into the application's I/O buffer. The blip is a list like the following:

```
(:menu item button-mask menu)
```

where:

item is the menu item that was clicked on.

button-mask indicates which mouse button was used (as in `tv:mouse-last-buttons`).

menu is the menu that was clicked on, in case you are using more than one.

This list can be read as an input character with the `:any-tyi` method from any other window sharing the same input buffer.

Usually, a command window is part of a team of windows managed by a single process and sharing a single input buffer. Menu clicks generate input that is read in a single stream together with mouse clicks on the other windows and keyboard input. For example, Peek and the Inspector both use command menus in this way. Once the controlling process reads the blip, the process can execute (`funcall menu :execute item`) if the process wishes the item to be processed in the usual way for menu items.

tv:command-menu-mixin [o] Flavor
tv:command-menu [o] Flavor

Implements command menus. `tv:command-menu` is `tv:command-menu-mixin` mixed with `tv:menu` to make it instantiable.

tv:io-buffer [o] Instance Variable of `tv:command-menu`
inltable, gettable, settable.

The I/O buffer where a command menu stores a blip when an item is selected.

tv:command-menu-abort-on-deexpose-mixin [o] Flavor

When a command menu built on the this flavor is deexposed, the command menu automatically clicks on its Abort item. In other words, the `:deexpose` method for this flavor searches the item list for an item whose displayed representation is "ABORT" (case is not significant). If such an item is found, a blip is sent to the input buffer claiming that that item was clicked on with the left mouse button.

Dynamic Item List Menus **A.4.6** *Dynamic item list menus* dynamically recompute the item list at various times. Whenever the program makes an explicit request to use the menu, the menu checks automatically to see whether its item list has changed.

tv:abstract-dynamic-item-list-mixin [o] Flavor
:update-item-list [o] Method of `tv:abstract-dynamic-item-list-mixin`

The flavor causes a menu to invoke the `:update-item-list` method at various times. The `:update-item-list` method receives no arguments, and its value is ignored; `:update-item-list` updates the item list if appropriate.

The **tv:abstract-dynamic-item-list-mixin** flavor does not *define* the **:update-item-list** method, however. For each use of the **tv:abstract-dynamic-item-list-mixin** flavor, you should define the **:update-item-list** method to execute **:set-item-list** if the item list changes and update the item list as desired.

NOTE: The **:update-item-list** method can be invoked in various processes, so your definition should use only global variables and data structures it can find from the menu itself.

tv:dynamic-item-list-mixin [o] Flavor

Provides for a Lisp expression—a form—that is evaluated to get the menu's item list. This form is in the **tv:list-pointer** instance variable. The **:update-item-list** method is defined to evaluate the form and set the item list to the form's value.

tv:item-list-pointer [o] Instance Variable of tv:dynamic-item-list-mixin
Inittable, gettable, settable.

The Lisp expression evaluated to recompute the current item list.

The following menu flavors—except the **tv:dynamic-multicolumn-mixin** flavor—are combinations of **tv:dynamic-item-list-mixin** with other flavors.

tv:dynamic-momentary-menu [o] Flavor
tv:dynamic-momentary-window-hacking-menu [o] Flavor

Produces a temporary dynamic menu that disappears when an item is chosen. The **tv:dynamic-momentary-window-hacking-menu** flavor also allows the user to specify a **:window-op**.

tv:dynamic-temporary-menu [o] Flavor
tv:dynamic-temporary-command-menu [o] Flavor
tv:dynamic-temporary-abort-on-deexpose-command-menu [o] Flavor

Produces a temporary menu that must be explicitly exposed and deexposed for the menu to appear and disappear. The **tv:dynamic-temporary-command-menu** flavor and the **tv:dynamic-temporary-abort-on-deexpose-command-menu** flavor also put mouse click information into the window's input buffer. Unlike using **tv:dynamic-temporary-command-menu**, deexposing a menu produced by **tv:dynamic-temporary-abort-on-deexpose-command-menu** does the same thing as deexposing a menu by clicking on the Abort item.

tv:dynamic-multicolumn-mixin [o] Flavor

When used with **tv:abstract-dynamic-item-list-mixin**, makes a menu of several columns, in which each column's items are independently and dynamically recomputed. The System menu is such a menu.

tv:column-spec-list [o] Instance Variable of **tv:dynamic-multicolumn-mixin**
lnittable, gettable, settable.

The column specification list. The value for this variable is a list; each element specifies one column of the menu, and looks like this:

(heading item-list-form options...)

where:

heading is a string to be displayed as a **:no-select** item at the top of the column.

item-list-form is a form to be evaluated to produce the list of items for the column. It should have no side effects and can be evaluated in any process.

options are modifier keywords and values, such as are found in menu items. These modifiers apply to the column heading only. The most useful of these modifiers is the **:font** keyword.

tv:dynamic-multicolumn-momentary-menu [o] Flavor
tv:dynamic-multicolumn-momentary-window-hacking-menu [o] Flavor

Instantiable, momentary mixtures of **tv:dynamic-multicolumn-mixin**.

tv:dynamic-multicolumn-momentary-window-hacking-menu also includes **tv>window-hacking-menu-mixin**. The System menu is an instance of **tv:dynamic-multicolumn-momentary-window-hacking-menu**.

tv:multicolumn-menu-choose *column-spec-list* [o] Function
&optional *label near-mode default-item superior*

Pops up a multicolumn menu and allows the user to choose one of the items using the mouse. When the choice is made, the menu disappears and the chosen item is executed. The value of that item is returned as the first value of **tv:multicolumn-menu-choose**, and the item itself is returned as the second value.

If the user moves the mouse blinker well outside of the menu, the menu disappears and **tv:multicolumn-menu-choose** returns **nil**.

NOTE: If the user moves the mouse blinker just outside the menu, the menu will not disappear. This facility prevents inadvertent deexposure of a menu.

Arguments: *column-spec-list* — The **tv:column-spec-list** instance variable.
label — A string to be displayed at the top of the menu, or **nil** (the default) to specify the absence of a label. The *label* argument becomes the menu label.
near-mode — Where to put the menu. The *near-mode* argument defaults to the list (**:mouse**) and must be an acceptable argument to the **:expose-near** method.

default-item — The item over which the mouse is initially positioned. This allows the user to select that item without moving the mouse. If *default-item* is *nil* or unspecified, the mouse is initially positioned in the center of the menu.

superior — The sheet of which the menu should be an inferior. The default is *tv:mouse-sheet*.

Multiple Menus A.4.7 A *multiple menu* asks the user to select any combination of menu items rather than a single item. The menu has a choice box (usually named Do It) at the bottom in addition to its menu items. Clicking on a menu item selects it or deselects it; the selected items are displayed in reverse video. Clicking on the Do It box indicates that the user has finished selecting items from the menu.

The *:choose* method on a multiple menu returns as its first value a list of the values of the items selected by the user.

tv:multiple-menu-choose *item-list* &optional *label near-mode highlighted-items superior* [o] Function

Pops up a menu and allows the user to choose any subset of the available items. The user finalizes the choice by clicking on the Do It box at the bottom of the menu. At this time, **tv:multiple-menu-choose** returns as its first value a list of the results of executing all the chosen menu items. The second value of **tv:multiple-menu-choose** is *t* in this case.

If the user moves the mouse well outside of the menu, the menu disappears, and the **tv:multiple-menu-choose** function returns *nil* for both values. The second value enables the caller to distinguish between a refusal to choose and choosing the empty set of items.

Arguments: *item-list* — A list of menu items.

label — A string to be displayed at the top of the menu, or *nil* (the default) to specify the absence of a label.

near-mode — Where to put the menu. This argument defaults to the list *(:mouse)* and must be an acceptable argument to the *:expose-near* method.

highlighted-items — A list of some of the same items as in *item-list*; these are the items to include, initially, in the set to be chosen. The user can add items to the set or remove items from the set.

The elements of *highlighted-items* must be *eq* in to items *item-list* for proper functioning.

superior — The sheet of which the menu should be an inferior. The default is *tv:mouse-sheet*, which is usually a screen.

Making Your Own Multiple Menu A.4.8 The **tv:multiple-menu-choose** function may not be adequate if you wish to keep the menu permanently exposed or if you wish to alter its behavior. You must then create a menu yourself. The following flavors create multiple menus.

tv:margin-multiple-menu-mixin [o] Flavor

Gives a menu the ability to have multiple items selected using the *:choose* method.

- tv:multiple-menu** [o] Flavor
 Produces a menu that behaves as described previously in the **tv:multiple-menu-choose** function discussion. This flavor is a combination of **tv:margin-multiple-menu-mixin** with **tv:menu**.
- tv:momentary-multiple-menu** [o] Flavor
 Produces a multiple menu that is also momentary.
- tv:momentary-multiple-menu &optional** (*superior tv:mouse-sheet*) [o] Resource
 A resource of momentary multiple menus, used by **tv:multiple-menu-choose**.
- :add-item** *item* [o] Method of tv:margin-multiple-menu-mixin
 Adds *item* to the item list of the multiple menu, which is initially unhighlighted. All the existing items remain and are highlighted if they already were.
- :set-item-list** *item-list* [o] Method of tv:margin-multiple-menu-mixin
 Sets the item list specified by *item-list* and redisplay the menu. The items are not highlighted unless they were highlighted previously. (See paragraph 14.2.1, Menu Items, for a discussion of specifying menu items.)
- :special-choices** *items* [o] Initialization Option of tv:margin-multiple-menu-mixin
 Equivalent to the **:menu-margin-choices** initialization option, which is provided by the component flavor **tv:menu-margin-choice-mixin**. **:special-choices** is provided for historical compatibility. The *items* argument is a list of menu items that specify the choice boxes desired and what to do if they are clicked on.
- tv:menu-highlighting-mixin** [o] Flavor
 Enables some of the menu items to be highlighted with reverse video. **tv:menu-highlighting-mixin** is used with menus of modes, where the modes currently in effect are highlighted. Highlighting the menu mode usually reflects the enabling or disabling of a mode.

 The **tv:margin-multiple-menu-mixin** flavor uses the **tv:menu-highlighting-mixin** flavor.
- tv:highlighted-items** [o] Instance Variable of tv:menu-highlighting-mixin
inittable, gettable, settable. Default: nil
 The list of items to be highlighted when the flavor is instantiated.
- :add-highlighted-item** *item* [o] Method of tv:menu-highlighting-mixin
:remove-highlighted-item *item* [o] Method of tv:menu-highlighting-mixin
 Enables or disables, respectively, the highlighting of the menu item identified by *item* when the menu is displayed.
- :highlighted-values** [o] Method of tv:menu-highlighting-mixin
:set-highlighted-values *list* [o] Method of tv:menu-highlighting-mixin
:add-highlighted-value *value* [o] Method of tv:menu-highlighting-mixin
:remove-highlighted-value *value* [o] Method of tv:menu-highlighting-mixin
 Analogous to the **:highlighted-items** methods of similar names, these methods refers to the items' values (that is, the result of executing them) instead of referring to items directly. For example, if the item list is an

association list with the elements (`string`, `symbol`), **:highlighted-values** uses `symbol`. These methods work only for menu items that can be executed without side effects, not for item types like `:eval`, `:funcall`, and so on.

The argument is the list of menu items.

Input

A.5

:any-tyi &optional *eof-action* [o] Method of `w:stream-mixin`

Reads and returns the next character of input from the window. If there is no input, **:any-tyi** waits until a character of input becomes available. The character comes from the window's input buffer if the buffer contains any characters; otherwise, the character comes from the keyboard.

:tyi &optional *eof-action* [o] Method of `w:stream-mixin`
:tyi-no-hang &optional *eof-action* [o] Method of `w:stream-mixin`
:any-tyi-no-hang &optional *eof-action* [o] Method of `w:stream-mixin`

These methods are similar to **:any-tyi** with the noted exceptions.

:tyi and **:tyi-no-hang** throw away any blips they receive; they continue reading until they find an actual character, then they return the character. Discarded blips are never seen as input. If there is no input available (that is, if the buffer is empty), **:tyi** waits until there is input, while **:tyi-no-hang** returns `nil`.

:any-tyi-no-hang returns `nil` immediately if the buffer is empty. This method is used by programs that first execute continuously until a key is pressed, then examine the key and decide what to do next.

:mouse-or-kbd-tyi [o] Method of `w:stream-mixin`
:mouse-or-kbd-tyi-no-hang [o] Method of `w:stream-mixin`

Like the **:tyi** and the **:tyi-no-hang** method, except that certain kinds of blips are not discarded and do count as input. All other blips (that is, all blips whose `car` is *not* the symbol **:mouse-button**) are discarded. If these methods read a blip whose `car` is the symbol **:mouse-button**, they return three values:

1. The third element (`caddr`) of the blip, which is always a `fixnum`.
2. The whole blip.
3. A character whose mouse bit is 1. This bit identifies the mouse button that was clicked. (See paragraph 11.4.2, Encoding Mouse Clicks as Characters.)

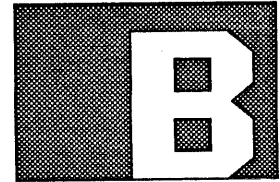
:list-tyi [o] Method of `w:stream-mixin`

The opposite of **:tyi**, **:list-tyi** returns only blips and discards real characters.

:untyi *character* [o] Method of `w:stream-mixin`

Puts the character just read back into the window's input buffer so that it is the next character returned by **:tyi**. This character must be the very last character that was read. It is illegal to perform two **:untyis** in a row. **:untyi** is used by parsers, such as `read`, that look ahead one character. *character* is the character just read from the window's input buffer.

CONVERTING APPLICATIONS TO COLOR



Introduction

B.1 This appendix describes how to convert to color an existing load band that contains software release 3.2 or later. Then, the section discusses the need to modify existing applications to run correctly on a color system.

Section 19 of this manual describes how to use color in the Explorer window system and gives details of the functions and methods for color. The *Explorer Programming Concepts* manual includes a section on the general theory behind color displays and how color is implemented on the Explorer system; the section also gives guidelines for color in a user interface.

Converting to Color

B.2 You need to convert from a monochrome system to a color system at least once. After you have converted, you should verify applications and then perform a **disk-save**. If you do not perform a **disk-save**, you will need to convert to the color system after each cold boot.

Requirements

B.2.1 Before you convert to color, your Explorer system should meet the following requirements:

- Have the following hardware installed:

- An Explorer II board
 - A color SIB
 - A color monitor

- Have software release 3.2 (or later) installed
- Have as few as possible additional applications loaded

NOTE: The conversion is more efficient if you convert a bare band to color and then load applications rather than converting a band with a number of applications loaded.

Converting the Load Band

B.2.2 To make use of the color monitor, change the load band to recognize and use color by executing the **w:convert-to-color** function. This function performs the following to each instance of a window that exists in the Explorer system:

- Creates a color map for the window.
- Sets the foreground and background colors of the window and of the window's label.

- Sets the window's `w:char-aluf` and `w:erase-aluf` to the correct value for color.
- Saves the contents of the 1-bit screen array, allocates an 8-bit screen array, copies the contents of the original screen array into the new, expanded array.
- Refreshes the window (that is, momentarily displays it) to ensure its screen array is correct.

In addition, any flavor instances based on `w:blinker-mixin` are changed incompatibly. The only problem caused is that you cannot readily change the color of the mouse or other existing blinkers. To do so, you can either remove the current instances and make new ones or kill the window that contains the blinker and create a new instance of that window.

Some applications may use windows that are not active when you execute the `w:convert-to-color` function. When you first expose such a window, a `:before :activate` method verifies that these windows are properly converted. Thus, some applications may appear sluggish when first accessed.

You can now add applications to the newly created color band. As you add each application, the Explorer system uses the updated flavors to create instances of blinkers and windows that include color.

Compatibility Issues

B.2 Most code that runs on a monochrome system also runs on a color system. The following items are known to cause problems; that is, if your code uses the following techniques, you must modify the code to run correctly on a color system.

- Using ALU arguments such as `w:alu-xor` to draw and erase on the display.
- Using `aref` to access the bit-save or screen array directly or drawing contents on the display by changing the bit-save or screen array and then displaying the array directly.
- Redefining any drawing method such as `:string-out` or `:draw-arc`.
- Using one of the previously unused instance variables of `tv:sheet`:

- `w:screen-array-text`
- `w:screen-array-graphics`
- `w:locations-per-line-text`

Each of these topics is discussed in more detail in the following paragraphs.

**Using the Correct
ALU Arguments**

B.2.1 When you draw an object (either text or graphics) on a window, you should use one of the new color ALU functions. The `w:convert-to-color` function translates the general functions `w:char-aluf` and `w:erase-aluf` to the appropriate color ALU function. However, if you used an explicit ALU function, you need to change the ALU function to the appropriate color ALU function.

- If you used `w:alu-ior` to draw output on a window, when you convert to color, the output is drawn in the same color as the background. It is therefore invisible. In most cases, instead of `w:alu-ior` you should use `w:combine` (`w:alu-transp`).
- If you used `w:alu-xor` to draw and erase output on a window, when you convert to color, the output is drawn in the background color on the foreground color. With text output, the ALU function produces an effect similar to reverse video. In most cases, instead of `w:alu-xor` you should use `w:alu-add` to draw on the display and `w:alu-sub` to erase.

If you used other monochrome ALU functions, choose an appropriate color ALU function from Table 19-3, Color ALU Operations.

Using Bit-Save Arrays

B.2.2 In most cases, you can allow the Explorer window system to allocate bit-save arrays automatically. You should never access the contents of a bit-save array directly. If you have done either of these in your code, that code will not work properly in a color system. A color system uses 8-bit bit-save arrays. A monochrome system uses 1-bit bit-save arrays.

If your application requires you to manipulate the bit-save array, you must supply duplicate code: one set for monochrome systems and another set for color systems. You can use the `w:color-system-p` function to distinguish between color and monochrome systems. In the following example, the code allocates a bit-save array for a window.

```
(defun arrays ()
  (let (my-bit-array my-window)
    (if (w:color-system-p w:default-screen) ; if t, the system is color
        (setq my-bit-array
              (make-array '(100 100) ; make an 8-bit array
                          :element-type '(unsigned-byte 8)))
        (setq my-bit-array
              (make-array '(100 100) ; if not t, make a 1-bit array
                          :element-type 'bit)))
    (setq my-window (make-instance 'w:window))
    (setf (w:sheet-bit-array my-window) my-bit-array)
    ; Put additional code to manipulate the bit array here
    (send my-window :expose)
    (send my-window :string-out "A text string")
  ))
```

**Using Drawing
Methods With
Added Arguments**

B.3.3 Methods that draw text now include an optional color argument. Graphics methods now include an optional texture argument. If you have redefined any of these methods, you should add the additional argument to your new definition.

Specifically, the following methods and functions for text output have changed:

:char-down	char-down
:char-up	char-up
:display-lozenged-string	display-lozenged-string
draw-char	
draw-string-internal	
:fat-string-out	fat-string-out
:insert-string	insert-string
:line-out	line-out
sheet-display-centered-string	
sheet-line-out	
sheet-string-out-explicit-1	
:string-out	string-out
:string-out-down	string-out-down
:string-out-centered-explicit	string-out-centered-explicit*
:string-out-explicit	
:string-out-up	string-out-up
:tyo	tyo

The following methods for graphic output have changed:

:draw-point	:draw-dashed-line
:draw-line	:draw-polyline
:draw-arc	:draw-filled-arc
:draw-circle	:draw-filled-circle
:draw-triangle	:draw-filled-triangle
:draw-rectangle	:draw-filled-rectangle
:draw-regular-polygon	:draw-filled-polygon
:draw-raster	:draw-string
:draw-cubic-spline	

In addition, the `sys:%draw-string` function has been removed.

**Using
Previously Unused
Instance Variables**

B.2.4 The following table lists the previously unused instance variables of `tv:sheet` that are used as instance variables for color. The other unused instance variables are reserved for future system expansion.

Unused Instance Variable	Macros to Access
<code>w:screen-array-text</code>	<code>w:sheet-color-map</code>
<code>w:screen-array-graphics</code>	<code>w:sheet-plane-mask</code>
<code>w:locations-per-line-text</code>	<code>w:sheet-color-reverse-video-state</code>

If you had been using these instance variables in your code, you should rewrite your code without those instance variables.

INDEX

Introduction

The indexes for this Explorer software manual are divided into several subindexes. Each subindex contains all the entries for a particular category, such as functions, variables, or concepts. The various subindexes for this manual and the pages on which they begin are as follows:

Index Name	Page
General	Index-2
Constants	<i>See</i> Variables
Defsubsts	<i>See</i> Functions
Flavors	Index-9
Functions	Index-13
Initialization Options	<i>See</i> Operations
Instance Variables	Index-17
Macros	<i>See</i> Functions
Methods	<i>See</i> Operations
Operations	Index-19
Special Forms	<i>See</i> Functions
Variables	Index-43

Alphabetization Scheme

The alphabetization scheme used in this index ignores package names and nonalphabetic symbol prefixes for the purposes of sorting. For example, the `rpc:*callrpc-retrys*` variable is sorted under the entries for the letter C rather than under the letter R.

Hyphens are sorted after spaces. Consequently, the `multiple` menu entry precedes the `multiple-choice` facility entry. However, the `apropos-flavor` entry precedes the `aproposb` entry, as follows:

```
apropos, 25-7
apropos-flavor, 25-9
aproposb, 25-9
```

Underscore characters are sorted after hyphens. Consequently, the `xdr-io` macro precedes the `xdr_destroy` macro.

General
A

ABORT key, implementing, 8-18, 8-20
 active inferiors set, 5-4
 active windows, 1-5
 ALU arguments, 7-2, 12-2—12-5
 color, 19-12—19-17
 examples of, 12-4—12-5
 general, 12-3—12-5
 w:char-aluf, 12-3
 w:erase-aluf, 12-3
 graphic methods, 12-13
 specifying ALUs with :edit-parameters menu,
 12-55
 ancestors. *See* superiors
 asynchronously intercepted characters,
 8-20—8-22
 audible feedback while typing, 8-26
 autoexposure, 5-19
 autoselection, 5-19—5-20

B

background process
 priority of, 6-8
 with a window, 6-14
 backquotes+~+, constraint frames and, 15-20
 baselines of fonts, 7-2, 9-7
 beep, 18-4—18-6
 bit arrays
 rotating, 12-28—12-30
 transferring, 12-25—12-30
 bit block transferring, 12-21—12-22,
 12-25—12-30
 bit-save arrays, 1-4, 5-1, 5-9—5-11
 output forced using w:sheet-force-access, 5-9
 black-on-white mode, 5-7—5-9
 blinkers
 See also mouse blinkers
 deselected visibility of, 10-2—10-4
 half period of, 10-3
 opening a, 10-1, 10-4
 position of, 10-4—10-6
 size of, 10-7
 types of, 10-7—10-11
 bitblt, 10-10
 box, 10-8
 character, 10-8—10-9
 hollow-rectangular, 10-8
 ibeam, 10-8
 magnifying, 10-10—10-11
 rectangular, 10-7—10-8
 reverse-character, 10-9
 visibility of, 10-2—10-4
 blips, 11-4
 general description of, 8-3

 processing within programs that do not
 check for blips, 8-3
 suggested actions, 8-3
 types of
 :choice-box of
 w:basic-choose-variable-values,
 14-57—14-58
 :execute of zwei:zmacs-frame, 18-11
 :line-area of w:line-area-text-scroll-mixin,
 16-11
 :menu of command menus, A-18
 mouse-sensitive scroll windows,
 17-11—17-12
 :timeout-execute of
 w:basic-mouse-sensitive-items, 16-9
 :variable-choice of
 w:basic-choose-variable-values,
 14-57—14-58
 blob (graphic object), 12-9
 border margin width, 3-2
 figure, 3-1
 operations to manipulate, 3-3
 borders, 3-2
 deleting from full-screen windows, 3-4
 functions to draw, 3-4
 operations to manipulate, 3-3
 shadow, 5-17
 BREAK key, implementing, 8-18, 8-20
 burying windows, 1-5, 5-22
 button masks, 11-4—11-5

C

char-exists table, 9-8
 character code, 8-25
 character height, 9-6
 character objects, decoding, 8-1
 character width, 9-7
 characters
 asynchronously intercepted, 8-20—8-22
 displaying, 7-3—7-5
 synchronously intercepted, 8-18—8-20
 choice box descriptor, 14-65—14-66
 choice boxes, 14-34
 choice facilities, 14-1
 choose-variable-values windows, 14-38—14-58
 defining your own variable type,
 14-52—14-53
 examples, 14-50—14-52
 complex, 14-51—14-52
 function surrounding the window call,
 14-51
 item modifiers for variables, 14-47
 margin choices, 14-50
 item modifiers for variables in, 14-46

- example of, 14-47
- making your own window, 14-53—14-58
- options keywords for, 14-48—14-49
- predefined variable types for, 14-43—14-45
- variables, 14-39—14-42
 - in linear format, 14-39
 - in table format, 14-41
- clicks, mouse. *See* mouse clicks
- clipping graphic images, 12-32
- Cohen-Sutherland algorithm, 12-38
- color
 - ALU arguments, 19-12—19-17
 - add operation, 19-14
 - add with saturate operation, 19-14
 - w:alu-add, 19-14, 19-16
 - w:alu-adds, 19-14, 19-16
 - w:alu-avg, 19-14, 19-16
 - w:alu-back, 19-15, 19-16
 - w:alu-max, 19-14, 19-16
 - w:alu-min, 19-14, 19-16
 - w:alu-sub, 19-14, 19-16
 - w:alu-subc, 19-14, 19-16
 - w:alu-transp, 19-15, 19-16
 - average operation, 19-14
 - background operation, 19-15
 - color versus monochrome ALU
 - arguments, 19-16
 - graphics methods, 12-2—12-5, 12-13, 19-12—19-17
 - list of, 19-16
 - max operation, 19-14
 - min operation, 19-14
 - monochrome ALU arguments used by
 - color ALU arguments in monochrome system, 19-16
 - specifying ALUs with :edit-parameters menu, 12-55
 - subtract operation, 19-14
 - subtract with clamping operation, 19-14
 - transparency operation, 19-15
 - truth table for monochrome displays, 19-17
- applications, converting to color, B-1—B-5
- background color, 19-3
 - initialization option and method for setting, 19-7
 - Profile variable, 19-18
 - transposing with foreground color (complement bow mode), 19-7
- blinker offset
 - initialization option and method for setting, 19-8
 - Profile variable, 19-18
- border color
 - initialization option and method for setting, 19-8
 - Profile variable, 19-18
- coding requirements for color, 19-1—19-2
- color argument for graphics methods, 12-10, 19-19
- color argument for text output methods, 7-5
- Color Look-Up Table (LUT), 19-2
- color map, 19-3
 - contents, 19-4
 - destruct elements, 19-4
 - functions that manipulate the color map, 19-8—19-12
 - named colors in the default color map table, 19-6
 - naming colors, 19-5
 - rainbow example, 19-8
 - ramp, 19-6
 - reserved colors, 19-5
 - table, 19-4
- compatibility issues when converting to color, B-2
 - bit-save arrays, B-3
 - correct ALU arguments, B-3
 - drawing methods with added arguments, B-4
 - previously unused instance variables, B-5
- complement bow mode (reverse video), 19-7
- converting a load band to color, B-1
- converting applications to color, B-1—B-5
- edge color
 - editing with :edit-parameters menu, 12-55
 - initialization option, 12-56
- fill color
 - editing with :edit-parameters menu, 12-55
 - initialization option, 12-56
- foreground color, 19-3
 - initialization option and method for setting, 19-7
 - Profile variable, 19-18
 - transposing with background color (complement bow mode), 19-7
- graphics methods
 - color argument, 12-10, 19-19
 - texture argument, 19-19
- how color works on a monitor, 19-2
- initialization options and methods used with color windows, 19-7—19-8
- label background color
 - initialization option and method for setting, 19-8
 - Profile variable, 19-18
- label foreground color
 - initialization option and method for setting, 19-8
 - Profile variable, 19-18
- list of drawing methods with added arguments, B-4
- load band conversion to color, B-1
- logical color, 19-3
- LUT, 19-2

- menu background color, Profile variable, 19-18
- menu foreground color, Profile variable, 19-18
- menu icons, 14-10
- menu label background color, Profile variable, 19-18
- menu label foreground color, Profile variable, 19-18
- menus, 14-24—14-25
- methods to control monitors directly, 19-23
- mouse documentation window background color, Profile variable, 19-18
- mouse documentation window foreground color, Profile variable, 19-18
- named colors, 19-6
- naming colors, 19-5
- physical color, 19-3
- pixel value, 19-3
- plane masks, 19-23
- printing color screens on monochrome printers, 19-21—19-23
 - gray patterns for printing the named colors, 19-22
- Profile variables, 19-18
- ramp, 19-6
- references to color in all Explorer manuals, 1-2
- reserved colors, 19-5
- reverse video (complement bow mode), 19-7
- RGB model, 19-2
- scroll bar shaded area, Profile variable, 19-18
- status line background color, Profile variable, 19-18
- status line foreground color, Profile variable, 19-18
- text output methods, color argument, 7-5
- texture argument for graphics methods, 12-11, 19-19
- transparency, 19-15
- using color, 19-1—19-24
- values for graphic methods in monochrome system, 12-12
- column specification list, 14-6—14-7
- columnar format of menus, 14-26
- command menus, 14-19—14-20
- conditions handled by character typeout, 7-3
- configuration, 15-4
- confirmation windows, 14-16—14-22
- constraint frame editor. *See* WINIFRED
- constraint frames, 15-1
 - backquotes and, 15-20
 - embedded configurations, 15-32—15-33
 - examples of, 15-15—15-21
 - flavors for, 15-13—15-15
 - keywords
 - for minimum and maximum sizes, 15-30—15-31
 - for size and position, 15-29
 - methods used with constraint frames, 15-34—15-36
 - pane-frame interaction, 15-35—15-36
 - selected pane of, 15-36
 - specifications for, 15-15
 - specifying panes and constraints, details, 15-22—15-28
 - stacking panes in, 15-6—15-7
- constraint frames+:+, examples of
 - graphics constraint frame, 15-16—15-17
 - horizontal constraint frame, 15-21
 - multiple-configuration constraint frame, 15-17—15-20
 - simple constraint frame, 15-15—15-16
- constraints, 15-4
- continuation of text. *See* horizontal wraparound
- current font, 7-2
 - definition of, 9-1
 - operations to manipulate, 9-4
- cursor motion, 7-15—7-17
- cursor position
 - figure, 7-15
 - of the window, 7-1

D

- deactivated windows, 1-5
- deexposed typeout actions, 7-9—7-11
 - :permit, screen manager updating partially visible windows, 5-21
- deexposed window, 1-4
- descendants. *See* inferiors
- deselected process, priority of, 6-8
- deselecting windows, 6-3
- display lists, 12-6
- display modes for scroll bars, 11-26
- displaying characters, pseudo-code for, 7-4—7-5
- documentation string, in the mouse
 - documentation window, 11-14—11-16
- dynamic menus, 14-22

E

- editor windows, 18-11—18-14
- end-of-line exceptions, 7-14—7-15
- end-of-page exceptions, 7-11—7-12
- entries in general scroll windows, 17-1—17-12
- exceptions
 - end-of-line, 7-14—7-15
 - end-of-page, 7-11—7-12
 - more processing, 7-12—7-14
 - output hold, 7-11—7-12
- exposed windows, 1-4
 - size constraints compared to their superiors, 4-2

F

fill patterns, 12-26—12-27
 filled format of menus, 14-26
 fixed-width fonts, 9-8
 flavor notation+~+, xxvi—xxvii
 flavors
 instantiable versus mixin+~+, xxviii
 mixing, 1-6
 order for mixing, 1-6
 overriding components, 1-6
 following blinkers, 10-1
 font descriptor, 9-9
 font indexing table for wide fonts, 9-11
 font map, 7-2
 definition of, 9-1
 operations to manipulate, 9-3
 font purposes, 9-2—9-3, 9-6
 operations to manipulate, 9-6
 font specifiers, 9-4—9-6
 fonts
 attributes of, 9-6—9-8
 commonly used, 9-2
 current, 7-2
 fixed-width, 7-2
 internal format, 9-9—9-11
 map. *See* font map
 purposes of, 9-6
 variable-width, 7-2
 foreground process. *See* selected process
 frame, 15-1
 frame editor. *See* WINIFRED
 frobboz+~+, margin item example, 3-12—3-14
 function keys, programming use of, 8-18
 function text scroll window, 16-5

G

garbage collection, inactive windows and, 5-3
 general scroll windows, 17-1—17-12
 automatically updating items, 17-8—17-10
 entry descriptors, 17-3—17-5
 inserting and deleting items, 17-7
 mouse-sensitive scroll windows, 17-11—17-12
 representation of items, 17-10—17-11
 root item, 17-6
 geometry of menus, 14-26—14-30
 global asynchronous characters, 8-22—8-26
 Glossary utility, example of constraint frame,
 15-2—15-3
 golden ratio used with menus, 14-30
 grabbing the mouse, 11-7
 graphic database, 12-34
 graphic objects, spline, 12-62—12-63
 graphics images
 clipping, 12-32
 compared with graphics objects, 12-1
 drawing using subprimitives, 12-30—12-34
 graphics objects, 12-34—12-36, 12-57—12-66

arc, 12-57—12-59
 background picture, 12-78
 characters, 12-70
 circle, 12-59—12-60
 compared with graphics images, 12-1
 font, 12-67
 line, 12-60—12-61
 polyline, 12-61—12-62
 raster
 character, 12-71—12-72
 object, 12-74—12-76
 ruler, 12-72—12-74
 spline, 12-64—12-65
 subpicture, 12-76
 text, 12-68—12-69
 triangle, 12-65—12-66
 vector character, 12-70—12-71
 graphs, labeling the y-axis, 7-6
 gray patterns, 12-26—12-27
 examples of, 12-12
 gridify points, 12-35, 12-49
 GWIN package, 2-1, 12-8

H

half period of blinkers, 10-3
 hierarchy of windows, 5-3—5-5
 highlighting menus. *See* multiple menus
 home position in a window, 7-16
 horizontal wraparound, 7-14
 hysteresis of a window, 11-8

I

I/O buffers, 8-13—8-17
 as input buffers, 8-16—8-17
 type-ahead and, 8-16
 icons, 14-7—14-11
 inferior list
 of windows, 5-3
 ordering the, 5-21—5-22
 inferiors set, active, 5-4
 input buffers, 8-2—8-3
 sharing among windows, 8-3
 input editor, 8-4—8-8
 activating an, 8-4—8-8
 implementing using :rubout-handler,
 8-7—8-8
 implementing using with-input-editing, 8-7
 options for (table), 8-6—8-7
 Inspector
 as an example of a text scroll window, 16-1
 flavors for, 16-12
 item generators for text scroll windows,
 16-6—16-8
 items
 in a text scroll window, 16-1—16-12
 in general scroll windows, 17-1—17-12
 in menus, 14-2—14-6

K

keyboard input buffer, 8-16
 keyclick, 8-26
 keypad, 8-26
 keys, trapping states of, 8-25—8-26
 Kill or Save Buffers menu, example of a
 multiple-choice window, 14-34
 killing a window, 1-5

L

Lisp Listener, windows, 18-10
 labels, 3-5—3-9
 as margin items, 3-1
 delaying redisplay of, 3-9
 examples of creating (figure), 3-7
 examples of creating boxed (figure), 3-8
 left kern, 9-7, 9-12—9-13
 line height, 9-6
 long-running process, with a window, 6-14
 lozenged characters, 7-8

M

-M format directive, 14-60—14-61
 main screen, 1-3, 1-4
 margin choices, 14-65—14-66
 for menus, 14-13, 14-21—14-22
 margin items
 example of creating, 3-12—3-14
 typical, 3-1
 margin region descriptors, 3-9
 margin regions, 3-9—3-14
 margins, 3-2
 menu items, 14-2—14-6
 menus, 14-2—14-33
 current item
 keystrokes to move, 14-18—14-19
 methods to reposition, 14-33
 examples of w:menu-choose, 14-3—14-10
 embedded menu, 14-5
 using :binding, 14-6
 filled versus columnar format, 14-26
 geometry of, 14-26—14-30
 item modifier keywords, 14-5
 keystrokes defined for, 14-18—14-19
 type value keywords for, 14-4
 microphone, recording sounds with,
 18-8—18-10
 more processing, 7-12—7-14
 mouse
 fast motion changing the shape of, 11-3
 grabbing the, 11-8—11-11
 handedness, setting, 11-4
 ownership of, 11-7—11-12
 tracking the, 11-1
 usurping the, 11-7
 warping the, 11-2

 windows and, 11-12—11-16
 mouse bit of a character, 11-6
 checking for, 8-1
 mouse blinkers, 11-1, 11-16—11-20
 reusable types, 11-18—11-19
 types of, 11-18
 mouse blips. *See* blips
 mouse characters
 creating a new glyph, 11-23
 mapping to new values, 11-19—11-20
 standard values for, 11-21—11-23
 mouse clicks, 11-4—11-7
 encoded as characters, 11-4
 encoding as characters, 11-5, 11-6—11-8
 incrementing keystates, 11-4
 processing clicks other than R2, 3-10
 scroll bar default operations, 11-26
 mouse cursor, position of, 11-1
 mouse documentation window, 1-3,
 18-15—18-18
 setting the string for, 11-14—11-16
 mouse glyphs. *See* mouse character
 mouse parameters, 11-3—11-4
 mouse process, 11-1
 mouse scrolling, 11-24—11-30
 mouse sheet, 11-1
 mouse-sensitive items, 14-58—14-65
 w:basic-mouse-sensitive-items, 14-61—14-65
 -M format directive, 14-60—14-61
 procedure for creating, 14-58—14-60
 multiple choose menus, function to create,
 14-15
 multiple menus, 14-20—14-21
 multiple-choice facility, 14-34—14-38

N

noises, making with Explorer sound chips,
 18-6—18-8
 notational conventions+ +, xxv—xxix
 notifications, 18-1—18-4

O

obsolete symbols, A-1—A-23
 offsets, screen arrays and window positions,
 5-11
 figure, 5-12
 orientation of a world, 12-6
 output, explicit positioning, 7-23—7-25
 output exceptions, 7-9—7-15
 end-of-line, 7-9—7-15
 end-of-page, 7-9—7-15
 more, 7-9—7-15
 output-hold, 7-9—7-15
 output on a window, anticipating the effects of,
 7-20
 output-hold exceptions, 7-11—7-12

overstriking of characters, 7-2

P

packages, window system, 2-1
 panes, 15-1
 Peek
 as an example of general scroll windows,
 17-1—17-2
 flavor for, 17-12
 showing the window stack, 5-2
 permanent menus, 14-22
 picture lists, 12-6, 12-10
 pixels, 5-7—5-9
 pop-up menus, 14-22
 precision of a world, 12-6
 priority of windows, 5-1
 process name in the status line, 18-16—18-22
 processes
 associating with a window, 6-13
 grabbing the mouse, 11-8—11-11
 input buffer for each, 8-2
 priorities of, 6-7
 usurping the mouse, 11-11—11-12
 windows and, 6-11

R

raster (graphic) fonts, 12-66
 raster height of fonts, 9-10
 raster width of fonts, 9-10
 resources
 w:inspect-frame-resource, 16-12
 w:menu, 14-19
 tv:momentary-menu, A-15
 tv:momentary-multiple-menu, A-22
 w:pop-up-finger-window, 18-14
 zwei:pop-up-standalone-editor-frame, 18-12
 w:pop-up-text-window, 18-14
 supdup:telnet-windows, 18-14
 w:temporary-choose-variable-values-window,
 14-54
 zwei:temporary-mode-line-with-borders-resou
 rce, 18-13
 w:temporary-multiple-choice-window, 14-38
 reverse video, 5-7
 right margin character (!), 7-14—7-15
 ring buffer, 8-13
 rubout handler. *See* input editor

S

screen arrays, 5-11
 screen manager, 5-1, 5-18—5-23
 delaying management, 5-22—5-23
 screens, 5-2—5-3
 definition of, 1-1
 main screen, 1-3, 1-4
 who-line screen, 1-3
 scroll bars, 11-25—11-29

 bumping the cursor against, 11-26
 scroll windows. *See* text scroll windows;
 general scroll windows; scroll bars
 scrolling, 11-24—11-30
 Select command, of the System menu,
 6-5—6-6
 selected window, 1-4, 6-1—6-2
 selecting windows
 using the SYSTEM or TERM keys, 6-6—6-9
 using the System menu Select command,
 6-5—6-6
 selection substitutes, 6-8—6-10
 typeout windows and, 6-9—6-10
 server names in the status line, 18-17—18-22
 shadow borders, 5-17
 sheets, definition of, 1-1
 sounds
 making with Explorer sound chips,
 18-6—18-8
 playing and recording, 18-8—18-10
 sprites, 12-40, 12-52—12-53, 12-54
 stack, window hierarchy, 1-5, 5-2
 stacking panes in a constraint frame+,,
 15-6—15-7
 status line, 1-3, 18-15—18-18
 status of a window (table), 6-10
 stream input operations, 8-8—8-12
 stream names in the status line, 18-16—18-22
 stream output, 7-5—7-9
 substitutes for selection, 6-8—6-10
 superiors of windows, 5-3
 synchronously intercepted characters,
 8-18—8-20
 SYSTEM key, 8-22
 adding and removing keystroke sequences,
 8-23—8-26
 System menu, 18-18—18-19
 mouse calling the, 11-16
 Select command, implementing, 6-5—6-6

T

tab size of windows, 7-26
 teams of windows, 6-4—6-8
 Telnet windows, 18-14—18-15
 temp locking, of windows under temporary
 windows, 5-18
 temporary windows, 5-16—5-18
 TERM key, 8-22
 adding and removing keystroke sequences,
 8-22—8-23
 text
 deleting, 7-19—7-20
 inserting, 7-19—7-20
 text scroll windows, 16-1—16-12
 function text scroll windows, 16-5—16-6

text scroll windows (continued)
 item generators, 16-6—16-8
 item list, 16-2—16-4
 keywords for the item generator function,
 16-7
 mouse-sensitive windows, 16-9—16-12
 tracking the mouse, 11-1
 trapping keystrokes, 8-25
 truncation, of characters at the end of a line,
 7-14
 TV package, 2-1
 type-ahead, 8-16, 8-23
 I/O buffers and, 8-16
 timeout, delaying redisplay after, 13-4
 timeout window, 13-1—13-5
 deactivated, 13-2—13-3
 selection substitutes and, 6-9—6-10
 windows with inferior, 13-3—13-4

U

usurping the mouse, 11-7, 11-11—11-12

V

variable-width fonts, 9-8
 vertical spacing (vsp), 9-6
 visible window, 1-4
 vsp, 9-6

W

W package, 2-1
 warping the mouse, 11-1
 white-on-black mode, 5-7—5-9
 who-line
See also mouse documentation window;
 status line
 screen, 1-3
 who-line screen, 1-3
 width of a window, inside versus outside
 (figure), 4-1
 window coordinates, 12-5
 direction of increase, 7-1
 window stack, 1-5
 window system, 1-1—1-4
 as a user interface, 1-4
 windows
 active, 1-5
 aliases for inferiors, 6-7
 as input streams, 8-1
 as instances of flavors, 1-5—1-6
 as output streams, 7-1
 associating processes with, not invoked by a
 SYSTEM key, 6-13—6-14
 basic flavors for, 2-3
 burying, 1-5

contents of, 5-9
 creating, 2-2
 deactivated, 1-5
 dedicated processes and, 6-11
 deexposed, 1-4
 deselecting, 6-3
 designing a, 1-12
 erasing contents of, 7-17—7-18
 exposed, 1-4, 5-1
 size constraints compared to their
 superiors, 4-2
 exposing, 5-12—5-13
 automatically, 5-19
 priority of windows for, 5-21—5-22
 finding, 18-21—18-22
 general choices among, 1-13—1-24
 greedy, 11-7
 hierarchy of, 5-2, 5-3—5-5
 home position in, 7-16
 inferior, 1-1
 inferior lists, sorting, 5-21—5-22
 killing, 1-5
 manipulating lists of, 5-5—5-7
 mouse handling and, 11-12—11-16
 overlapping
 figure, 5-1
 predicates to determine, 4-7
 position of an offset window, 5-12
 previously selected, 6-6—6-9
 priority of
 for exposure, 5-21—5-22
 negative, 5-22
 resources of, 18-20—18-21
 selected, 1-4, 6-1—6-2
 selecting, 6-2—6-4
 automatically, 5-19—5-20
 size of, using the mouse to specify the
 corners of a window, 11-10—11-11
 status of (table), 6-10
 tab size of, 7-26
 teams of, 6-4—6-8
 controlling selection, 6-8
 treated as a unit for selection, 6-6
 temporary, 5-16—5-18
 text scroll, 16-1—16-12
 timeout and, 7-1
 visible, 1-4, 5-1
 WINIFRED, 15-4—15-13
 editing code generated by, 15-11—15-12
 invoking, 15-5
 typical procedure when using, 15-5—15-11
 using the generated code in your
 application, 15-13
 world coordinates, 12-6—12-8
 wraparound of characters on a line, 7-14

Flavors**A**

- tv: abstract-dynamic-item-list-mixin, A-18
- w: alias-for-inferiors-mixin, 6-7
- gwin: arc, 12-57
- w: autoexposing-more-mixin, 7-14

B

- gwin: backgroundpic, 12-78
- gwin: basic-character-mixin, 12-70
 - w: basic-choose-variable-values, 14-53
- gwin: basic-cursor-mixin, 12-50
 - w: basic-frame, 15-13
- gwin: basic-graphics-mixin, 12-55
 - tv: basic-menu, A-14
 - tv: basic-momentary-menu, A-15
 - w: basic-mouse-sensitive-items, 14-61
 - w: basic-multiple-choice, 14-36
- tv: basic-scroll-bar, A-1
 - w: basic-scroll-window, 17-5
 - w: basic-typeout-window, 13-1
- gwin: bitblt-blinker, 12-51
 - w: bitblt-blinker, 10-10
 - w: blinker, 10-1
- gwin: block-cursor, 12-52
 - w: bordered-constraint-frame, 15-14
 - w: bordered-constraint-frame-with-shared-io-buffer, 15-14
 - w: borders-mixin, 3-3
 - w: bottom-box-label-mixin, 3-8
 - w: box-blinker, 10-8
 - w: box-label-mixin, 3-7

C

- w: cache-window, 12-46
- w: centered-label-mixin, 3-7
- w: character-blinker, 10-8
- w: choose-variable-values-pane, 14-53
- w: choose-variable-values-window, 14-53
- tv: command-menu, A-18
- tv: command-menu-abort-on-deexpose-mixin, A-18
- tv: command-menu-mixin, A-18
 - w: constraint-frame, 15-14
 - w: constraint-frame-with-shared-io-buffer, 15-14
 - w: current-item-mixin, 16-12
- gwin: cursor, 12-51

D

- w: delay-notification-mixin, 18-2
- w: delayed-redisplay-label-mixin, 3-9
- w: displayed-items-text-scroll-window, 16-12
- gwin: draw-mixin, A-9
 - tv: dynamic-item-list-mixin, A-19
 - tv: dynamic-momentary-menu, A-19

- tv: dynamic-momentary-window-hacking-menu, A-19
- tv: dynamic-multicolumn-mixin, A-19
- tv: dynamic-multicolumn-momentary-menu, A-20
- tv: dynamic-multicolumn-momentary-window-hacking-menu, A-20
- tv: dynamic-temporary-abort-on-deexpose-command-menu, A-19
- tv: dynamic-temporary-command-menu, A-19
- tv: dynamic-temporary-menu, A-19

E

- w: essential-scroll-mouse-mixin, 17-12
- w: essential-window-with-typeout-mixin, 13-3

F

- tv: flashy-margin-scrolling-mixin, A-3
- tv: flashy-scrolling-mixin, A-2
- gwin: font, 12-67
- w: frame-forwarding-mixin, 15-35
- w: full-screen-hack-mixin, 3-4
- w: function-text-scroll-window, 16-5

G

- tv: graphics-mixin, A-5
- w: graphics-mixin, 12-9
- gwin: graphics-window, 12-44
- gwin: graphics-window-mixin, 12-45
- gwin: graphics-window-pane, 12-44
- w: gray-deexposed-right-mixin, 5-20
- w: gray-deexposed-wrong-mixin, 5-20
- gwin: circle, 12-59

H

- w: hollow-rectangular-blinker, 10-8
- w: hysteretic-window-mixin, 11-8

I

- w: ibeam-blinker, 10-8
- w: inferiors-not-in-select-menu-mixin, 6-5
- w: initially-invisible-mixin, 5-20
- w: inspect-frame, 16-12
- w: interaction-pane, 15-36
- w: intrinsic-no-more-mixin, 13-4

K

- w: kbd-mouse-buttons-mixin, 11-6

L

- w: label-mixin, 3-5
- gwin: line, 12-60
- w: line-area-mouse-sensitive-text-scroll-mixin, 16-11
- w: line-area-text-scroll-mixin, 16-11
- w: line-truncating-mixin, 7-14
- w: lisp-interactor, 18-10
- w: lisp-listener, 18-10
- w: list-mouse-buttons-mixin, 11-6

w: listener-mixin, 18-10
 w: listener-mixin-internal, 18-10

M

w: magnifying-blinker, 10-10
 tv: margin-choice-menu, A-4
 w: margin-choice-mixin, 14-65
 tv: margin-multiple-menu-mixin, A-21
 w: margin-region-mixin, 3-9
 tv: margin-scroll-mixin, A-2
 tv: menu, A-15
 w: menu, 14-19
 tv: menu-execute-mixin, A-12
 tv: menu-highlighting-mixin, A-22
 tv: menu-margin-choice-mixin, A-4
 w: minimum-window, 2-3
 tv: momentary-margin-choice-menu, A-4
 tv: momentary-menu, A-15
 tv: momentary-multiple-menu, A-22
 tv: momentary-window-hacking-menu, A-16
 w: mouse-blinker-mixin, 11-17
 w: mouse-box-blinker, 11-18
 w: mouse-box-stay-inside-blinker, 11-18
 w: mouse-character-blinker, 11-18
 gwin: mouse-handler-mixin, 12-48
 w: mouse-hollow-rectangular-blinker, 11-18
 w: mouse-rectangular-blinker, 11-18
 w: mouse-sensitive-text-scroll-window, 16-9
 w: mouse-sensitive-text-scroll-window-without-click, 16-10
 w: multiple-choice, 14-38
 tv: multiple-menu, A-22

N

w: no-screen-managing-mixin, 5-19
 w: not-externally-selectable-mixin, 6-6
 w: notification-mixin, 18-2

P

tv: peek-frame, 17-12
 gwin: polyline, 12-61
 zwei: pop-up-standalone-editor-frame, 18-12
 w: pop-up-text-window, 18-14
 w: preemptable-read-any-tyi-mixin, 8-12
 w: process-mixin, 6-12

R

gwin: raster-character, 12-71
 gwin: raster-object, 12-74
 gwin: rectangle, 12-62
 w: rectangular-blinker, 10-7
 w: reset-on-output-hold-flag-mixin, 6-13
 w: reverse-character-blinker, 10-9
 gwin: ruler, 12-72

S

w: screen, 5-2
 w: scroll-bar-mixin, 11-26
 w: scroll-mouse-mixin, 17-12
 tv: scroll-stuff-on-off-mixin, A-3
 w: scroll-window, 17-5
 w: scroll-window-with-typeout, 17-7
 w: scroll-window-with-typeout-mixin, 17-7
 w: select-mixin, 6-2
 w: shadow-borders-mixin, 5-17
 w: sheet, 2-3
 w: show-partially-visible-mixin, 5-20
 gwin: spline, 12-64
 gwin: sprite-cursor, 12-52
 w: sprite-window, 12-46
 zwei: standalone-editor-frame, 18-11
 zwei: standalone-editor-window, 18-11
 w: stay-inside-blinker-mixin, 10-8
 w: stream-mixin, 8-3
 gwin: subpicture, 12-76

T

supdup: telnet, 18-14
 w: temporary-choose-variable-values-window, 14-54
 tv: temporary-menu, A-15
 zwei: temporary-mode-line-window-with-borders, 18-13
 w: temporary-multiple-choice-window, 14-38
 w: temporary-shadow-borders-window-mixin, 5-17
 w: temporary-window-mixin, 5-17
 gwin: text, 12-68
 w: text-scroll-window, 16-2
 w: text-scroll-window-empty-gray-hack, 16-6
 w: text-scroll-window-typeout-mixin, 16-6
 w: top-box-label-mixin, 3-8
 w: top-label-mixin, 3-7
 w: transform-mixin, 12-47
 gwin: triangle, 12-65
 w: truncating-pop-up-text-window, 18-14
 w: truncating-pop-up-text-window-with-reset, 18-14
 w: truncating-window, 7-15
 w: typeout-window, 13-2
 w: typeout-window-with-mouse-sensitive-items, 13-2

V

gwin: vector-character, 12-70

W

w: who-line-file-sheet, 18-17
 w: window, 2-3
 tv: window-hacking-menu-mixin, A-12
 w: window-with-typeout-mixin, 13-3
 gwin: world, 12-40

Z

zwei: zmacs-frame, 18-11

Functions
Numbers

- w: 12-hour-clock-setup, 18-16
- w: 24-hour-clock-setup, 18-16

A

- w: add-system-key, 8-23
- w: add-terminal-key, 8-22
- w: add-to-system-menu-column, 18-18
- w: add-timeout-item-type, 14-62
- w: add-window-type, 18-19
- w: adjust-by-interval, 14-17
- w: apropos-flavor+--, 18-21
- w: apropos-method+--, 18-22
- w: apropos-resource+--, 18-21
- w: await-window-exposure, 7-11

B

- beep, 18-5
- bitblt+--, 12-25
- w: black-on-white, 5-8

C

- gwin: calculate-string-motion, 12-39
- w: careful-notify, 18-1
- w: char-code+--, 8-1
- w: choose-process-in-error, 18-3
- w: choose-variable-values, 14-47
- w: choose-variable-values-process-message, 14-57
- w: close-all-servers, 18-18
- w: color-map-xxx, 19-4
- w: color-system-p, 19-1
- w: complement-bow-mode, 5-8
- w: copy-color-map, 19-11
- w: copy-speech, 18-9
- w: create-color-map, 19-10
- gwin: create-gwin-fonts, A-11
- w: create-w-fonts, 12-39
- w: current-color-lut-buffer, 19-12

D

- w: def-beep-function, 18-5
- w: def-beep-type, 18-5
- w: default-beep, 18-6
- w: define-mouse-char-mapping, 11-19
- w: defsound, 18-6
- w: defwindow-resource, 18-20
- w: delaying-screen-management, 5-23
- w: delete-from-system-menu-column, 18-19
- w: describe-servers, 18-18
- w: deselect-and-maybe-bury-window, 6-3

- w: determinant, 12-39
- w: display-font, 9-8
- w: dist, 12-36
- gwin: dist-from-rectangle, 12-36
- gwin: dist-from-segment, 12-36
- w: do-sound, 18-7
- w: download-color-lut-buffer, 19-12
- sys: %draw-char, A-11
- tv: draw-char, A-11
- w: draw-char, 7-7
- w: draw-char-down, 7-7
- w: draw-char-up, 7-7
- sys: %draw-character, 12-33
- sys: %draw-filled-raster-line, A-12
- sys: %draw-filled-triangle, A-12
- w: draw-icon, 14-8
- sys: %draw-line, 12-33, A-12
- sys: %draw-rectangle, 12-33, A-11
- tv: %draw-rectangle-clipped, A-11
- tv: %draw-rectangle-inside-clipped, A-11
- w: draw-rectangular-border, 3-4
- sys: %draw-shaded-raster-line, 12-33
- sys: %draw-shaded-triangle, 12-34
- sys: %draw-triangle, A-12

E

- w: expose-window-near, 5-14

F

- gwin: find-corresponding-y, 12-39
- w: find-process-in-error, 18-3
- w: find-window+--, 18-21
- w: find-window-of-flavor, 8-25
- w: flush-full-screen-borders, 3-4
- w: font-baseline, 9-11
- w: font-blinker-height, 9-11
- w: font-blinker-width, 9-11
- w: font-char-height, 9-11
- w: font-char-width, 9-11
- w: font-char-width-table, 9-12
- w: font-chars-exist-table, 9-13
- w: font-evaluate, 9-5
- w: font-indexing-table, 9-14
- w: font-left-kern-table, 9-12
- w: font-name, 9-11
- w: font-raster-height, 9-14
- w: font-raster-width, 9-14
- w: font-rasters-per-word, 9-14
- w: font-words-per-char, 9-14

G

- w: get-color-lut-buffer, 19-12

- w: get-display-type, 19-2
- printer: get-gray-scale-value, 19-22
- w: get-standard-font, 9-6
- w: get-visibility-of-all-sheets-blinkers, 10-5

I

- w: idle-lisp-listener, 18-10
- int-char+--, 8-1
- w: io-buffer-clear, 8-15
- w: io-buffer-empty-p, 8-14
- w: io-buffer-full-p, 8-14
- w: io-buffer-get, 8-15
- w: io-buffer-input-function, 8-13
- w: io-buffer-input-pointer, 8-13
- w: io-buffer-last-input-process, 8-14
- w: io-buffer-last-output-process, 8-14
- w: io-buffer-output-function, 8-13
- w: io-buffer-output-pointer, 8-13
- w: io-buffer-plist, 8-14
- w: io-buffer-push, 8-15
- w: io-buffer-put, 8-15
- w: io-buffer-record, 8-14
- w: io-buffer-record-pointer, 8-14
- w: io-buffer-size, 8-13
- w: io-buffer-state, 8-14
- w: io-buffer-unget, 8-15

K

- w: kbd-asynchronous-intercept-character, 8-21
- w: kbd-char-typed-p, 8-17
- w: kbd-default-output-function, 8-16
- w: kbd-intercept-abort, 8-18
- w: kbd-intercept-abort-all, 8-18
- w: kbd-intercept-break, 8-18
- w: kbd-intercept-error-break, 8-18
- w: kbd-io-buffer-get, 8-16
- w: kbd-snarf-input, 8-17
- w: kbd-wait-for-input-or-deexposure, 8-17
- w: kbd-wait-for-input-with-timeout, 8-17
- w: key-state, 8-25

L

- gwin: lines-intersect-p, 12-37
- gwin: load-picture, 12-39
- w: lock-sheet, 5-18

M

- w: make-blinker, 10-2
- w: make-color-map, 19-10
- w: make-default-io-buffer, 8-16
- w: make-font-purpose, 9-6
- w: make-gray, 12-26
- make-instance+--, 2-2

- w: make-io-buffer, 8-15
- w: make-sheet-bit-array, 12-25
- w: make-simple-icon, 14-7
- gwin: make-sprite-from-objects, 12-40
- w: map-over-exposed-sheet, 5-5, 5-6
- w: map-over-sheet, 5-6
- w: map-over-sheets, 5-6
- w: margin-region-area, 3-10
- w: margin-region-bottom, 3-10
- w: margin-region-function, 3-10, 3-11-3-12
- w: margin-region-left, 3-10
- w: margin-region-margin, 3-10
- w: margin-region-right, 3-10
- w: margin-region-size, 3-10
- w: margin-region-top, 3-10
- tv: menu-choose, A-13
- w: menu-choose, 14-11
- w: menu-compute-geometry, 14-30
- w: merge-shift-keys, 11-5
- w: mouse-buttons, 11-5
- w: mouse-call-system-menu, 11-16
- w: mouse-character-button-encode, 11-5
- w: mouse-confirm, 14-16
- w: mouse-default-handler, 11-13
- w: mouse-define-blinker-type, 11-18
- w: mouse-discard-clickahead, 11-3
- w: mouse-get-blinker, 11-19
- w: mouse-input, 11-12
- w: mouse-save-image, 12-27
- w: mouse-select, 11-16
- w: mouse-set-blinker, 11-17
- w: mouse-set-blinker-cursorpos, 11-13
- w: mouse-set-blinker-definition, 11-19
- w: mouse-set-sheet, 11-2
- w: mouse-set-sheet-then-call, 11-2
- w: mouse-set-window-position, 11-11
- w: mouse-set-window-size, 11-11
- w: mouse-specify-rectangle, 11-10
- w: mouse-standard-blinker, 11-17
- w: mouse-wait, 11-9
- w: mouse-wakeup, 11-8
- w: mouse-warp, 11-2
- w: mouse-y-or-n-p, 14-16
- tv: multicolumn-menu-choose, A-20
- w: multicolumn-menu-choose, 14-14
- w: multiple-choose, 14-34
- tv: multiple-menu-choose, A-21
- w: multiple-menu-choose, 14-15

N

- gwin: nearest-circle-pt, 12-36
- gwin: nearest-pt-on-arc, 12-36
- gwin: nearest-rectangle-pt, 12-36
- gwin: nearest-triangle-pt, 12-36
- w: noise, 18-7
- w: notify, 18-1

O

- gwin: off-window, 12-37
- w: open-all-sheets-blinkers, 10-5
- w: open-blinker, 10-4

P

- w: play, 18-9
- gwin: point-in-extents-p, 12-37
- gwin: point-in-polygon-p, 12-37
- zwei: pop-up-edstring, 18-12
- w: position-window-next-to-rectangle, 5-14
- w: prepare-sheet, 12-32
- w: preserve-substitute-status, 6-9
- print-graphics+~+, 12-40
- w: print-notifications, 18-2
- w: process-typeahead, 8-15
- w: process-who-line-documentation-list, 18-15

R

- gwin: rasterize-objects, 12-40
- w: read-any, 8-9
- w: read-any-no-hang, 8-9
- w: read-bit-array-file, 12-27
- w: read-color-lut-buffer, 19-12
- w: read-color-map, 19-11
- zwei: read-defaulted-pathname-near-window, 18-13
- w: read-list, 8-9
- w: read-mouse-or-kbd, 8-9
- w: read-mouse-or-kbd-no-hang, 8-9
- w: read-speech, 18-9
- w: rec, 18-9
- w: remap-mouse, 11-20
- w: remove-beep-function, 18-5
- w: remove-beep-type, 18-6
- w: remove-system-key, 8-23
- w: remove-terminal-key, 8-22
- w: rename-speech, 18-9
- w: reset-sound, 18-6
- w: rotate-90, 12-28
- w: rotate-180, 12-28
- w: rotate-270, 12-28

S

- w: save-speech, 18-9
- w: scroll-maintain-list, 17-8
- w: scroll-maintain-list-unordered, 17-8
- w: scroll-maintain-list-update-states, 17-10
- w: scroll-parse-item, 17-4
- w: scroll-string-item-with-embedded-newlines, 17-5
- w: sector-code, 12-37
- w: select-color-with-mouse, 19-11

- w: select-or-create-window-of-flavor, 8-25
- w: select-texture-with-mouse, 19-19
- w: set-number-of-who-line-documentation-lines, 18-15
- w: set-screen-standard-font, 9-5
- w: set-standard-font, 9-5
- w: set-visibility-of-all-sheets-blinkers, 10-5
- w: setup-keyboard-keyclick, 8-26
- w: setup-mouse-left-handed, 11-4
- w: setup-mouse-right-handed, 11-4
- w: sheet-backspace-not-overprinting-flag, 7-26
- w: sheet-baseline, 9-7
- w: sheet-bit-array, 5-10
- w: sheet-blinker-list, 10-6
- w: sheet-bottom-margin-size, 3-2
- w: sheet-bounds-within-sheet-p, 4-7
- w: sheet-calculate-offsets, 4-7
- w: sheet-char-aluf, 12-3
- w: sheet-char-width, 7-3
- w: sheet-clear, 7-18
- w: sheet-clear-char, 7-17
- w: sheet-clear-eof, 7-18
- w: sheet-clear-eol, 7-18
- w: sheet-color-map, 19-10
- w: sheet-compute-motion, 7-21
- w: sheet-contains-sheet-point-p, 4-7
- w: sheet-cr-not-newline-flag, 7-26
- w: sheet-current-font, 9-4
- w: sheet-cursor-x, 7-17
- w: sheet-cursor-y, 7-17
- w: sheet-deexposed-typeout-action, 7-10
- w: sheet-end-of-page-flag, 7-12
- w: sheet-erase-aluf, 12-3
- w: sheet-exposed-inferiors, 5-15
- w: sheet-exposed-p, 5-15
- w: sheet-following-blinker, 10-4
- w: sheet-font-map, 9-3
- w: sheet-force-access, 5-13, 7-11
- w: sheet-get-screen, 5-2
- w: sheet-height, 4-4
- w: sheet-inferiors, 5-4
- w: sheet-inside-bottom, 4-5
- w: sheet-inside-height, 4-5
- w: sheet-inside-left, 4-5
- w: sheet-inside-right, 4-5
- w: sheet-inside-top, 4-5
- w: sheet-inside-width, 4-5
- w: sheet-label-needs-updating, 3-9
- w: sheet-left-margin-size, 3-2
- w: sheet-line-height, 7-3
- w: sheet-line-out, 7-8
- w: sheet-me-or-my-kid-p, 5-5
- w: sheet-more-flag, 7-13

S

- tv: scroll-bar instance variable of tv:basic-scroll-bar, A-1
- tv: scroll-bar-always-displayed instance variable of tv:basic-scroll-bar, A-1
- tv: scroll-bar-in instance variable of tv:basic-scroll-bar, A-2

T

- gwin: tick-x-points instance variable of gwin:ruler, 12-72
- gwin: tick-y-points instance variable of gwin:ruler, 12-72
- w: time-until-blink instance variable of w:blinker, 10-3

X

- w: x-offset instance variable of windows, 4-6

Y

- w: y-offset instance variable of windows, 4-6

Instance Variables

B

- w: bit-array instance variable of windows, 5-10
- w: bits-per-pixel instance variable of w:screen, 5-8
- w: buffer instance variable of w:screen, 5-9
- w: buffer-halfword-array instance variable of w:screen, 5-9

C

- w: char-aluf instance variable of windows, 12-3
- w: char-width instance variable of windows, 7-3
- w: choice-value instance variable of w:basic-multiple-choice, 14-38
- tv: chosen-item instance variable of tv:basic-menu, A-15
- tv: column-spec-list instance variable of tv:dynamic-multicolumn-mixin, A-20
- tv: current-item instance variable of tv:basic-menu, A-15
- w: cursor-x instance variable of windows, 7-17
- w: cursor-y instance variable of windows, 7-17

D

- w: displayed-items instance variable of w:displayed-items-text-scroll-window , 16-12

E

- w: erase-aluf instance variable of windows, 12-3

G

- tv: geometry instance variable of tv:basic-menu, A-15

H

- tv: highlighted-items instance variable of tv:menu-highlighting-mixin, A-22

I

- tv: io-buffer instance variable of tv:command-menu, A-18
- tv: item-list instance variable of tv:basic-menu, A-14
- tv: item-list-pointer instance variable of tv:dynamic-item-list-mixin, A-19

L

- w: label instance variable of w:label-mixin, 3-5
- tv: last-item instance variable of tv:basic-menu, A-15
- w: line-height instance variable of windows, 7-3

M

- w: margin-choices instance variable of w:basic-multiple-choice, 14-38
- gwin: min-dot-delta instance variable of gwin:draw-mixin, A-10
- gwin: min-nil-delta instance variable of gwin:draw-mixin, A-10
- w: mouse-blinkers instance variable of w:screen, 11-19

R

- w: recursion instance variable of w:basic-frame, 15-14
- w: restored-bits-p instance variable of windows, 5-11

S

- tv: scroll-bar instance variable of tv:basic-scroll-bar, A-1
- tv: scroll-bar-always-displayed instance variable of tv:basic-scroll-bar, A-1
- tv: scroll-bar-in instance variable of tv:basic-scroll-bar, A-2

T

- gwin: tick-x-points instance variable of gwin:ruler, 12-72
- gwin: tick-y-points instance variable of gwin:ruler, 12-72
- w: time-until-blink instance variable of w:blinker, 10-3

X

- w: x-offset instance variable of windows, 4-6

Y

- w: y-offset instance variable of windows, 4-6

Operations

A

- :abort-on-deexpose initialization option of w:menu, 14-23
- :activate method of windows, 5-5
- :activate-p initialization option of windows and screens, 5-5
- :active-p method of w:basic-typeout-window, 13-3
- :active-p method of windows and screens, 5-5
- :add-asynchronous-character method of w:stream-mixin, 8-21
- :add-cursor method of gwin:graphics-window-mixin, 12-45
- :add-highlighted-item method of tv:menu-highlighting-mixin, A-22
- :add-highlighted-item method of w:menu, 14-20
- :add-highlighted-value method of tv:menu-highlighting-mixin, A-22
- :add-highlighted-value method of w:menu, 14-21
- :add-item method of tv:margin-multiple-menu-mixin, A-22
- :add-item method of w:menu, 14-22
- :add-server method of w:who-line-file-sheet, 18-17
- :add-stream method of w:who-line-file-sheet, 18-17
- :adjustable-size-p method of tv:scroll-stuff-on-off-mixin, A-4
- :adjustable-size-p method of w:basic-choose-variable-values, 14-56
- :alias-for-inferiors method of windows, 6-7
- :alias-for-selected-windows method of windows, 6-7
- :allow-interrupts? initialization option of gwin:draw-mixin, A-9
- :allow-interrupts? initialization option of w:graphics-mixin, 12-10
- :allow-interrupts? method of gwin:draw-mixin, A-9
- :allow-interrupts? method of w:graphics-mixin, 12-10
- :alu initialization option of gwin:basic-graphics-mixin, 12-56
- :alu initialization option of gwin:bitblt-blinker, 12-51
- :alu initialization option of gwin:sprite-cursor, 12-52
- :alu method of gwin:basic-graphics-mixin, 12-56
- :alu method of gwin:bitblt-blinker, 12-51
- :angle initialization option of gwin:arc, 12-58
- :angle method of gwin:arc, 12-58
- :any-tyi method of w:stream-mixin, A-23
- :any-tyi-no-hang method of w:stream-mixin, A-23
- :append method of gwin:text, 12-69
- :append-item method of w:text-scroll-window, 16-3
- :appropriate-width method of w:basic-choose-variable-values, 14-56
- :array initialization option of gwin:bitblt-blinker, 12-51
- :array initialization option of gwin:sprite-cursor, 12-52
- :array initialization option of w:bitblt-blinker, 10-10
- :array method of w:bitblt-blinker, 10-10
- :arrest method of w:select-mixin, 6-13
- :asynchronous-character-p method of w:stream-mixin, 8-21
- :asynchronous-characters initialization option of w:stream-mixin, 8-21
- :await-exposure method of windows, 7-11

B

- :background-color initialization option of windows, 19-7
- :background-color method of windows, 19-7
- :backspace-not-overprinting-flag initialization option of windows, 7-26
- :backward-char method of windows, 7-16
- :baseline method of windows, 9-7
- :beep method of windows, 18-4
- :bit-array initialization option of w:cache-window, 12-46

- :bit-array initialization option of w:sprite-window, 12-46
- :bitarray initialization option of gwin:raster-object, 12-75
- :bitblt method of w:stream-mixin, 12-26
- :bitblt-from-sheet method of w:stream-mixin, 12-26
- :bitblt-within-sheet method of w:stream-mixin, 12-26
- :blink method of gwin:bitblt-blinker, 12-51
- :blink method of w:blinker, 10-6
- :blinker-deselected-visibility initialization option of windows, 10-3
- :blinker-flavor initialization option of windows, 10-5
- :blinker-height initialization option of gwin:font, 12-67
- :blinker-height method of gwin:font, 12-67
- :blinker-list method of windows and screens, 10-6
- :blinker-offset initialization option of blinkers, 19-8
- :blinker-offset method of blinkers, 19-8
- :blinker-p initialization option of gwin:graphics-window, 12-45
- :blinker-p initialization option of gwin:graphics-window-pane, 12-45
- :blinker-p initialization option of windows, 10-5
- :blinker-width initialization option of gwin:font, 12-67
- :blinker-width method of gwin:font, 12-67
- :border-color initialization option of gwin:text, 12-68
- :border-color initialization option of w:borders-mixin, 19-8
- :border-color method of gwin:text, 12-68
- :border-color method of w:borders-mixin, 19-8
- :border-margin-width initialization option of w:borders-mixin, 3-3
- :border-margin-width method of w:borders-mixin, 3-3
- :borders initialization option of gwin:graphics-window, 12-45
- :borders initialization option of gwin:graphics-window-pane, 12-45
- :borders initialization option of w:borders-mixin, 3-3
- :borders method of w:borders-mixin, 3-3
- :bottom initialization option of windows, 4-2
- :bottom-flag initialization option of gwin:sprite-cursor, 12-53
- :bottom-flag method of gwin:sprite-cursor, 12-53
- :bottom-limit initialization option of gwin:sprite-cursor, 12-53
- :bottom-limit initialization option of gwin:world, 12-44
- :bottom-limit method of gwin:sprite-cursor, 12-53
- :bottom-limit method of gwin:world, 12-44
- :bottom-margin-size method of windows, 3-2
- :bottom-reached method of w:basic-typeout-window, 13-2
- :bottom-shadow-width initialization option of w:shadow-borders-mixin, 5-17
- :bury method of windows, 5-22

C

- :calculate-extents method of gwin:world, 12-44
- :call method of w:select-mixin, 6-13
- :call-mini-buffer-near-window method of zwei:temporary-mode-line-window-with-borders, 18-14
- :center-around method of tv:menu, A-16
- :center-around method of w:menu, 14-33
- :center-around method of windows, 4-6
- :change-of-default-font method of windows, 9-5
- :change-of-size-or-margins method of windows, 4-6
- :character initialization option of gwin:cursor, 12-51
- :character initialization option of w:character-blinker, 10-9
- :character method of gwin:cursor, 12-51
- :character method of w:character-blinker, 10-9
- :character-height initialization option of windows, 4-3
- :character-size method of gwin:font, 12-67
- :character-width initialization option of windows, 4-3
- :character-width method of windows, 7-20

:character-x-offset initialization option of w:reverse-character-blinker, 10-9
 :character-x-offset method of w:reverse-character-blinker, 10-9
 :character-y-offset initialization option of w:reverse-character-blinker, 10-9
 :character-y-offset method of w:reverse-character-blinker, 10-9
 :characters initialization option of gwin:font, 12-67
 :characters method of gwin:font, 12-67
 :choice-types initialization option of w:basic-multiple-choice, 14-37
 :choice-types method of w:basic-multiple-choice, 14-37
 :choose method of tv:menu, A-16
 :choose method of w:basic-multiple-choice, 14-38
 :choose method of w:menu, 14-31
 :chop method of gwin:text, 12-69
 :chosen-item method of tv:menu, A-17
 :chosen-item method of w:menu, 14-25
 :clear-between-cursorposes method of windows, 7-18
 :clear-char method of windows, 7-17
 :clear-eof method of windows, 7-18
 :clear-eol method of windows, 7-18
 :clear-input method of w:stream-mixin, 8-11
 :clear-screen method of windows, 7-18
 :clear-string method of windows, 7-17
 :close-all-servers method of w:who-line-file-sheet, 18-18
 :closedp initialization option of gwin:polyline, 12-61
 :closedp initialization option of gwin:spline, 12-64
 :closedp method of gwin:polyline, 12-61
 :color-blanking method of w:control-register, 19-24
 :color-map method of windows, 19-10
 :column-row-size method of tv:menu, A-17
 :column-row-size method of w:menu, 14-29
 :column-spec-list initialization option of tv:dynamic-multicolumn-mixin, A-20
 :column-spec-list initialization option of w:menu, 14-24
 :column-spec-list method of tv:dynamic-multicolumn-mixin, A-20
 :column-spec-list method of w:menu, 14-24
 :columns initialization option of tv:menu, A-14
 :columns initialization option of w:menu, 14-29
 :command-characters initialization option of w:menu, 14-26
 :command-menu initialization option of w:menu, 14-20
 :command-menu method of w:menu, 14-20
 :complement-bow-mode method of windows, 5-8, 19-7
 :compute-margins method of windows, 3-13
 :compute-motion method of windows, 7-21
 :comtab initialization option of standalone editor windows, 18-11
 :configuration initialization option of w:basic-constraint-frame, 15-34
 :configuration method of w:basic-constraint-frame, 15-34
 :constraints initialization option of all constraint frame flavors, 15-22
 :copy method of gwin:basic-graphics-mixin, 12-56
 :copy method of gwin:polyline, 12-62
 :copy method of gwin:raster-object, 12-76
 :copy method of gwin:ruler, 12-74
 :copy method of gwin:spline, 12-65
 :copy method of gwin:subpicture, 12-78
 :copy method of gwin:text, 12-69
 :cr-not-newline-flag initialization option of windows, 7-26
 :create-and-add-entity method of gwin:world, 12-42
 :create-and-add-entity-to-front method of gwin:world, 12-42
 :create-pane method of w:basic-constraint-frame, 15-34
 :crosshair-mode initialization option of gwin:mouse-handler-mixin, 12-48
 :crosshair-mode method of gwin:mouse-handler-mixin, 12-48

- :cur-height initialization option of gwin:raster-object, 12-75
- :cur-width initialization option of gwin:raster-object, 12-75
- :current-alu initialization option of gwin:world, 12-41
- :current-alu method of gwin:world, 12-41
- :current-edge-color initialization option of gwin:world, 12-41
- :current-edge-color method of gwin:world, 12-41
- :current-fill-color initialization option of gwin:world, 12-41
- :current-fill-color method of gwin:world, 12-41
- :current-font initialization option of gwin:world, 12-41
- :current-font method of gwin:world, 12-41
- :current-font method of windows, 9-4
- :current-geometry method of tv:menu, A-13
- :current-geometry method of w:menu, 14-28
- :current-item method of tv:menu, A-16
- :current-item method of w:current-item-mixin, 16-12
- :current-item method of w:menu, 14-32
- :current-margin-width initialization option of gwin:world, 12-41
- :current-margin-width method of gwin:world, 12-41
- :current-pick-tolerance initialization option of gwin:world, 12-41
- :current-pick-tolerance method of gwin:world, 12-41
- :current-tab-width initialization option of gwin:world, 12-41
- :current-tab-width method of gwin:world, 12-41
- :current-thickness initialization option of gwin:world, 12-41
- :current-thickness method of gwin:world, 12-41
- :cursor-list initialization option of gwin:graphics-window-mixin, 12-45
- :cursor-list method of gwin:graphics-window-mixin, 12-45
- :curve-x-points method of gwin:spline, 12-65
- :curve-y-points method of gwin:spline, 12-65

D

- :deactivate method of windows, 5-5
- :decide-if-scrolling-necessary method of tv:scroll-stuff-on-off-mixin, A-3
- :decide-if-scrolling-necessary method of w:scroll-bar-mixin, 11-28
- :decode-variable-type method of w:basic-choose-variable-values, 14-52
- :deexpose method of windows and screens, 5-16
- :deexposed-typeout-action initialization option of w:cache-window, 12-46
- :deexposed-typeout-action initialization option of w:sprite-window, 12-46
- :deexposed-typeout-action initialization option of windows, 7-10
- :deexposed-typeout-action method of windows, 7-10
- :default-font initialization option of tv:menu, A-14
- :default-font initialization option of w:menu, 14-30
- :default-window method of w:transform-mixin, 12-47
- :defer-reappearance method of w:blinker, 10-4
- :delayed-set-label method of w:delayed-redisplay-label-mixin, 3-9
- :delete method of gwin:basic-cursor-mixin, 12-50
- :delete-all-servers method of w:who-line-file-sheet, 18-18
- :delete-all-streams method of w:who-line-file-sheet, 18-17
- :delete-char method of windows, 7-20
- :delete-cursor method of gwin:graphics-window-mixin, 12-45
- :delete-entity method of gwin:world, 12-42
- :delete-item method of w:basic-scroll-window, 17-7
- :delete-item method of w:text-scroll-window, 16-3
- :delete-line method of windows, 7-20
- :delete-server method of w:who-line-file-sheet, 18-17
- :delete-stream method of w:who-line-file-sheet, 18-17
- :delete-string method of windows, 7-20
- :deselect method of windows, 6-3
- :deselected-visibility initialization option of w:blinker, 10-3

:deselected-visibility method of w:blinker, 10-3
 :display-item initialization option of w:basic-scroll-window, 17-6
 :display-item method of w:basic-scroll-window, 17-6
 :display-list initialization option of gwin:world, 12-42
 :display-list method of gwin:world, 12-42
 :display-lozenge-string method of windows, 7-8
 :distance method of graphic object, 12-54
 :distance method of gwin:arc, 12-59
 :distance method of gwin:backgroundpic, 12-78
 :distance method of gwin:circle, 12-60
 :distance method of gwin:line, 12-61
 :distance method of gwin:polyline, 12-62
 :distance method of gwin:raster-object, 12-76
 :distance method of gwin:rectangle, 12-63
 :distance method of gwin:ruler, 12-74
 :distance method of gwin:spline, 12-65
 :distance method of gwin:subpicture, 12-77
 :distance method of gwin:text, 12-69
 :distance method of gwin:triangle, 12-66
 :draw method of graphic object, 12-54
 :draw method of gwin:arc, 12-59
 :draw method of gwin:circle, 12-60
 :draw method of gwin:line, 12-61
 :draw method of gwin:polyline, 12-62
 :draw method of gwin:raster-character, 12-71
 :draw method of gwin:raster-object, 12-76
 :draw method of gwin:rectangle, 12-63
 :draw method of gwin:ruler, 12-74
 :draw method of gwin:spline, 12-65
 :draw method of gwin:subpicture, 12-77
 :draw method of gwin:text, 12-69
 :draw method of gwin:triangle, 12-66
 :draw method of gwin:vector-character, 12-70
 :draw-arc method of gwin:draw-mixin, A-9
 :draw-arc method of w:graphics-mixin, 12-16
 :draw-char method of tv:stream-mixin, A-9
 :draw-char method of w:stream-mixin, 7-7
 :draw-character method of gwin:font, 12-67
 :draw-circle method of gwin:draw-mixin, A-9
 :draw-circle method of tv:graphics-mixin, A-7
 :draw-circle method of w:graphics-mixin, 12-16
 :draw-circular-arc method of tv:graphics-mixin, A-7
 :draw-crosshair method of gwin:mouse-handler-mixin, 12-48
 :draw-cubic-spline method of tv:graphics-mixin, A-7
 :draw-cubic-spline method of w:graphics-mixin, 12-22
 :draw-curve method of tv:graphics-mixin, A-6
 :draw-dashed-line method of tv:graphics-mixin, A-5
 :draw-dashed-line method of w:graphics-mixin, 12-14
 :draw-filled-arc method of gwin:draw-mixin, A-9
 :draw-filled-arc method of w:graphics-mixin, 12-16
 :draw-filled-circle method of gwin:draw-mixin, A-9
 :draw-filled-circle method of w:graphics-mixin, 12-16
 :draw-filled-in-circle method of tv:graphics-mixin, A-7
 :draw-filled-in-sector method of tv:graphics-mixin, A-7
 :draw-filled-polygon method of w:graphics-mixin, 12-20
 :draw-filled-rectangle method of gwin:draw-mixin, A-9
 :draw-filled-rectangle method of w:graphics-mixin, 12-18
 :draw-filled-triangle method of gwin:draw-mixin, A-9

- :draw-filled-triangle method of w:graphics-mixin, 12-17
- :draw-filled-triangle-list method of gwin:draw-mixin, A-10
- :draw-grid method of gwin:mouse-handler-mixin, 12-49
- :draw-line method of gwin:draw-mixin, A-10
- :draw-line method of tv:graphics-mixin, A-5
- :draw-line method of w:graphics-mixin, 12-13
- :draw-lines method of tv:graphics-mixin, A-5
- :draw-picture-list method of gwin:draw-mixin, A-10
- :draw-picture-list method of w:graphics-mixin, 12-10
- :draw-point method of tv:graphics-mixin, A-5
- :draw-point method of w:graphics-mixin, 12-13
- :draw-polyline method of gwin:draw-mixin, A-10
- :draw-polyline method of w:graphics-mixin, 12-15
- :draw-raster method of gwin:draw-mixin, A-10
- :draw-raster method of w:graphics-mixin, 12-21
- :draw-rect method of gwin:draw-mixin, A-10
- :draw-rectangle method of tv:stream-mixin, A-9
- :draw-rectangle method of w:graphics-mixin, 12-18
- :draw-regular-polygon method of tv:graphics-mixin, A-7
- :draw-regular-polygon method of w:graphics-mixin, 12-19
- :draw-solid-polygon method of gwin:draw-mixin, A-10
- :draw-string method of gwin:draw-mixin, A-10
- :draw-string method of gwin:font, 12-67
- :draw-string method of w:graphics-mixin, 12-21
- :draw-triangle method of gwin:draw-mixin, A-10
- :draw-triangle method of tv:graphics-mixin, A-6
- :draw-triangle method of w:graphics-mixin, 12-17
- :draw-wide-curve method of tv:graphics-mixin, A-6
- :dynamic initialization option of w:menu, 14-22

E

- :edge-color initialization option of gwin:basic-graphics-mixin, 12-56
- :edge-color initialization option of gwin:subpicture, 12-77
- :edge-color method of gwin:basic-graphics-mixin, 12-56
- :edge-point method of graphic object, 12-54
- :edge-point method of gwin:arc, 12-59
- :edge-point method of gwin:circle, 12-60
- :edge-point method of gwin:line, 12-61
- :edge-point method of gwin:polyline, 12-62
- :edge-point method of gwin:raster-object, 12-76
- :edge-point method of gwin:rectangle, 12-63
- :edge-point method of gwin:ruler, 12-74
- :edge-point method of gwin:spline, 12-65
- :edge-point method of gwin:subpicture, 12-77
- :edge-point method of gwin:text, 12-69
- :edge-point method of gwin:triangle, 12-66
- :edges initialization option of windows, 4-2
- :edges method of windows, 4-5
- :edges-from initialization option of windows, 4-3
- :edit method of standalone editor windows, 18-11
- :edit-parameters method of graphic object, 12-55
- :edit-parameters method of gwin:arc, 12-59
- :edit-parameters method of gwin:backgroundpic, 12-78
- :edit-parameters method of gwin:circle, 12-60
- :edit-parameters method of gwin:line, 12-61
- :edit-parameters method of gwin:polyline, 12-62
- :edit-parameters method of gwin:raster-object, 12-76
- :edit-parameters method of gwin:rectangle, 12-63

- :edit-parameters method of gwin:ruler, 12-74
- :edit-parameters method of gwin:spline, 12-65
- :edit-parameters method of gwin:subpicture, 12-77
- :edit-parameters method of gwin:text, 12-69
- :edit-parameters method of gwin:triangle, 12-66
- :enable-scrolling-p method of tv:basic-scroll-bar, A-1
- :enable-scrolling-p method of w:scroll-bar-mixin, 11-30
- :end-of-line-exception method of windows, 7-14
- :end-of-page-exception method of windows, 7-12
- :entities initialization option of gwin:subpicture, 12-77
- :entities method of gwin:subpicture, 12-77
- :execute method of tv:menu, A-16
- :execute method of tv:menu-execute-mixin, A-12
- :execute method of w:menu, 14-32
- :execute-no-side-effects method of tv:menu-execute-mixin , A-12
- :execute-no-side-effects method of w:menu, 14-32
- :execute-window-op method of w:menu, 14-32
- :exposable-p method of windows and screens, 5-15
- :expose method of windows and screens, 5-13
- :expose-for-typeout method of w:basic-typeout-window, 13-3
- :expose-near method of windows, 5-14
- :expose-p initialization option of windows and screens, 5-15
- :expose-p method of windows and screens, 5-15
- :exposed-inferiors method of windows and screens, 5-15
- :extents method of gwin:basic-graphics-mixin, 12-56

F

- :fasd-form method of graphic object, 12-55
- :fasd-form method of gwin:arc, 12-59
- :fasd-form method of gwin:backgroundpic, 12-78
- :fasd-form method of gwin:circle, 12-60
- :fasd-form method of gwin:font, 12-67
- :fasd-form method of gwin:line, 12-61
- :fasd-form method of gwin:polyline, 12-62
- :fasd-form method of gwin:raster-character, 12-71
- :fasd-form method of gwin:raster-object, 12-76
- :fasd-form method of gwin:rectangle, 12-63
- :fasd-form method of gwin:ruler, 12-74
- :fasd-form method of gwin:spline, 12-65
- :fasd-form method of gwin:subpicture, 12-77
- :fasd-form method of gwin:text, 12-69
- :fasd-form method of gwin:triangle, 12-66
- :fasd-form method of gwin:vector-character, 12-71
- :fat-string-out method of windows, 7-5
- :fill-color initialization option of gwin:basic-graphics-mixin, 12-56
- :fill-color method of gwin:basic-graphics-mixin, 12-56
- :fill-p initialization option of tv:menu, A-14
- :fill-p initialization option of w:menu, 14-29
- :fill-p method of tv:menu, A-14
- :fill-p method of w:menu, 14-29
- :flashy-scrolling-region initialization option of tv:flashy-scrolling-mixin, A-2
- :flush-typeout method of w:text-scroll-window-typeout-mixin, 16-6
- :follow-p initialization option of w:blinker, 10-6
- :follow-p method of w:blinker, 10-6
- :font initialization option of gwin:cursor, 12-51
- :font initialization option of gwin:ruler, 12-72
- :font initialization option of w:character-blinker, 10-9
- :font-map initialization option of windows, 9-3

- :font-map method of windows, 9-3
- :font-name initialization option of gwin:text, 12-68
- :font-name method of gwin:text, 12-68
- :force-kbd-input method of w:stream-mixin, 8-10
- :foreground-color initialization option of windows, 19-7
- :foreground-color method of windows, 19-7
- :forward-char method of windows, 7-16
- :fresh-line method of windows, 7-8
- :frozen? initialization option of gwin:sprite-cursor, 12-53
- :frozen? method of gwin:sprite-cursor, 12-53
- :function initialization option of w:basic-choose-variable-values, 14-54
- :function method of w:basic-choose-variable-values, 14-54

G

- :geometry initialization option of tv:menu, A-13
- :geometry initialization option of w:menu, 14-28
- :geometry method of tv:menu, A-13
- :geometry method of w:menu, 14-28
- :get-configuration method of w:basic-constraint-frame, 15-35
- :get-item method of w:basic-scroll-window, 17-7
- :get-mouse-position method of gwin:mouse-handler-mixin, 12-49
- :get-pane method of w:basic-constraint-frame, 15-34
- :gray-array initialization option of w:gray-deexposed-right-mixin, 5-20
- :gray-array initialization option of w:gray-deexposed-wrong-mixin, 5-20
- :gray-array method of w:gray-deexposed-right-mixin, 5-20
- :gray-array method of w:gray-deexposed-wrong-mixin, 5-20
- :grid-on initialization option of gwin:mouse-handler-mixin, 12-48
- :grid-on method of gwin:mouse-handler-mixin, 12-48
- :grid-x initialization option of gwin:mouse-handler-mixin, 12-48
- :grid-x method of gwin:mouse-handler-mixin, 12-48
- :grid-y initialization option of gwin:mouse-handler-mixin, 12-48
- :grid-y method of gwin:mouse-handler-mixin, 12-48
- :gridify-point method of gwin:mouse-handler-mixin, 12-49

H

- :half-period initialization option of w:blinker, 10-3
- :half-period method of w:blinker, 10-3
- :handle-asynchronous-character method of w:stream-mixin, 8-21
- :handle-exceptions method of windows, 7-12
- :handle-mouse method of windows, 11-12
- :handle-mouse-scroll method of w:scroll-bar-mixin, 11-27
- :height initialization option of gwin:bitblt-blinker, 12-51
- :height initialization option of gwin:block-cursor, 12-52
- :height initialization option of gwin:raster-object, 12-75
- :height initialization option of gwin:rectangle, 12-63
- :height initialization option of gwin:sprite-cursor, 12-53
- :height initialization option of w:bitblt-blinker, 10-10
- :height initialization option of w:cache-window, 12-46
- :height initialization option of w:ibeam-blinker, 10-8
- :height initialization option of w:rectangular-blinker, 10-7
- :height initialization option of w:sprite-window, 12-46
- :height initialization option of windows, 4-2
- :height method of gwin:bitblt-blinker, 12-51
- :height method of gwin:block-cursor, 12-52
- :height method of gwin:rectangle, 12-63
- :height method of windows, 4-4
- :highlight method of gwin:backgroundpic, 12-78
- :highlight method of gwin:basic-graphics-mixin, 12-56

:highlight-2 method of gwin:basic-graphics-mixin, 12-56
 :highlighted-items initialization option of tv:menu-highlighting-mixin, A-22
 :highlighted-items initialization option of w:menu, 14-20
 :highlighted-items method of tv:menu-highlighting-mixin, A-22
 :highlighted-items method of w:menu, 14-20
 :highlighted-values method of tv:menu-highlighting-mixin, A-22
 :highlighted-values method of w:menu, 14-21
 :highlighting initialization option of w:menu, 14-20
 :highlighting method of w:menu, 14-20
 :home-cursor method of windows, 7-16
 :home-down method of windows, 7-16
 :horizontal-spacing initialization option of gwin:basic-character-mixin, 12-70
 :horizontal-spacing method of gwin:basic-character-mixin, 12-70
 :horz-spacing initialization option of gwin:font, 12-67
 :horz-spacing method of gwin:font, 12-67
 :hysteresis initialization option of w:hysteretic-window-mixin, 11-8
 :hysteresis method of w:hysteretic-window-mixin, 11-8

I

:identity? initialization option of w:transform-mixin, 12-47
 :identity? method of w:transform-mixin, 12-47
 :identity-cache initialization option of gwin:raster-character, 12-71
 :identity-cache method of gwin:raster-character, 12-71
 :identity-height initialization option of gwin:raster-character, 12-71
 :identity-height method of gwin:raster-character, 12-71
 :identity-width initialization option of gwin:raster-character, 12-71
 :identity-width method of gwin:raster-character, 12-71
 :incomplete-p method of w:basic-typeout-window, 13-5
 :increment-cursorpos method of windows, 7-15
 :inferior-set-edges method of windows, 4-6
 :inferiors method of windows and screens, 5-4
 :insert-arc method of gwin:world, 12-43
 :insert-backgroundpic method of gwin:world, 12-43
 :insert-char method of windows, 7-19
 :insert-circle method of gwin:world, 12-43
 :insert-item method of w:basic-scroll-window, 17-7
 :insert-item method of w:text-scroll-window, 16-3
 :insert-line method of gwin:world, 12-43
 :insert-line method of windows, 7-20
 :insert-polyline method of gwin:world, 12-43
 :insert-raster method of gwin:world, 12-43
 :insert-rectangle method of gwin:world, 12-43
 :insert-ruler method of gwin:world, 12-43
 :insert-spline method of gwin:world, 12-43
 :insert-string method of windows, 7-19
 :insert-subpicture method of gwin:world, 12-43
 :insert-text method of gwin:world, 12-44
 :insert-triangle method of gwin:world, 12-44
 :inside-edges method of windows, 4-5
 :inside-height initialization option of windows, 4-3
 :inside-height method of windows, 4-5
 :inside-p method of gwin:basic-graphics-mixin, 12-56
 :inside-size initialization option of windows, 4-3
 :inside-size method of windows, 4-5
 :inside-width initialization option of windows, 4-3
 :inside-width method of windows, 4-5
 :integral-p initialization option of windows, 4-3
 :interval method of editor windows, 18-12

- :interval-string method of editor windows, 18-11
- :io-buffer initialization option of nil, 14-57
- :io-buffer initialization option of tv:command-menu, A-18
- :io-buffer initialization option of w:bordered-constraint-frame-with-shared-iobuffer, 15-15
- :io-buffer initialization option of w:constraint-frame-with-shared-iobuffer, 15-15
- :io-buffer initialization option of w:stream-mixin, 8-3
- :io-buffer method of tv:command-menu, A-18
- :io-buffer method of w:stream-mixin, 8-3
- :item method of w:basic-mouse-sensitive-items, 14-63
- :item method of w:mouse-sensitive-text-scroll-window, 16-9
- :item-alignment initialization option of w:menu, 14-30
- :item-cursorpos method of tv:menu, A-17
- :item-cursorpos method of w:menu, 14-32
- :item-generator method of w:text-scroll-window, 16-6
- :item-list initialization option of tv:menu, A-16
- :item-list initialization option of w:basic-multiple-choice, 14-37
- :item-list initialization option of w:menu, 14-24
- :item-list method of tv:menu, A-16
- :item-list method of w:basic-mouse-sensitive-items, 14-64
- :item-list method of w:basic-multiple-choice, 14-37
- :item-list method of w:menu, 14-24
- :item-list-pointer initialization option of tv:dynamic-item-list-mixin, A-19
- :item-list-pointer initialization option of w:menu, 14-23
- :item-list-pointer method of tv:dynamic-item-list-mixin, A-19
- :item-list-pointer method of w:menu, 14-23
- :item-name initialization option of w:basic-multiple-choice, 14-37
- :item-name method of w:basic-multiple-choice, 14-37
- :item-of-number method of w:text-scroll-window, 16-3
- :item-rectangle method of tv:menu, A-17
- :item-rectangle method of w:menu, 14-32
- :item-type-alist initialization option of w:basic-mouse-sensitive-items, 14-62
- :item-type-alist method of w:basic-mouse-sensitive-items, 14-62
- :item1 method of w:mouse-sensitive-text-scroll-window, 16-9
- :items method of w:text-scroll-window, 16-2

K

- :keypad-enable initialization option of windows, 8-26
- :kill method of windows, 5-5

L

- :label initialization option of w:label-mixin, 3-5
- :label method of w:label-mixin, 3-5
- :label-background initialization option of w:label-mixin, 19-8
- :label-background method of w:label-mixin, 19-8
- :label-box-p initialization option of w:box-label-mixin, 3-8
- :label-color initialization option of w:label-mixin, 19-8
- :label-color method of w:label-mixin, 19-8
- :label-size method of w:label-mixin, 3-5
- :labels initialization option of gwin:ruler, 12-72
- :labels method of gwin:ruler, 12-72
- :last-item method of tv:menu, A-17
- :last-item method of w:menu, 14-25
- :last-item method of w:text-scroll-window, 16-3
- :left initialization option of gwin:rectangle, 12-63
- :left initialization option of windows, 4-2
- :left method of gwin:rectangle, 12-63
- :left-flag initialization option of gwin:sprite-cursor, 12-53
- :left-flag method of gwin:sprite-cursor, 12-53

- :left-kern initialization option of gwin:raster-character, 12-72
- :left-kern method of gwin:raster-character, 12-72
- :left-limit initialization option of gwin:sprite-cursor, 12-53
- :left-limit initialization option of gwin:world, 12-44
- :left-limit method of gwin:sprite-cursor, 12-53
- :left-limit method of gwin:world, 12-44
- :left-margin-size method of windows, 3-2
- :line-area-mouse-documentation method of w:line-area-text-scroll-mixin, 16-11
- :line-area-width initialization option of w:line-area-text-scroll-mixin, 16-11
- :line-out method of windows, 7-5
- :list-tyl method of w:stream-mixin, A-23
- :listen method of w:stream-mixin, 8-10

M

- :magnification initialization option of w:magnifying-blinker, 10-11
- :magnification method of w:magnifying-blinker, 10-11
- :make-complete method of w:basic-typeout-window, 13-5
- :make-incomplete method of w:basic-typeout-window, 13-5
- :margin initialization option of gwin:subpicture, 12-77
- :margin initialization option of gwin:text, 12-68
- :margin method of gwin:subpicture, 12-77
- :margin method of gwin:text, 12-68
- :margin-choice-default-font initialization option of w:margin-choice-mixin, 14-66
- :margin-choices initialization option of w:choose-variable-values-window, 14-55
- :margin-choices initialization option of w:margin-choice-mixin, 14-66
- :margin-scroll-regions initialization option of tv:margin-scroll-mixin, A-3
- :margins method of windows, 3-2
- :markers method of gwin:basic-graphics-mixin, 12-56
- :menu-draw method of tv:menu, A-17
- :menu-draw method of w:menu, 14-32
- :menu-margin-choices initialization option of tv:menu-margin-choice-mixin, A-4
- :menu-margin-choices initialization option of w:menu, 14-21
- :min-dot-delta initialization option of gwin:draw-mixin, A-10
- :min-dot-delta initialization option of w:cache-window, 12-47
- :min-dot-delta initialization option of w:graphics-mixin, 12-9
- :min-dot-delta initialization option of w:sprite-window, 12-47
- :min-dot-delta method of gwin:draw-mixin, A-10
- :min-dot-delta method of w:graphics-mixin, 12-9
- :min-nil-delta initialization option of gwin:draw-mixin, A-10
- :min-nil-delta initialization option of w:cache-window, 12-47
- :min-nil-delta initialization option of w:graphics-mixin, 12-9
- :min-nil-delta initialization option of w:sprite-window, 12-47
- :min-nil-delta method of gwin:draw-mixin, A-10
- :min-nil-delta method of w:graphics-mixin, 12-9
- :minimum-height initialization option of windows, 4-3
- :minimum-width initialization option of windows, 4-3
- :monochrome-blanking method of w:control-register, 19-24
- :monochrome-polarity method of w:control-register, 19-24
- :more-exception method of windows, 7-13
- :more-p initialization option of windows, 7-25
- :more-p method of w:basic-typeout-window, 13-4
- :more-p method of windows, 7-25
- :more-vpos method of windows, 7-13
- :mouse-buttons method of windows, 11-14
- :mouse-buttons-on-item method of tv:menu, A-17
- :mouse-buttons-on-item method of w:menu, 14-32
- :mouse-buttons-scroll method of tv:basic-scroll-bar, A-2
- :mouse-click method of gwin:mouse-handler-mixin, 12-49

:mouse-click method of w:margin-region-mixin, processing mouse clicks other than R2, 3-10
 :mouse-click method of w:mouse-sensitive-text-scroll-window, 16-9
 :mouse-click method of windows, 11-14
 :mouse-moves method of windows, 11-13
 :mouse-or-kbd-tyi method of w:stream-mixin, A-23
 :mouse-or-kbd-tyi-no-hang method of w:stream-mixin, A-23
 :mouse-select method of windows, 6-3
 :mouse-sensitive-item method of w:basic-mouse-sensitive-items, 14-64
 :mouse-sensitive-item method of w:mouse-sensitive-text-scroll-window-without-click, 16-10
 :mouse-standard-blinker method of gwin:mouse-handler-mixin, 12-49
 :mouse-standard-blinker method of w:menu, 14-31
 :mouse-standard-blinker method of windows, 11-17
 :move method of graphic object, 12-55
 :move method of gwin:arc, 12-59
 :move method of gwin:circle, 12-60
 :move method of gwin:line, 12-61
 :move method of gwin:polyline, 12-62
 :move method of gwin:raster-object, 12-76
 :move method of gwin:rectangle, 12-63
 :move method of gwin:ruler, 12-74
 :move method of gwin:spline, 12-65
 :move method of gwin:sprite-cursor, 12-54
 :move method of gwin:subpicture, 12-77
 :move method of gwin:text, 12-69
 :move method of gwin:triangle, 12-66
 :move-near-window method of tv:menu, A-16
 :move-near-window method of w:menu, 14-32
 :multicolumn initialization option of w:menu, 14-23

N

:name initialization option of gwin:subpicture, 12-77
 :name initialization option of w:minimum-window, 3-5
 :name method of gwin:subpicture, 12-77
 :name method of w:minimum-window, 3-5
 :name-font initialization option of w:basic-choose-variable-values, 14-54
 :name-for-selection method of windows, 6-6
 :nearest-x method of gwin:basic-graphics-mixin, 12-56
 :nearest-y method of gwin:basic-graphics-mixin, 12-56
 :new-scroll-position method of scrolling windows, 16-4
 :new-window method of w:transform-mixin, 12-47
 :non-sensitive-mouse-click user-defined method of w:basic-mouse-sensitive-items, 14-64
 :notice method of windows, 18-4
 :number-of-item method of w:text-scroll-window, 16-3
 :number-of-items method of w:text-scroll-window, 16-2

O

:objects-in-window initialization option of gwin:world, 12-42
 :objects-in-window method of gwin:world, 12-42
 :offset method of gwin:basic-cursor-mixin, 12-50
 :offsets method of w:mouse-blinker-mixin, 11-17
 :open-streams method of w:who-line-file-sheet, 18-17
 :order-inferiors method of windows, 5-21
 :origin method of gwin:basic-graphics-mixin, 12-56
 :output-hold-exception method of windows, 7-12
 :outside-p method of gwin:basic-graphics-mixin, 12-56
 :overlap-p method of gwin:basic-graphics-mixin, 12-57

P

- :package method of w:listener-mixin-internal, 18-10
- :pan method of w:transform-mixin, 12-47
- :pane-name method of w:basic-constraint-frame, 15-34
- :pane-size method of windows, 15-34
- :pane-types-alist method of frames, 15-36
- :panes initialization option of all constraint frame flavors, 15-22
- :parse-font-name method of w:screen, 9-5
- :parse-font-specifier method of w:screen, 9-5
- :permanent initialization option of w:menu, 14-22
- :phase method of w:blinker, 10-6
- :pick method of gwin:world, 12-43
- :plane-mask method of windows, 19-23
- :playback method of w:stream-mixin, 8-11
- :point method of tv:graphics-mixin, A-5
- :point method of w:graphics-mixin, 12-13
- :pop-up initialization option of w:menu, 14-22
- :position initialization option of windows, 4-2
- :position method of gwin:basic-cursor-mixin, 12-50
- :position method of windows, 4-5
- :preemptable-read method of w:preemptable-read-any-tyi-mixin, 8-12
- :primitive-item method of w:basic-mouse-sensitive-items, 14-63
- :primitive-item-outside method of w:basic-mouse-sensitive-items, 14-63
- :print-function initialization option of w:function-text-scroll-window, 16-5
- :print-function method of w:function-text-scroll-window, 16-5
- :print-function-arg initialization option of w:function-text-scroll-window, 16-5
- :print-function-arg method of w:function-text-scroll-window, 16-5
- :print-item method of w:text-scroll-window, 16-2
- :print-notification method of windows, 18-2
- :print-notification-on-self method of w:notification-mixin, 18-2
- :priority initialization option of windows, 5-22
- :process initialization option of w:process-mixin, 6-12
- :process method of w:process-mixin, 6-12
- :process method of w:select-mixin, 6-12
- :process-character method of w:basic-choose-variable-values, 14-58
- :process-character method of w:menu, 14-25
- :processes method of windows, 6-13
- :prompt-text initialization option of gwin:graphics-window-mixin, 12-45
- :prompt-text method of gwin:graphics-window-mixin, 12-45
- :put-item-in-window method of w:text-scroll-window, 16-3
- :put-last-item-in-window method of w:text-scroll-window, 16-3

R

- :radius initialization option of gwin:circle, 12-59
- :radius method of gwin:circle, 12-59
- :read-any method of w:stream-mixin, 8-9
- :read-any-no-hang method of w:stream-mixin, 8-9
- :read-cursorpos method of w:blinker, 10-6
- :read-cursorpos method of windows, 7-15
- :read-display-list method of gwin:world, 12-42
- :read-list method of w:stream-mixin, 8-9
- :read-mouse-or-kbd method of w:stream-mixin, 8-9
- :read-mouse-or-kbd-no-hang method of w:stream-mixin, 8-9
- :redefine-configuration method of w:basic-constraint-frame, 15-35
- :redefine-margins method of windows, 3-14
- :redisplay method of w:basic-scroll-window, 17-6
- :redisplay method of w:text-scroll-window, 16-3
- :redisplay-selected-items method of w:basic-scroll-window, 17-7

- :redraw method of gwin:basic-cursor-mixin, 12-51
- :redraw method of gwin:sprite-cursor, 12-54
- :redraw-crosshair method of gwin:mouse-handler-mixin, 12-49
- :refresh method of windows, 5-10
- :refresh-area method of gwin:graphics-window-mixin, 12-46
- :refresh-margins method of windows, 3-13
- :refresh-rubout-handler method of w:stream-mixin, 8-12
- :region-list initialization option of w:margin-region-mixin, 3-9
- :remove-asynchronous-character method of w:stream-mixin, 8-21
- :remove-highlighted-item method of tv:menu-highlighting-mixin, A-22
- :remove-highlighted-item method of w:menu, 14-20
- :remove-highlighted-value method of tv:menu-highlighting-mixin, A-22
- :remove-highlighted-value method of w:menu, 14-21
- :remove-selection-substitute method of windows, 6-9
- :replace-entity method of gwin:world, 12-42
- :restore-rubout-handler-buffer method of w:stream-mixin, 8-12
- :resume-crosshair method of gwin:mouse-handler-mixin, 12-49
- :reverse-video-p initialization option of windows, 5-9
- :reverse-video-p method of windows, 5-9
- :right initialization option of windows, 4-2
- :right-flag initialization option of gwin:sprite-cursor, 12-53
- :right-flag method of gwin:sprite-cursor, 12-53
- :right-limit initialization option of gwin:sprite-cursor, 12-53
- :right-limit initialization option of gwin:world, 12-44
- :right-limit method of gwin:sprite-cursor, 12-53
- :right-limit method of gwin:world, 12-44
- :right-margin-character-flag initialization option of windows, 7-25
- :right-margin-size method of windows, 3-2
- :right-shadow-width initialization option of w:shadow-borders-mixin, 5-17
- :rows initialization option of tv:menu, A-14
- :rows initialization option of w:menu, 14-29
- :rubout-handler method of w:stream-mixin, 8-11

S

- :save-bits initialization option of windows, 5-10
- :save-bits method of windows, 5-10
- :save-rubout-handler-buffer method of w:stream-mixin, 8-12
- :scale method of graphic object, 12-55
- :scale method of gwin:arc, 12-59
- :scale method of gwin:circle, 12-60
- :scale method of gwin:line, 12-61
- :scale method of gwin:polyline, 12-62
- :scale method of gwin:raster-object, 12-76
- :scale method of gwin:rectangle, 12-63
- :scale method of gwin:ruler, 12-74
- :scale method of gwin:spline, 12-65
- :scale method of gwin:subpicture, 12-77
- :scale method of gwin:text, 12-69
- :scale method of gwin:triangle, 12-66
- :screen-array method of windows and screens, 5-15
- :screen-manage method of windows and screens, 5-19
- :screen-manage-autoexpose-inferiors method of windows and screens, 5-19
- :screen-manage-deexposed-visibility method of windows, 5-20
- :scroll-bar initialization option of tv:basic-scroll-bar, A-1
- :scroll-bar method of tv:basic-scroll-bar, A-1
- :scroll-bar method of w:scroll-bar-mixin, 11-27
- :scroll-bar-always-displayed initialization option of tv:basic-scroll-bar, A-1
- :scroll-bar-always-displayed method of tv:basic-scroll-bar, A-1

:scroll-bar-delay-time method of w:scroll-bar-mixin, 11-30
 :scroll-bar-draw-edge-p method of w:scroll-bar-mixin, 11-28
 :scroll-bar-icon-height method of w:scroll-bar-mixin, 11-27
 :scroll-bar-icon-width method of w:scroll-bar-mixin, 11-27
 :scroil-bar-lines method of w:scroll-bar-mixin, 11-30
 :scroll-bar-mode method of w:scroll-bar-mixin, 11-28
 :scroll-bar-on-off method of w:scroll-bar-mixin, 11-28
 :scroll-bar-on-right method of w:scroll-bar-mixin, 11-27
 :scroll-bar-side initialization option of w:scroll-bar-mixin, 11-26
 :scroll-more-above method of w:scroll-bar-mixin, 11-29
 :scroll-more-below method of w:scroll-bar-mixin, 11-29
 :scroll-position method of scrolling windows, 16-4
 :scroll-redisplay method of w:text-scroll-window, 16-3
 :scroll-relative method of w:scroll-bar-mixin, 11-29
 :scroll-to method of scrolling windows, 16-4
 :scrolling-p initialization option of w:menu, 14-23
 :select method of windows, 6-2
 :selectable-windows method of windows, 6-5
 :selected-choice-font initialization option of w:basic-choose-variable-values, 14-55
 :selected-pane initialization option of w:basic-constraint-frame, 15-36
 :selection-substitute method of windows, 6-9
 :self-or-substitute-selected-p method of windows, 6-9
 :send-all-exposed-panes method of w:basic-constraint-frame, 15-34
 :send-all-panes method of w:basic-constraint-frame, 15-34
 :send-pane method of w:basic-constraint-frame, 15-34
 :sensitive-item-types initialization option of w:mouse-sensitive-text-scroll-window, 16-10
 :sensitive-item-types method of w:mouse-sensitive-text-scroll-window, 16-10
 :servers method of w:who-line-file-sheet, 18-17
 :set-allow-interrupts? method of gwin:draw-mixin, A-9
 :set-allow-interrupts? method of w:graphics-mixin, 12-10
 :set-alu method of gwin:basic-graphics-mixin, 12-56
 :set-angle method of gwin:arc, 12-58
 :set-array method of w:bitblt-blinker, 10-10
 :set-background-color method of windows, 19-7
 :set-blinker-height method of gwin:font, 12-67
 :set-blinker-offset method of blinkers, 19-8
 :set-blinker-width method of gwin:font, 12-67
 :set-border-color method of gwin:text, 12-68
 :set-border-color method of w:borders-mixin, 19-8
 :set-border-margin-width method of w:borders-mixin, 3-3
 :set-borders method of w:borders-mixin, 3-3
 :set-bottom-flag method of gwin:sprite-cursor, 12-53
 :set-bottom-limit method of gwin:sprite-cursor, 12-53
 :set-bottom-shadow-width method of w:shadow-borders-mixin, 5-17
 :set-character method of gwin:cursor, 12-51
 :set-character method of gwin:font, 12-67
 :set-character method of w:character-blinker, 10-9
 :set-characters method of gwin:font, 12-67
 :set-choice-types method of w:basic-multiple-choice, 14-37
 :set-chosen-item method of w:menu, 14-25
 :set-color-blanking method of w:control-register, 19-24
 :set-color-map method of w:sheet, 19-10
 :set-column-spec-list method of tv:dynamic-multicolumn-mixin, A-20
 :set-column-spec-list method of w:menu, 14-24
 :set-command-characters method of w:menu, 14-26
 :set-configuration method of w:basic-constraint-frame, 15-34
 :set-crosshair-mode method of gwin:mouse-handler-mixin, 12-48
 :set-current-alu method of gwin:world, 12-41

:set-current-edge-color method of gwin:world, 12-41
 :set-current-fill-color method of gwin:world, 12-41
 :set-current-font method of gwin:world, 12-41
 :set-current-font method of windows, 9-4
 :set-current-item method of w:current-item-mixin, 16-12
 :set-current-margin-width method of gwin:world, 12-41
 :set-current-pick-tolerance method of gwin:world, 12-41
 :set-current-tab-width method of gwin:world, 12-41
 :set-current-thickness method of gwin:world, 12-41
 :set-cursor-list method of gwin:graphics-window-mixin, 12-45
 :set-cursorpos method of gwin:bitblt-blinker, 12-52
 :set-cursorpos method of w:blinker, 10-6
 :set-cursorpos method of windows, 7-16
 :set-deexposed-typeout-action method of windows, 7-10
 :set-default-font method of tv:menu, A-14
 :set-default-font method of w:menu, 14-30
 :set-deselected-visibility method of w:blinker, 10-3
 :set-display-item method of w:basic-scroll-window, 17-6
 :set-display-list method of gwin:world, 12-42
 :set-edge-color method of gwin:basic-graphics-mixin, 12-56
 :set-edges method of tv:menu, A-14
 :set-edges method of windows, 4-6
 :set-entities method of gwin:subpicture, 12-77
 :set-fill-color method of gwin:basic-graphics-mixin, 12-56
 :set-fill-p method of tv:menu, A-14
 :set-fill-p method of w:menu, 14-29
 :set-follow-p method of w:blinker, 10-6
 :set-font-map method of windows, 9-3
 :set-font-name method of gwin:text, 12-68
 :set-foreground-color method of windows, 19-7
 :set-frozen? method of gwin:sprite-cursor, 12-53
 :set-function method of w:basic-choose-variable-values, 14-54
 :set-geometry method of tv:menu, A-13
 :set-geometry method of w:menu, 14-28
 :set-gray-array method of w:gray-deexposed-right-mixin, 5-20
 :set-gray-array method of w:gray-deexposed-wrong-mixin, 5-20
 :set-grid-on method of gwin:mouse-handler-mixin, 12-48
 :set-grid-x method of gwin:mouse-handler-mixin, 12-48
 :set-grid-y method of gwin:mouse-handler-mixin, 12-48
 :set-half-period method of w:blinker, 10-3
 :set-height method of gwin:rectangle, 12-63
 :set-highlighted-items method of tv:menu-highlighting-mixin, A-22
 :set-highlighted-items method of w:menu, 14-20
 :set-highlighted-values method of tv:menu-highlighting-mixin, A-22
 :set-highlighted-values method of w:menu, 14-21
 :set-horizontal-spacing method of gwin:basic-character-mixin, 12-70
 :set-horz-spacing method of gwin:font, 12-67
 :set-hysteresis method of w:hysteretic-window-mixin, 11-8
 :set-identity? method of w:transform-mixin, 12-47
 :set-inside-size method of windows, 4-5
 :set-interval method of editor windows, 18-12
 :set-interval-string method of editor windows, 18-11
 :set-io-buffer method of tv:command-menu, A-18
 :set-io-buffer method of w:stream-mixin, 8-3
 :set-item method of w:basic-scroll-window, 17-7
 :set-item-generator method of w:text-scroll-window, 16-6
 :set-item-list method of tv:margin-multiple-menu-mixin, A-22
 :set-item-list method of tv:menu, A-16

:set-item-list method of w:basic-multiple-choice, 14-37
 :set-item-list method of w:menu, 14-24
 :set-item-list-pointer method of tv:dynamic-item-list-mixin, A-19
 :set-item-list-pointer method of w:menu, 14-23
 :set-item-name method of w:basic-multiple-choice, 14-37
 :set-item-type-alist method of w:basic-mouse-sensitive-items, 14-62
 :set-items method of w:text-scroll-window, 16-2
 :set-label method of w:label-mixin, 3-5
 :set-label-background method of w:label-mixin, 19-8
 :set-label-color method of w:label-mixin, 19-8
 :set-last-item method of tv:menu, A-17
 :set-last-item method of w:menu, 14-25
 :set-left method of gwin:rectangle, 12-63
 :set-left-flag method of gwin:sprite-cursor, 12-53
 :set-left-limit method of gwin:sprite-cursor, 12-53
 :set-magnification method of w:magnifying-blinker, 10-11
 :set-margin method of gwin:subpicture, 12-77
 :set-margin method of gwin:text, 12-68
 :set-margin-choices method of w:margin-choice-mixin, 14-66
 :set-menu-margin-choices method of tv:menu-margin-choice-mixin, A-4
 :set-menu-margin-choices method of w:menu, 14-21
 :set-min-dot-delta method of gwin:draw-mixin, A-10
 :set-min-dot-delta method of w:graphics-mixin, 12-9
 :set-min-nil-delta method of gwin:draw-mixin, A-10
 :set-min-nil-delta method of w:graphics-mixin, 12-9
 :set-monochrome-blanking method of w:control-register, 19-24
 :set-monochrome-polarity method of w:control-register, 19-24
 :set-more-p method of w:basic-typeout-window, 13-4
 :set-more-p method of windows, 7-25
 :set-mouse-cursorpos method of windows, 11-13
 :set-mouse-position method of windows, 11-13
 :set-name method of gwin:subpicture, 12-77
 :set-nearest-x method of gwin:basic-graphics-mixin, 12-56
 :set-nearest-y method of gwin:basic-graphics-mixin, 12-56
 :set-objects-in-window method of gwen:world, 12-42
 :set-offset method of gwin:basic-cursor-mixin, 12-50
 :set-offsets method of w:mouse-blinker-mixin, 11-17
 :set-origin method of gwin:basic-graphics-mixin, 12-57
 :set-package method of w:listener-mixin-internal, 18-10
 :set-plane-mask method of windows, 19-23
 :set-position method of gwin:basic-cursor-mixin, 12-50
 :set-position method of gwin:sprite-cursor, 12-54
 :set-position method of w:menu, 14-29
 :set-position method of windows, 4-5
 :set-print-function method of w:function-text-scroll-window, 16-5
 :set-print-function-arg method of w:function-text-scroll-window, 16-5
 :set-process method of w:process-mixin, 6-12
 :set-process method of w:select-mixin, 6-12
 :set-prompt-text method of gwin:graphics-window-mixin, 12-45
 :set-radius method of gwin:circle, 12-59
 :set-region-list method of w:margin-region-mixin, 3-9
 :set-reverse-video-p method of windows, 5-9
 :set-right-limit method of gwin:sprite-cursor, 12-53
 :set-right-shadow-width method of w:shadow-borders-mixin, 5-17
 :set-save-bits method of windows, 5-10
 :set-scales method of w:cache-window, 12-46
 :set-scales method of w:sprite-window, 12-46
 :set-scroll-bar method of tv:basic-scroll-bar, A-1

:set-scroll-bar-always-displayed method of tv:basic-scroll-bar, A-1
 :set-scroll-bar-delay-time method of w:scroll-bar-mixin, 11-30
 :set-scroll-bar-draw-edge-p method of w:scroll-bar-mixin, 11-28
 :set-scroll-bar-icon-height method of w:scroll-bar-mixin, 11-27
 :set-scroll-bar-icon-width method of w:scroll-bar-mixin, 11-27
 :set-scroll-bar-lines method of w:scroll-bar-mixin, 11-30
 :set-scroll-bar-mode method of w:scroll-bar-mixin, 11-28
 :set-scroll-bar-on-off method of w:scroll-bar-mixin, 11-28
 :set-sensitive-item-types method of w:mouse-sensitive-text-scroll-window, 16-10
 :set-shadow-draw-function method of w:shadow-borders-mixin, 5-17
 :set-sheet method of w:blinker, 10-5
 :set-size method of gwin:block-cursor, 12-52
 :set-size method of w:bitblt-blinker, 10-10
 :set-size method of w:blinker, 10-7
 :set-size method of w:rectangular-blinker, 10-8
 :set-size method of windows, 4-5
 :set-size-and-cursorpos method of w:blinker, 10-7
 :set-size-and-cursorpos method of w:rectangular-blinker, 10-8
 :set-size-in-characters method of windows, 7-16
 :set-sort method of w:menu, 14-31
 :set-stack-group method of w:basic-choose-variable-values, 14-54
 :set-superior method of windows and screens, 5-4
 :set-tab-width method of gwin:text, 12-68
 :set-text-string method of gwin:text, 12-68
 :set-thickness method of gwin:basic-character-mixin, 12-70
 :set-tick method of gwin:world, 12-42
 :set-time-between-moves method of gwin:sprite-cursor, 12-54
 :set-top method of gwin:rectangle, 12-63
 :set-top-flag method of gwin:sprite-cursor, 12-53
 :set-top-item method of w:text-scroll-window, 16-2
 :set-top-limit method of gwin:sprite-cursor, 12-53
 :set-tracker-cursor method of gwin:mouse-handler-mixin, 12-48
 :set-transform method of gwin:ruler, 12-74
 :set-transform method of gwin:subpicture, 12-77
 :set-transform method of w:transform-mixin, 12-47
 :set-truncation method of w:basic-scroll-window, 17-6
 :set-value-array method of w:basic-scroll-window, 17-6
 :set-variables method of w:basic-choose-variable-values, 14-54, 14-56
 :set-vert-spacing method of gwin:font, 12-67
 :set-vertical-spacing method of gwin:basic-character-mixin, 12-70
 :set-visibility method of gwin:basic-cursor-mixin, 12-50
 :set-visibility method of w:blinker, 10-3
 :set-vsp method of windows, 7-25
 :set-weight method of gwin:arc, 12-58
 :set-weight method of gwin:circle, 12-59
 :set-weight method of gwin:line, 12-60
 :set-weight method of gwin:polyline, 12-61
 :set-weight method of gwin:rectangle, 12-63
 :set-weight method of gwin:ruler, 12-73
 :set-weight method of gwin:spline, 12-64
 :set-weight method of gwin:subpicture, 12-77
 :set-weight method of gwin:text, 12-69
 :set-weight method of gwin:triangle, 12-65
 :set-width method of gwin:rectangle, 12-63
 :set-world method of gwin:graphics-window-mixin, 12-45
 :set-x-center method of gwin:arc, 12-58
 :set-x-center method of gwin:circle, 12-59
 :set-x-end method of gwin:line, 12-60

- :set-x-end method of gwin:text, 12-69
- :set-x-max method of gwin:basic-graphics-mixin, 12-57
- :set-x-min method of gwin:basic-graphics-mixin, 12-57
- :set-x-points method of gwin:polyline, 12-62
- :set-x-points method of gwin:spline, 12-64
- :set-x-start method of gwin:arc, 12-58
- :set-x-start method of gwin:line, 12-60
- :set-x-start method of gwin:text, 12-69
- :set-x-step method of gwin:sprite-cursor, 12-53
- :set-x1 method of gwin:triangle, 12-66
- :set-x2 method of gwin:triangle, 12-66
- :set-x3 method of gwin:triangle, 12-66
- :set-y-center method of gwin:arc, 12-58
- :set-y-center method of gwin:circle, 12-59
- :set-y-end method of gwin:line, 12-60
- :set-y-end method of gwin:text, 12-69
- :set-y-max method of gwin:basic-graphics-mixin, 12-57
- :set-y-min method of gwin:basic-graphics-mixin, 12-57
- :set-y-points method of gwin:polyline, 12-62
- :set-y-points method of gwin:spline, 12-64
- :set-y-start method of gwin:arc, 12-58
- :set-y-start method of gwin:line, 12-60
- :set-y-step method of gwin:sprite-cursor, 12-53
- :set-y1 method of gwin:triangle, 12-66
- :set-y2 method of gwin:triangle, 12-66
- :set-y3 method of gwin:triangle, 12-66
- :setup method of w:basic-choose-variable-values, 14-55
- :setup method of w:basic-multiple-choice, 14-37
- :setup method of w:function-text-scroll-window, 16-5
- :shadow-draw-function initialization option of w:shadow-borders-mixin, 5-17
- :sheet initialization option of w:blinker, 10-5
- :sheet method of w:blinker, 10-5
- :size initialization option of windows, 4-2
- :size method of gwin:bitblt-blinker, 12-52
- :size method of gwin:sprite-cursor, 12-54
- :size method of w:bitblt-blinker, 10-10
- :size method of w:blinker, 10-7
- :size method of windows, 4-5
- :size-in-characters method of windows, 7-16
- :sort initialization option of w:menu, 14-31
- :spacing initialization option of gwin:ruler, 12-72
- :special-choices initialization option of tv:marginal-menu-mixin, A-22
- :square-pane-inside-size method of windows, 15-34
- :square-pane-size method of windows, 15-34
- :stack-group initialization option of w:basic-choose-variable-values, 14-54
- :stack-group method of w:basic-choose-variable-values, 14-54
- :start-value initialization option of gwin:ruler, 12-72
- :status method of windows, 6-10
- :string-font initialization option of w:basic-choose-variable-values, 14-54
- :string-in method of w:stream-mixin, 8-10
- :string-length method of windows, 7-22
- :string-line-in method of w:stream-mixin, 8-10
- :string-out method of windows, 7-5
- :string-out-centered method of windows, 7-7
- :string-out-centered-explicit method of windows, 7-24
- :string-out-down method of windows, 7-6
- :string-out-explicit method of windows, 7-23
- :string-out-up method of windows, 7-6

:string-out-x-y-centered-explicit method of windows, 7-24
 :superior initialization option of windows and screens, 5-4
 :superior method of windows and screens, 5-4
 :suspend-crosshair method of gwin:mouse-handler-mixin, 12-49

T

:tab-nchars initialization option of windows, 7-26
 :tab-width initialization option of gwin:text, 12-68
 :tab-width method of gwin:text, 12-68
 :temporary-bit-array method of windows, 5-17
 :text-string initialization option of gwin:text, 12-68
 :text-string method of gwin:text, 12-68
 :thickness initialization option of gwin:basic-character-mixin, 12-70
 :thickness method of gwin:basic-character-mixin, 12-70
 :tick initialization option of gwin:world, 12-42
 :tick method of gwin:world, 12-42
 :time-between-moves initialization option of gwin:sprite-cursor, 12-54
 :time-between-moves method of gwin:sprite-cursor, 12-54
 :top initialization option of gwin:rectangle, 12-63
 :top initialization option of windows, 4-2
 :top method of gwin:rectangle, 12-63
 :top-flag initialization option of gwin:sprite-cursor, 12-53
 :top-flag method of gwin:sprite-cursor, 12-53
 :top-item method of w:text-scroll-window, 16-2
 :top-limit initialization option of gwin:sprite-cursor, 12-53
 :top-limit initialization option of gwin:world, 12-44
 :top-limit method of gwin:sprite-cursor, 12-53
 :top-limit method of gwin:world, 12-44
 :top-margin-size method of windows, 3-2
 :tracker-cursor initialization option of gwin:mouse-handler-mixin, 12-48
 :tracker-cursor method of gwin:mouse-handler-mixin, 12-48
 :transform initialization option of gwin:ruler, 12-74
 :transform initialization option of gwin:subpicture, 12-77
 :transform initialization option of w:transform-mixin, 12-47
 :transform method of gwin:ruler, 12-74
 :transform method of gwin:subpicture, 12-77
 :transform method of w:transform-mixin, 12-47
 :transform-deltas method of w:transform-mixin, 12-47
 :transform-point method of w:transform-mixin, 12-48
 :truncate-line-out-flag initialization option of w:line-truncating-mixin, 7-14
 :truncation initialization option of w:basic-scroll-window, 17-6
 :truncation method of w:basic-scroll-window, 17-6
 :turn-off-blinkers-for-typeout method of w:essential-window-with-typeout-mixin, 13-3
 :turn-off-crosshair method of gwin:mouse-handler-mixin, 12-49
 :turn-on-blinkers-for-typeout method of w:essential-window-with-typeout-mixin, 13-3
 :turn-on-crosshair method of gwin:mouse-handler-mixin, 12-49
 :tyi method of w:stream-mixin, A-23
 :tyi-no-hang method of w:stream-mixin, A-23
 :tyo method of windows, 7-5
 :tyo-right-margin-character method of windows, 7-14
 :typeout-window initialization option of w:essential-window-with-typeout-mixin, 13-3
 :typeout-window method of w:essential-window-with-typeout-mixin, 13-3

U

:ultimate-selection-substitute method of windows, 6-9
 :un-arrest method of w:select-mixin, 6-13
 :undraw method of graphic object, 12-55
 :undraw method of gwin:arc, 12-59

- :undraw method of gwin:circle, 12-60
- :undraw method of gwin:line, 12-61
- :undraw method of gwin:polyline, 12-62
- :undraw method of gwin:raster-object, 12-76
- :undraw method of gwin:rectangle, 12-63
- :undraw method of gwin:ruler, 12-74
- :undraw method of gwin:spline, 12-65
- :undraw method of gwin:subpicture, 12-77
- :undraw method of gwin:text, 12-69
- :undraw method of gwin:triangle, 12-66
- :undraw-crosshair method of gwin:mouse-handler-mixin, 12-49
- :undraw-picture-list method of gwin:draw-mixin, A-11
- :undraw-picture-list method of w:graphics-mixin, 12-10
- :unhighlight method of gwin:backgroundpic, 12-78
- :unhighlight method of gwin:basic-graphics-mixin, 12-57
- :unread-any method of w:stream-mixin, 8-9
- :unselected-choice-font initialization option of w:basic-choose-variable-values, 14-54
- :untransform-deltas method of w:transform-mixin, 12-47
- :untransform-point method of w:transform-mixin, 12-48
- :untyi method of w:stream-mixin, A-23
- :update method of gwin:sprite-cursor, 12-54
- :update-crosshair method of gwin:mouse-handler-mixin, 12-49
- :update-item-list method of tv:abstract-dynamic-item-list-mixin , A-18
- :update-label method of w:delayed-redisplay-label-mixin, 3-9

V

- :value-array initialization option of w:basic-scroll-window, 17-6
- :value-array method of w:basic-scroll-window, 17-6
- :value-font initialization option of w:basic-choose-variable-values, 14-54
- :variables initialization option of w:basic-choose-variable-values, 14-54
- :vert-spacing initialization option of gwin:font, 12-67
- :vert-spacing method of gwin:font, 12-67
- :vertical-spacing initialization option of gwin:basic-character-mixin, 12-70
- :vertical-spacing method of gwin:basic-character-mixin, 12-70
- :visibility initialization option of gwin:basic-cursor-mixin, 12-50
- :visibility initialization option of w:blinker, 10-3
- :visibility method of gwin:basic-cursor-mixin, 12-50
- :visibility method of w:blinker, 10-3
- :vsp initialization option of windows, 7-25
- :vsp method of windows, 7-25

W

- :wait-for-input-with-timeout method of w:stream-mixin, 8-11
- :weight initialization option of gwin:arc, 12-58
- :weight initialization option of gwin:circle, 12-59
- :weight initialization option of gwin:line, 12-60
- :weight initialization option of gwin:polyline, 12-61
- :weight initialization option of gwin:rectangle, 12-63
- :weight initialization option of gwin:ruler, 12-73
- :weight initialization option of gwin:spline, 12-64
- :weight initialization option of gwin:subpicture, 12-77
- :weight initialization option of gwin:text, 12-69
- :weight initialization option of gwin:triangle, 12-65
- :weight method of gwin:arc, 12-58
- :weight method of gwin:circle, 12-59
- :weight method of gwin:line, 12-60
- :weight method of gwin:polyline, 12-61
- :weight method of gwin:rectangle, 12-63

- :weight method of gwin:ruler, 12-73
- :weight method of gwin:subpicture, 12-77
- :weight method of gwin:text, 12-69
- :weight method of gwin:triangle, 12-65
- :who-line-documentation-string method of gwin:graphics-window-mixin, 12-46
- :who-line-documentation-string method of windows, 11-14
- :width initialization option of gwin:bitblt-blinker, 12-51
- :width initialization option of gwin:block-cursor, 12-52
- :width initialization option of gwin:raster-object, 12-75
- :width initialization option of gwin:rectangle, 12-63
- :width initialization option of gwin:sprite-cursor, 12-53
- :width initialization option of w:bitblt-blinker, 10-10
- :width initialization option of w:cache-window, 12-47
- :width initialization option of w:rectangular-blinker, 10-7
- :width initialization option of w:sprite-window, 12-47
- :width initialization option of windows, 4-2
- :width method of gwin:bitblt-blinker, 12-51
- :width method of gwin:block-cursor, 12-52
- :width method of gwin:rectangle, 12-63
- :width method of windows, 4-4
- :window initialization option of gwin:basic-cursor-mixin, 12-50
- :window method of gwin:basic-cursor-mixin, 12-50
- :world initialization option of gwin:graphics-window-mixin, 12-45
- :world method of gwin:graphics-window-mixin, 12-45
- :world-edges method of w:transform-mixin, 12-48
- :world-extents-window method of w:transform-mixin, 12-45
- :write-display-list method of gwin:world, 12-42

X

- :x initialization option of windows, 4-2
- :x-center initialization option of gwin:arc, 12-58
- :x-center initialization option of gwin:circle, 12-59
- :x-center method of gwin:arc, 12-58
- :x-center method of gwin:circle, 12-59
- :x-end initialization option of gwin:arc, 12-58
- :x-end initialization option of gwin:line, 12-60
- :x-end initialization option of gwin:ruler, 12-73
- :x-end initialization option of gwin:text, 12-69
- :x-end method of gwin:line, 12-60
- :x-end method of gwin:text, 12-69
- :x-max method of gwin:basic-graphics-mixin, 12-57
- :x-min method of gwin:basic-graphics-mixin, 12-57
- :x-offset initialization option of gwin:basic-cursor-mixin, 12-50
- :x-offset method of gwin:basic-cursor-mixin, 12-50
- :x-origin initialization option of gwin:subpicture, 12-77
- :x-points initialization option of gwin:polyline, 12-62
- :x-points initialization option of gwin:spline, 12-64
- :x-points initialization option of gwin:vector-character, 12-71
- :x-points method of gwin:polyline, 12-62
- :x-points method of gwin:vector-character, 12-71
- :x-pos initialization option of w:blinker, 10-5
- :x-pos method of w:blinker, 10-5
- :x-position initialization option of gwin:basic-cursor-mixin, 12-50
- :x-position method of gwin:basic-cursor-mixin, 12-50
- :x-scale initialization option of gwin:subpicture, 12-77
- :x-start initialization option of gwin:arc, 12-58
- :x-start initialization option of gwin:line, 12-60
- :x-start initialization option of gwin:ruler, 12-73

- :x-start initialization option of gwin:text, 12-69
- :x-start method of gwin:arc, 12-58
- :x-start method of gwin:line, 12-60
- :x-start method of gwin:text, 12-69
- :x-step initialization option of gwin:sprite-cursor, 12-53
- :x-step method of gwin:sprite-cursor, 12-53
- :x1 initialization option of gwin:triangle, 12-66
- :x1 method of gwin:triangle, 12-66
- :x2 initialization option of gwin:triangle, 12-66
- :x2 method of gwin:triangle, 12-66
- :x3 initialization option of gwin:triangle, 12-66
- :x3 method of gwin:triangle, 12-66
- :xscale initialization option of gwin:raster-object, 12-75
- :xstart initialization option of gwin:raster-object, 12-76

Y

- :y initialization option of windows, 4-2
- :y-center initialization option of gwin:arc, 12-58
- :y-center initialization option of gwin:circle, 12-59
- :y-center method of gwin:arc, 12-58
- :y-center method of gwin:circle, 12-59
- :y-end initialization option of gwin:arc, 12-58
- :y-end initialization option of gwin:line, 12-60
- :y-end initialization option of gwin:ruler, 12-73
- :y-end initialization option of gwin:text, 12-69
- :y-end method of gwin:line, 12-60
- :y-end method of gwin:text, 12-69
- :y-max method of gwin:basic-graphics-mixin, 12-57
- :y-min method of gwin:basic-graphics-mixin, 12-57
- :y-offset initialization option of gwin:basic-cursor-mixin, 12-50
- :y-offset method of gwin:basic-cursor-mixin, 12-50
- :y-origin initialization option of gwin:subpicture, 12-77
- :y-points initialization option of gwin:polyline, 12-62
- :y-points initialization option of gwin:spline, 12-64
- :y-points initialization option of gwin:vector-character, 12-71
- :y-points method of gwin:polyline, 12-62
- :y-points method of gwin:vector-character, 12-71
- :y-pos initialization option of w:blinker, 10-5
- :y-pos method of w:blinker, 10-5
- :y-position initialization option of gwin:basic-cursor-mixin, 12-50
- :y-position method of gwin:basic-cursor-mixin, 12-50
- :y-scale initialization option of gwin:subpicture, 12-77
- :y-start initialization option of gwin:arc, 12-58
- :y-start initialization option of gwin:line, 12-60
- :y-start initialization option of gwin:ruler, 12-73
- :y-start initialization option of gwin:text, 12-69
- :y-start method of gwin:arc, 12-58
- :y-start method of gwin:line, 12-60
- :y-step initialization option of gwin:sprite-cursor, 12-53
- :y-step method of gwin:sprite-cursor, 12-53
- :y1 initialization option of gwin:triangle, 12-66
- :y1 method of gwin:triangle, 12-66
- :y2 initialization option of gwin:triangle, 12-66
- :y2 method of gwin:triangle, 12-66
- :y3 initialization option of gwin:triangle, 12-66
- :y3 method of gwin:triangle, 12-66
- :yscale initialization option of gwin:raster-object, 12-75
- :ystart initialization option of gwin:raster-object, 12-76

Z

:zoom method of w:transform-mixin, 12-48

Variables
A

w: all-the-screens, 5-3
w: alu-add, 19-14, 19-16
w: alu-adds, 19-14, 19-16
w: alu-and, 12-4
w: alu-andca, 12-5
w: alu-avg, 19-14, 19-16
w: alu-back, 19-15, 19-16
w: alu-ior, 12-4
w: alu-max, 19-14, 19-16
w: alu-min, 19-14, 19-16
w: alu-seta, 12-4
w: alu-setz, 12-5
w: alu-sub, 19-14, 19-16
w: alu-subc, 19-14, 19-16
w: alu-transp, 19-15, 19-16
w: alu-xor, 12-4

B

w: beep, 18-5
w: *beep-types*, 18-5
w: *beeping-functions*, 18-5
w: *bidirectional-more-standard-message*, 7-13
w: bitmap-mouse-pathname, 11-3

C

sys: clipping-rectangle-bottom-edge, 12-32
sys: clipping-rectangle-left-edge, 12-32
sys: clipping-rectangle-right-edge, 12-32
sys: clipping-rectangle-top-edge, 12-32
w: color-alist, 19-5
printer: color-to-gray-scale-table, 19-22
w: *control-register*, 19-24
gwin: *current-cache-for-raster-objects*, 12-75

D

w: *default-background*, 19-7, 19-18
w: default-beep, 18-6
w: *default-blinker-offset*, 19-18
w: *default-border-color*, 19-18
w: *default-color-map*, 19-4
tv: *default-directory-pathname*, 14-43
w: *default-documentation-background*, 19-18
w: *default-documentation-foreground*, 19-18
w: *default-foreground*, 12-11, 19-7, 19-18
gwin: *default-gwin-fonts*, 12-67
w: *default-initial-colors*, 19-4
w: *default-menu-background*, 19-18
w: *default-menu-foreground*, 19-18
w: *default-menu-item-who-line-documentation-function*, 14-31
w: *default-menu-label-background*, 19-18
w: *default-menu-label-foreground*, 19-18

- w: *default-read-whostate*, 8-16
- w: default-screen, 5-3
- w: *default-scroll-bar-color*, 19-18
- w: *default-status-background*, 19-18
- w: *default-status-foreground*, 19-18
- w: *default-texture*, 12-11
- w: *default-w-fonts*, 12-66
- w: default-window-types-item-list, 18-19
- w: deferred-notifications, 18-3
- w: *deselected-process-priority*, 6-8

E

- w: *enable-typeout-window-borders*, 13-2

F

- w: flash-duration, 18-5
- w: *font-list*, 12-66

H

- w: *hollow-m-choice-box-p*, 14-66

I

- w: initial-lisp-listener, 18-10

K

- w: kbd-global-asynchronous-characters, 8-22
- w: kbd-intercepted-characters, 8-19
- w: kbd-last-activity-time, 8-3
- w: kbd-standard-asynchronous-characters, 8-21
- w: kbd-standard-intercepted-characters, 8-20
- w: kbd-tyi-hook, 8-20

L

- w: last-who-line-process, 18-16

M

- w: main-screen, 5-3
- w: margin-choice-abort-string, 14-22
- w: margin-choice-completion-string, 14-22
- w: menu-default-command-characters, 14-18
- w: menu-fill-breakage, 14-30
- w: menu-golden-ratio, 14-30
- w: menu-intercolumn-spacing, 14-31
- w: menu-interword-spacing, 14-31
- w: more-processing-global-enable, 7-13
- w: mouse-blinker, 11-17
- w: mouse-bounce-time, 11-3
- w: mouse-double-click-time, 11-3
- w: mouse-fast-motion-bitmap-time, 11-3
- w: mouse-fast-motion-cross-size, 11-3
- w: mouse-fast-motion-cross-time, 11-3
- w: mouse-fast-motion-speed, 11-3
- w: mouse-fast-track-bitmap-mouse-p, 11-3
- w: mouse-handedness, 11-4
- w: *mouse-incrementing-keystates*, 11-4

w: mouse-last-buttons, 11-5
 w: mouse-sheet, 11-2
 w: mouse-speed, 11-2
 w: mouse-window, 11-8
 sys: mouse-x, 11-2
 sys: mouse-y, 11-2

N

w: number-of-who-line-documentation-lines, 18-15

O

w: on-volume, 18-8

P

w: pending-notifications, 18-3
 w: previously-selected-windows, 6-7

R

w: rubout-handler, 8-6

S

w: screen-manage-update-permitted-windows, 5-21
 w: *scroll-bar-char-index*, 11-27
 w: *scroll-bar-char-x-offset*, 11-27
 w: *scroll-bar-char-y-offset*, 11-27
 w: *scroll-bar-default-clicks*, 11-28
 w: scroll-bar-max-exit-speed, 11-29
 w: scroll-bar-max-speed, 11-29
 w: scroll-bar-reluctance, 11-29
 w: *scroll-bar-shade*, 11-27
 w: *scroll-bar-who-line-documentation*, 11-29
 w: scroll-item-leader-offset, 17-11
 w: *selected-process-priority*, 6-7
 w: selected-window, 6-2
 w: sheet-area, 5-2
 w: *system-keys*, 8-23
 w: *system-menu-debug-tools-column*, 18-19
 w: *system-menu-edit-windows-column*, 18-19
 w: *system-menu-programs-column*, 18-19
 w: *system-menu-user-aids-column*, 18-19

T

w: *terminal-keys*, 8-22
 w: *textures*, 12-11, 19-19

U

w: *unidirectional-more-standard-message*, 7-13
 w: use-kbd-buttons, 11-3
 w: *user-defined-terminal-keys*, 8-22

W

w: who-line-file-state-sheet, 18-17
 w: who-line-mouse-grabbed-documentation, 11-9
 w: who-line-process, 18-16
 w: who-line-screen, 5-3

- w: window-owning-mouse, 11-8
- w: window-resource-names, 18-21

Data Systems Group - Austin Documentation Questionnaire

Explorer Window System Reference

Do you use other TI manuals? If so, which one(s)?

_____	_____
_____	_____
_____	_____

How would you rate the quality of our manuals?

	Excellent	Good	Fair	Poor
Accuracy	_____	_____	_____	_____
Organization	_____	_____	_____	_____
Clarity	_____	_____	_____	_____
Completeness	_____	_____	_____	_____
Overall design	_____	_____	_____	_____
Size	_____	_____	_____	_____
Illustrations	_____	_____	_____	_____
Examples	_____	_____	_____	_____
Index	_____	_____	_____	_____
Binding method	_____	_____	_____	_____

Was the quality of documentation a criterion in your selection of hardware or software?

- Yes No

How do you find the technical level of our manuals?

- Written for a more experienced user than yourself
 Written for a user with the same experience
 Written for a less experienced user than yourself

What is your experience using computers?

- Less than 1 year 1-5 years 5-10 years Over 10 years

We appreciate your taking the time to complete this questionnaire. If you have additional comments about the quality of our manuals, please write them in the space below. Please be specific.

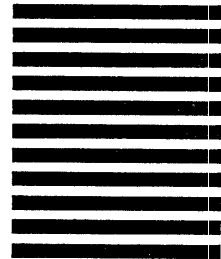
Name _____ Title/Occupation _____
Company Name _____
Address _____ City/State/Zip _____
Telephone _____ Date _____

TAPE EDGE TO SEAL

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST-CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DATA SYSTEMS GROUP
ATTN: PUBLISHING CENTER
P.O. Box 2909 M/S 2146
Austin, Texas 78769-9990



FOLD