

VMS

---

digital

VMS DECwindows  
Xlib Routines Reference Manual:  
Part I

Order Number: AA-MG26A-TE

# **VMS DECwindows Xlib Routines Reference Manual**

Order Numbers: Part I: AA-MG26A-TE  
Part II: AA-MG27A-TE

**December 1988**

This manual describes the VMS DECwindows Xlib programming routines.

**Revision/Update Information:** This is a new manual.

**Software Version:** VMS Version 5.1

**digital equipment corporation  
maynard, massachusetts**

---

**December 1988**

---

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

---


© Digital Equipment Corporation 1988.

All Rights Reserved.  
Printed in U.S.A.

---

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDA	MASSBUS	VAX RMS
DDIF	PrintServer 40	VAXstation
DEC	Q-bus	VMS
DECnet	ReGIS	VT
DECUS	ULTRIX	XUI
DECwindows	UNIBUS	
DIGITAL	VAX	
LN03	VAXcluster	

The following are third-party trademarks:

PostScript is a registered trademark of Adobe Systems, Inc.

X Window System, Version 10 and its derivations (X, X10, X Version 10, X Window System) are trademarks of the Massachusetts Institute of Technology.

X Window System, Version 11 and its derivations (X, X11, X Version 11, X Window System) are trademarks of the Massachusetts Institute of Technology.

---

## Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.





---

# Contents

---

PREFACE

xxv

---

## CHAPTER 1 INTRODUCTION TO THE DECWINDOWS XLIB ROUTINES 1-1

---

1.1	ROUTINE DOCUMENTATION FORMAT	1-4
1.1.1	Routine Name _____	1-4
1.1.2	Overview _____	1-4
1.1.3	VAX Format _____	1-4
1.1.4	MIT C Format _____	1-8
1.1.5	Returns _____	1-8
1.1.6	Argument Information _____	1-9
1.1.7	Description _____	1-9
1.1.8	X Errors _____	1-9
1.2	DATA STRUCTURES	1-9
1.3	PROTOCOL REQUEST AND UTILITY ROUTINES	1-10

---

---

## CHAPTER 2 DISPLAY ROUTINES 2-1

---

2.1	DISPLAY ROUTINES	2-4
	ALL PLANES	2-5
	BITMAP BIT ORDER	2-6
	BITMAP PAD	2-7
	BITMAP UNIT	2-8
	BLACK PIXEL	2-9
	BLACK PIXEL OF SCREEN	2-10
	CELLS OF SCREEN	2-11
	CLOSE DISPLAY	2-12
	CONNECTION NUMBER	2-14
	DEFAULT COLORMAP	2-15
	DEFAULT COLORMAP OF SCREEN	2-16
	DEFAULT DEPTH	2-17
	DEFAULT DEPTH OF SCREEN	2-18
	DEFAULT GC	2-19

## Contents

DEFAULT GC OF SCREEN	2-20
DEFAULT ROOT WINDOW	2-21
DEFAULT SCREEN	2-22
DEFAULT SCREEN OF DISPLAY	2-23
DEFAULT VISUAL	2-24
DEFAULT VISUAL OF SCREEN	2-25
DISPLAY CELLS	2-26
DISPLAY HEIGHT	2-27
DISPLAY HEIGHT MM	2-28
DISPLAY NAME	2-29
DISPLAY OF SCREEN	2-31
DISPLAY PLANES	2-32
DISPLAY STRING	2-33
DISPLAY WIDTH	2-35
DISPLAY WIDTH MM	2-36
DOES BACKING STORE	2-37
DOES SAVE UNDERS	2-38
EVENT MASK OF SCREEN	2-39
FREE	2-40
HEIGHT MM OF SCREEN	2-41
HEIGHT OF SCREEN	2-42
IMAGE BYTE ORDER	2-43
LAST KNOWN REQUEST PROCESSED	2-44
MAX CMAPS OF SCREEN	2-45
MIN CMAPS OF SCREEN	2-46
NEXT REQUEST	2-47
NO OP	2-48
OPEN DISPLAY	2-49
PLANES OF SCREEN	2-51
PROTOCOL REVISION	2-52
PROTOCOL VERSION	2-53
Q LENGTH	2-54
ROOT WINDOW	2-55
ROOT WINDOW OF SCREEN	2-56
ROTATE BUFFERS	2-57
SCREEN COUNT	2-59
SCREEN OF DISPLAY	2-60
SERVER VENDOR	2-61
STORE BUFFER	2-63
STORE BYTES	2-65
VENDOR RELEASE	2-67
WHITE PIXEL	2-68

WHITE PIXEL OF SCREEN	2-69
WIDTH MM OF SCREEN	2-70
WIDTH OF SCREEN	2-71

---

**CHAPTER 3 WINDOW ROUTINES** 3-1

---

3.1	SET WINDOW ATTRIBUTES DATA STRUCTURE	3-3
-----	--------------------------------------	-----

---

3.2	WINDOW CHANGES DATA STRUCTURE	3-7
-----	-------------------------------	-----

---

3.3	WINDOW ATTRIBUTES DATA STRUCTURE	3-9
-----	----------------------------------	-----

---

3.4	WINDOW ROUTINES	3-14
-----	-----------------	------

CHANGE WINDOW ATTRIBUTES	3-15
CIRCULATE SUBWINDOWS	3-18
CIRCULATE SUBWINDOWS DOWN	3-20
CIRCULATE SUBWINDOWS UP	3-22
CONFIGURE WINDOW	3-24
CREATE SIMPLE WINDOW	3-27
CREATE WINDOW	3-31
DESTROY SUBWINDOWS	3-35
DESTROY WINDOW	3-37
GET GEOMETRY	3-39
GET WINDOW ATTRIBUTES	3-42
LOWER WINDOW	3-44
MAP RAISED	3-46
MAP SUBWINDOWS	3-47
MAP WINDOW	3-48
MOVE RESIZE WINDOW	3-50
MOVE WINDOW	3-53
QUERY POINTER	3-55
QUERY TREE	3-58
RAISE WINDOW	3-61
RESIZE WINDOW	3-62
RESTACK WINDOWS	3-64
SET WINDOW BACKGROUND	3-66
SET WINDOW BACKGROUND PIXMAP	3-68
SET WINDOW BORDER	3-70
SET WINDOW BORDER PIXMAP	3-72

## Contents

SET WINDOW BORDER WIDTH	3-74
TRANSLATE COORDINATES	3-75
UNMAP SUBWINDOWS	3-78
UNMAP WINDOW	3-79

---

## CHAPTER 4 EVENT ROUTINES 4-1

---

4.1	EVENT DATA STRUCTURE	4-3
4.2	THE EVENT MASK	4-4
4.3	THE PREDICATE PROCEDURE	4-6
4.4	TIME COORDINATE DATA STRUCTURE	4-6
4.5	ERROR HANDLING	4-7
4.6	ERROR EVENT DATA STRUCTURE	4-9
4.7	EVENT ROUTINES	4-11
	CHECK IF EVENT	4-12
	CHECK MASK EVENT	4-14
	CHECK TYPED EVENT	4-16
	CHECK TYPED WINDOW EVENT	4-18
	CHECK WINDOW EVENT	4-20
	EVENTS QUEUED	4-22
	FLUSH	4-24
	GET ERROR DATABASE TEXT	4-25
	GET ERROR TEXT	4-27
	GET MOTION EVENTS	4-29
	IF EVENT	4-32
	MASK EVENT	4-34
	NEXT EVENT	4-36
	PEEK EVENT	4-37
	PEEK IF EVENT	4-38
	PENDING	4-40
	PUT BACK EVENT	4-41
	SELECT ASYNC EVENT	4-42

SELECT ASYNC INPUT	4-44
SELECT INPUT	4-46
SEND EVENT	4-48
SET AFTER FUNCTION	4-51
SET ERROR HANDLER	4-53
SET IO ERROR HANDLER	4-55
SYNC	4-56
SYNCHRONIZE	4-57
WINDOW EVENT	4-59

---

**CHAPTER 5 GRAPHICS CONTEXT ROUTINES** 5-1

---

**5.1 THE GC VALUES DATA STRUCTURE** 5-2

---

**5.2 GC MASK** 5-22

---

**5.3 GRAPHICS CONTEXT ROUTINES** 5-24

CHANGE GC	5-25
COPY GC	5-27
CREATE GC	5-29
FREE GC	5-32
GCONTEXT FROM GC	5-33
QUERY BEST SIZE	5-34
QUERY BEST STIPPLE	5-37
QUERY BEST TILE	5-40
SET ARC MODE	5-43
SET BACKGROUND	5-45
SET CLIP MASK	5-47
SET CLIP ORIGIN	5-49
SET CLIP RECTANGLES	5-51
SET DASHES	5-54
SET FILL RULE	5-57
SET FILL STYLE	5-60
SET FONT	5-62
SET FOREGROUND	5-64
SET FUNCTION	5-66
SET GRAPHICS EXPOSURES	5-68
SET LINE ATTRIBUTES	5-70
SET PLANE MASK	5-75
SET STATE	5-77

# Contents

SET STIPPLE	5-80
SET SUBWINDOW MODE	5-82
SET TILE	5-84
SET TS ORIGIN	5-86

---

## CHAPTER 6 GRAPHICS ROUTINES 6-1

---

6.1	POINT DATA STRUCTURE	6-2
-----	----------------------	-----

---

6.2	SEGMENT DATA STRUCTURE	6-3
-----	------------------------	-----

---

6.3	RECTANGLE DATA STRUCTURE	6-4
-----	--------------------------	-----

---

6.4	ARC DATA STRUCTURE	6-5
-----	--------------------	-----

---

6.5	IMAGE DATA STRUCTURE	6-6
-----	----------------------	-----

---

6.6	GRAPHICS ROUTINES	6-10
-----	-------------------	------

ADD PIXEL	6-11
CLEAR AREA	6-12
CLEAR WINDOW	6-15
COPY AREA	6-17
COPY PLANE	6-21
CREATE IMAGE	6-25
DESTROY IMAGE	6-29
DRAW ARC	6-30
DRAW ARCS	6-34
DRAW LINE	6-37
DRAW LINES	6-40
DRAW POINT	6-45
DRAW POINTS	6-47
DRAW RECTANGLE	6-51
DRAW RECTANGLES	6-55
DRAW SEGMENTS	6-58
FILL ARC	6-61
FILL ARCS	6-64
FILL POLYGON	6-67
FILL RECTANGLE	6-72
FILL RECTANGLES	6-75

GET IMAGE	6-78
GET PIXEL	6-81
GET SUBIMAGE	6-83
PUT IMAGE	6-87
PUT PIXEL	6-90
SUBIMAGE	6-92

---

**CHAPTER 7 TEXT ROUTINES** 7-1

---

7.1	<b>DRAWING TEXT</b>	7-2
7.1.1	Text Item 8-Bit Data Structure	7-2
7.1.2	Text Item 16-Bit Data Structure	7-3
<hr/>		
7.2	<b>TEXT ROUTINES</b>	7-4
	DRAW IMAGE STRING	7-5
	DRAW IMAGE STRING 16	7-8
	DRAW STRING	7-11
	DRAW STRING 16	7-14
	DRAW TEXT	7-17
	DRAW TEXT 16	7-20
	QUERY TEXT EXTENTS	7-23
	QUERY TEXT EXTENTS 16	7-26
	TEXT EXTENTS	7-29
	TEXT EXTENTS 16	7-32
	TEXT WIDTH	7-35
	TEXT WIDTH 16	7-37

---

**CHAPTER 8 PROPERTY ROUTINES** 8-1

---

8.1	<b>SIZE HINTS DATA STRUCTURE</b>	8-3
<hr/>		
8.2	<b>ICON SIZE DATA STRUCTURE</b>	8-6
<hr/>		
8.3	<b>WINDOW MANAGER HINTS DATA STRUCTURE</b>	8-8



## Contents

---

8.4	PROPERTY ROUTINES	8-10
	CHANGE PROPERTY	8-11
	CONVERT SELECTION	8-14
	DELETE CONTEXT	8-16
	DELETE PROPERTY	8-18
	FETCH BUFFER	8-20
	FETCH BYTES	8-22
	FETCH NAME	8-24
	FIND CONTEXT	8-26
	GET ATOM NAME	8-28
	GET CLASS HINT	8-30
	GET ICON NAME	8-32
	GET ICON SIZES	8-34
	GET NORMAL HINTS	8-37
	GET SELECTION OWNER	8-39
	GET SIZE HINTS	8-41
	GET TRANSIENT FOR HINT	8-43
	GET WINDOW PROPERTY	8-45
	GET WM HINTS	8-50
	GET ZOOM HINTS	8-52
	INTERN ATOM	8-54
	LIST PROPERTIES	8-56
	ROTATE WINDOW PROPERTIES	8-58
	SAVE CONTEXT	8-60
	SET CLASS HINT	8-62
	SET COMMAND	8-64
	SET ICON NAME	8-66
	SET ICON SIZES	8-68
	SET NORMAL HINTS	8-70
	SET SELECTION OWNER	8-72
	SET SIZE HINTS	8-74
	SET STANDARD PROPERTIES	8-76
	SET TRANSIENT FOR HINT	8-79
	SET WM HINTS	8-81
	SET ZOOM HINTS	8-83
	STORE NAME	8-85
	UNIQUE CONTEXT	8-87

---

**CHAPTER 9 REGION ROUTINES** 9-1

---

9.1 **RECTANGLE DATA STRUCTURE** 9-2

---

9.2 **REGION ROUTINES** 9-3

CLIP BOX	9-4
CREATE REGION	9-5
DESTROY REGION	9-6
EMPTY REGION	9-7
EQUAL REGION	9-8
INTERSECT REGION	9-9
OFFSET REGION	9-11
POINT IN REGION	9-12
POLYGON REGION	9-14
RECT IN REGION	9-17
SET REGION	9-19
SHRINK REGION	9-20
SUBTRACT REGION	9-22
UNION RECT WITH REGION	9-24
UNION REGION	9-26
XOR REGION	9-28

---

**CHAPTER 10 WINDOW AND SESSION MANAGER ROUTINES** 10-1

---

10.1 **NETWORK DATA STRUCTURE** 10-5

---

10.2 **KEYBOARD CONTROL DATA STRUCTURE** 10-6

10.2.1 **Keyboard Control Value Mask** \_\_\_\_\_ 10-8

---

10.3 **KEYBOARD STATE DATA STRUCTURE** 10-10

---

10.4 **COMPOSE DATA STRUCTURE** 10-12

---

10.5 **MODIFIER KEY MAP DATA STRUCTURE** 10-12

---

## Contents

---

<b>10.6</b>	<b>WINDOW AND SESSION MANAGER ROUTINES</b>	<b>10-13</b>
	ACTIVATE SCREEN SAVER	10-14
	ADD HOST	10-15
	ADD HOSTS	10-17
	ADD TO SAVE SET	10-19
	ALLOW EVENTS	10-21
	AUTO REPEAT OFF	10-25
	AUTO REPEAT ON	10-26
	BELL	10-27
	CHANGE ACTIVE POINTER GRAB	10-29
	CHANGE KEYBOARD CONTROL	10-32
	CHANGE KEYBOARD MAPPING	10-34
	CHANGE POINTER CONTROL	10-37
	CHANGE SAVE SET	10-39
	DELETE MODIFIERMAP ENTRY	10-41
	DISABLE ACCESS CONTROL	10-43
	ENABLE ACCESS CONTROL	10-45
	FORCE SCREEN SAVER	10-47
	FREE MODIFIERMAP	10-49
	GEOMETRY	10-50
	GET DEFAULT	10-54
	GET INPUT FOCUS	10-56
	GET KEYBOARD CONTROL	10-58
	GET KEYBOARD MAPPING	10-59
	GET MODIFIER MAPPING	10-62
	GET POINTER CONTROL	10-64
	GET POINTER MAPPING	10-66
	GET SCREEN SAVER	10-68
	GRAB BUTTON	10-71
	GRAB KEY	10-77
	GRAB KEYBOARD	10-81
	GRAB POINTER	10-84
	GRAB SERVER	10-89
	INSERT MODIFIERMAP ENTRY	10-90
	INSTALL COLORMAP	10-93
	KEYCODE TO KEYSYM	10-95
	KEYSYM TO KEYCODE	10-96
	KEYSYM TO STRING	10-97
	KILL CLIENT	10-98
	LIST HOSTS	10-100
	LIST INSTALLED COLORMAPS	10-102
	LOOKUP KEYSYM	10-104

LOOKUP STRING	10-105
NEW MODIFIER MAP	10-107
PARSE COLOR	10-108
PARSE GEOMETRY	10-112
QUERY KEYMAP	10-115
REBIND KEYSYM	10-116
REFRESH KEYBOARD MAPPING	10-118
REMOVE FROM SAVE SET	10-119
REMOVE HOST	10-121
REMOVE HOSTS	10-123
REPARENT WINDOW	10-125
RESET SCREEN SAVER	10-128
SET ACCESS CONTROL	10-129
SET CLOSE DOWN MODE	10-131
SET INPUT FOCUS	10-133
SET MODIFIER MAPPING	10-136
SET POINTER MAPPING	10-138
SET SCREEN SAVER	10-140
STRING TO KEYSYM	10-143
UNGRAB BUTTON	10-144
UNGRAB KEY	10-146
UNGRAB KEYBOARD	10-148
UNGRAB POINTER	10-149
UNGRAB SERVER	10-150
UNINSTALL COLORMAP	10-151
WARP POINTER	10-153

---

**CHAPTER 11 PIXMAP AND BITMAP ROUTINES**

11-1

---

11.1	PIXMAP AND BITMAP ROUTINES	11-1
	CREATE BITMAP FROM DATA	11-2
	CREATE PIXMAP	11-4
	CREATE PIXMAP FROM BITMAP DATA	11-6
	FREE PIXMAP	11-8
	READ BITMAP FILE	11-9
	WRITE BITMAP FILE	11-12

## Contents

---

<b>CHAPTER 12</b>	<b>COLOR ROUTINES</b>	<b>12-1</b>
<hr/>		
12.1	STANDARD COLOR MAP DATA STRUCTURE	12-2
<hr/>		
12.2	COLOR DEFINITION DATA STRUCTURE	12-5
<hr/>		
12.3	COLOR ROUTINES	12-7
	ALLOC COLOR	12-8
	ALLOC COLOR CELLS	12-10
	ALLOC COLOR PLANES	12-13
	ALLOC NAMED COLOR	12-17
	COPY COLORMAP AND FREE	12-20
	CREATE COLORMAP	12-22
	FREE COLORMAP	12-25
	FREE COLORS	12-27
	GET STANDARD COLORMAP	12-29
	GET VISUAL INFO	12-32
	LOOKUP COLOR	12-35
	MATCH VISUAL INFO	12-38
	QUERY COLOR	12-40
	QUERY COLORS	12-42
	SET STANDARD COLORMAP	12-44
	SET WINDOW COLORMAP	12-46
	STORE COLOR	12-48
	STORE COLORS	12-50
	STORE NAMED COLOR	12-52
<hr/>		
<b>CHAPTER 13</b>	<b>FONT ROUTINES</b>	<b>13-1</b>
<hr/>		
13.1	FONT DATA STRUCTURE	13-2
<hr/>		
13.2	CHARACTER DATA STRUCTURE	13-6
<hr/>		
13.3	FONT PROPERTY DATA STRUCTURE	13-8

---

13.4	<b>FONT ROUTINES</b>	13-9
	FREE FONT	13-10
	FREE FONT INFO	13-11
	FREE FONT NAMES	13-12
	FREE FONT PATH	13-13
	GET CHAR STRUCT	13-14
	GET FONT PATH	13-15
	GET FONT PROPERTY	13-17
	LIST FONT	13-19
	LIST FONT WITH INFO	13-21
	LIST FONTS	13-23
	LIST FONTS WITH INFO	13-25
	LOAD FONT	13-27
	LOAD QUERY FONT	13-29
	QUERY FONT	13-31
	SET FONT PATH	13-33
	UNLOAD FONT	13-35

---

<b>CHAPTER 14</b>	<b>CURSOR ROUTINES</b>	14-1
-------------------	------------------------	------

---

14.1	<b>CURSOR ROUTINES</b>	14-1
	CREATE FONT CURSOR	14-2
	CREATE GLYPH CURSOR	14-4
	CREATE PIXMAP CURSOR	14-7
	DEFINE CURSOR	14-10
	FREE CURSOR	14-12
	QUERY BEST CURSOR	14-13
	RECOLOR CURSOR	14-15
	UNDEFINE CURSOR	14-17

---

<b>CHAPTER 15</b>	<b>RESOURCE MANAGER ROUTINES</b>	15-1
-------------------	----------------------------------	------

---

15.1	<b>THE RESOURCE MANAGER</b>	15-2
------	-----------------------------	------

---

15.2	<b>RESOURCE MANAGER MATCHING RULES</b>	15-3
------	--	------

## Contents

---

15.3	QUARKS	15-4
------	--------	------

---

15.4	THE RESOURCE MANAGER VALUE DATA STRUCTURE	15-5
------	---	------

---

15.5	RESOURCE MANAGER ROUTINES	15-6
	PERMALLOC	15-7
	RM GET FILE DATABASE	15-8
	RM GET RESOURCE	15-9
	RM GET STRING DATABASE	15-11
	RM INITIALIZE	15-12
	RM MERGE DATABASES	15-13
	RM PARSE COMMAND	15-14
	RM PUT FILE DATABASE	15-16
	RM PUT LINE RESOURCE	15-17
	RM PUT RESOURCE	15-18
	RM PUT STRING RESOURCE	15-20
	RM Q GET RESOURCE	15-21
	RM Q GET SEARCH LIST	15-23
	RM Q GET SEARCH RESOURCE	15-25
	RM Q PUT RESOURCE	15-27
	RM Q PUT STRING RESOURCE	15-29
	RM QUARK TO STRING	15-31
	RM STRING TO BIND QUARK LIST	15-32
	RM STRING TO QUARK	15-33
	RM STRING TO QUARK LIST	15-34
	RM UNIQUE QUARK	15-35

---

## INDEX

---

## FIGURES

3-1	Set Window Attributes Data Structure (VAX Binding)	3-3
3-2	Set Window Attributes Data Structure (MIT C Binding)	3-6
3-3	Window Changes Data Structure (VAX Binding)	3-8
3-4	Window Changes Data Structure (MIT C Binding)	3-9
3-5	Window Attributes Data Structure (VAX Binding)	3-9
3-6	Window Attributes Data Structure (MIT C Binding)	3-13
4-1	Event Data Structure (VAX Binding)	4-3
4-2	Event Data Structure (MIT C Binding)	4-4

4-3	Time Coordinate Data Structure (VAX Binding) _____	4-6
4-4	Time Coordinate Data Structure (MIT C Binding) _____	4-7
4-5	Error Event Data Structure (VAX Binding) _____	4-10
4-6	Error Event Data Structure (MIT C Binding) _____	4-10
5-1	GC Values Data Structure (VAX Binding) _____	5-3
5-2	Line Styles _____	5-10
5-3	Cap Styles _____	5-11
5-4	Join Styles _____	5-12
5-5	Fill Rules _____	5-13
5-6	Arc Fill Options _____	5-13
5-7	GC Values Data Structure (MIT C Binding) _____	5-15
5-8	Arc Fill Options _____	5-44
5-9	Dash Offset and Dash List _____	5-55
5-10	Odd Dash List _____	5-56
5-11	Fill Rules _____	5-59
5-12	Line Styles _____	5-71
5-13	Cap Styles _____	5-72
5-14	Join Styles _____	5-73
6-1	Point Data Structure (VAX Binding) _____	6-2
6-2	Point Data Structure (MIT C Binding) _____	6-3
6-3	Segment Data Structure (VAX Binding) _____	6-3
6-4	Segment Data Structure (MIT C Binding) _____	6-4
6-5	Rectangle Data Structure (VAX Binding) _____	6-4
6-6	Rectangle Data Structure (MIT C Binding) _____	6-5
6-7	Arc Data Structure (VAX Binding) _____	6-5
6-8	Arc Data Structure (MIT C Binding) _____	6-6
6-9	Image Data Structure (VAX Binding) _____	6-7
6-10	Image Data Structure (MIT C Binding) _____	6-9
6-11	Rectangular Area Cleared _____	6-14
6-12	Specifying an Arc _____	6-32
6-13	Lines Drawn in Different Line Modes _____	6-43
6-14	Points Drawn in Different Coordinate Modes _____	6-49
6-15	Outline of a Rectangle _____	6-53
6-16	Polygon Shapes Drawn in Different Coordinate Modes _____	6-70
7-1	Text Item Data Structure (VAX Binding) _____	7-2
7-2	Text Item Data Structure (MIT C Binding) _____	7-3
7-3	Text Item 16 Data Structure (VAX Binding) _____	7-3
7-4	Text Item 16 Data Structure (MIT C Binding) _____	7-4
8-1	Size Hints Data Structure (VAX Binding) _____	8-3
8-2	Size Hints Data Structure (MIT C Binding) _____	8-5



## Contents

8-3	Icon Size Data Structure (VAX Binding) _____	8-7
8-4	Icon Size Data Structure (MIT C Binding) _____	8-8
8-5	WM Hints Data Structure (VAX Binding) _____	8-8
8-6	WM Hints Data Structure (MIT C Binding) _____	8-10
9-1	Rectangle Data Structure (VAX Binding) _____	9-2
9-2	Rectangle Data Structure (MIT C Binding) _____	9-3
9-3	Region Intersection _____	9-10
9-4	Point Data Structure (VAX Binding) _____	9-15
9-5	Point Data Structure (MIT C Binding) _____	9-16
9-6	Shrinking a Region _____	9-21
9-7	Subtracting a Region _____	9-23
9-8	Union of a Source Region and a Rectangle _____	9-25
9-9	Union of Two Regions _____	9-27
9-10	Exclusive OR Operation _____	9-29
10-1	Network Data Structure (VAX Binding) _____	10-5
10-2	Network Data Structure (MIT C Binding) _____	10-5
10-3	Keyboard Control Data Structure (VAX Binding) _____	10-6
10-4	Keyboard Control Data Structure (MIT C Binding) _____	10-7
10-5	Keyboard State Data Structure (VAX Binding) _____	10-10
10-6	Keyboard State Data Structure (MIT C Binding) _____	10-11
10-7	Compose Data Structure (VAX Binding) _____	10-12
10-8	Compose Data Structure (MIT C Binding) _____	10-12
10-9	Modifier Key Map Data Structure (VAX Binding) _____	10-13
10-10	Modifier Key Map Data Structure (MIT C Binding) _____	10-13
10-11	Color Definition Data Structure (VAX Binding) _____	10-110
10-12	Color Definition Data Structure (MIT C Binding) _____	10-111
12-1	Standard Color Map Data Structure (VAX Binding) _____	12-3
12-2	Standard Color Map Data Structure (MIT C Binding) _____	12-4
12-3	Color Definition Data Structure (VAX Binding) _____	12-6
12-4	Color Definition Data Structure (MIT C Binding) _____	12-7
13-1	Font Data Structure (VAX Binding) _____	13-2
13-2	Font Data Structure (MIT C binding) _____	13-5
13-3	Character Data Structure (VAX Binding) _____	13-7
13-4	Character Data Structure (MIT C Binding) _____	13-8
13-5	Font Property Data Structure (VAX Binding) _____	13-9
13-6	Font Property Data Structure (MIT C Binding) _____	13-9
15-1	Resource Manager Value Data Structure (VAX Binding) _____	15-5
15-2	Resource Manager Value Data Structure (MIT C Binding) _____	15-6

---

**TABLES**

1-1	General Rules of Syntax _____	1-4
1-2	VAX Usage Entries _____	1-5
1-3	Access Entries _____	1-7
1-4	Mechanism Entries _____	1-8
1-5	Protocol and Utility Routines _____	1-10
2-1	Display Routines _____	2-1
3-1	Window Routines _____	3-1
3-2	Members of the Set Window Attributes Data Structure (VAX Binding) _____	3-3
3-3	Default Values of the Set Window Attributes Structure _____	3-5
3-4	Members of the Set Window Attributes Structure (MIT C Binding) _____	3-6
3-5	Members of the Window Changes Data Structure (VAX Binding) _____	3-8
3-6	Members of the Window Changes Data Structure (MIT C Binding) _____	3-9
3-7	Members of the Window Attributes Data Structure (VAX Binding) _____	3-10
3-8	Members of the Window Attributes Data Structure (MIT C Binding) _____	3-13
3-9	CHANGE WINDOW ATTRIBUTES Flags _____	3-16
3-10	Change Mask Bits _____	3-25
4-1	Event Routines _____	4-1
4-2	Event Mask Elements _____	4-5
4-3	Members of the Time Coordinate Data Structure (VAX Binding) _____	4-7
4-4	Members of the Time Coordinate Data Structure (MIT C Binding) _____	4-7
4-5	Xlib Error Codes _____	4-8
4-6	Members of the Error Event Data Structure (VAX Binding) _____	4-10
4-7	Members of the Error Event Data Structure (MIT C Binding) _____	4-11
5-1	Graphics Context Routines _____	5-1
5-2	Members of the GC Values Data Structure (VAX Binding) _____	5-4
5-3	Default Values for the GC Values Data Structure _____	5-14
5-4	Members of the GC Values Data Structure (MIT C Binding) _____	5-15
5-5	Default Values for the GC Values Data Structure _____	5-21
5-6	GC Mask Bits _____	5-22
5-7	Graphics Context Codes for Function Member _____	5-67

## Contents

5-8	Graphics Context Codes for Function Member _____	5-78
6-1	Graphics Routines _____	6-1
6-2	Members of the Point Data Structure (VAX Binding) _____	6-3
6-3	Members of the Point Data Structure (MIT C Binding) _____	6-3
6-4	Members of the Rectangle Data Structure (VAX Binding) _____	6-4
6-5	Members of the Rectangle Data Structure (MIT C Binding) _____	6-5
6-6	Members of the Arc Data Structure (VAX Binding) _____	6-5
6-7	Members of the Arc Data Structure (MIT C Binding) _____	6-6
6-8	Members of the Image Data Structure (VAX Binding) _____	6-8
6-9	Members of the Image Data Structure (MIT C Binding) _____	6-9
7-1	Text Routines _____	7-1
7-2	Members of the Text Item Data Structure (VAX Binding) _____	7-2
7-3	Members of the Text Item Data Structure (MIT C Binding) _____	7-3
7-4	Members of the Text Item 16 Data Structure (VAX Binding) _____	7-3
7-5	Members of the Text Item 16 Data Structure (MIT C Binding) _____	7-4
8-1	Property Routines _____	8-1
8-2	Members of the Size Hints Data Structure (VAX Binding) _____	8-4
8-3	Members of the Size Hints Data Structure (MIT C Binding) _____	8-5
8-4	Members of the Icon Size Data Structure (VAX Binding) _____	8-7
8-5	Members of the Icon Size Data Structure (MIT C Binding) _____	8-8
8-6	Members of the WM Hints Data Structure (VAX Binding) _____	8-9
8-7	Members of the WM Hints Data Structure (MIT C Binding) _____	8-10
9-1	Region Routines _____	9-1
9-2	Members of the Rectangle Data Structure (VAX Binding) _____	9-3
9-3	Members of the Rectangle Data Structure (MIT C Binding) _____	9-3
9-4	Fill Rule Constants _____	9-15
9-5	Members of the Point Data Structure (VAX Binding) _____	9-16
9-6	Members of the Point Data Structure (MIT C Binding) _____	9-16
10-1	Window and Session Manager Routines _____	10-1
10-2	Members of the Network Data Structure (VAX Binding) _____	10-5
10-3	Members of the Network Data Structure (MIT C Binding) _____	10-6
10-4	Members of the Keyboard Control Data Structure (VAX Binding) _____	10-7
10-5	Members of the Keyboard Control Data Structure (MIT C Binding) _____	10-7
10-6	Keyboard Control Value Mask _____	10-8
10-7	Members of the Keyboard State Data Structure (VAX Binding) _____	10-10
10-8	Members of the Keyboard State Data Structure (MIT C Binding) _____	10-11
10-9	Members of the Compose Data Structure (VAX Binding) _____	10-12

10-10	Members of the Compose Data Structure (MIT C Binding) _____	10-12
10-11	Members of the Modifier Key Map Data Structure (VAX Binding) _____	10-13
10-12	Members of the Modifier Key Map Data Structure (MIT C Binding) _____	10-13
10-13	Event Mask Description _____	10-30
10-14	Parse Mask Bits _____	10-51
10-15	Event Mask Description _____	10-73
10-16	Event Mask Description _____	10-85
10-17	Adding a Key Code to a Zero Value _____	10-92
10-18	Adding a Key Code to a Nonzero Value _____	10-92
10-19	Members of the Color Definition Data Structure (VAX Binding) _____	10-110
10-20	Members of the Color Definition Data Structure (MIT C Binding) _____	10-111
10-21	Parse Mask Bits _____	10-113
11-1	Pixmap and Bitmap Routines _____	11-1
12-1	Color Routines _____	12-1
12-2	Members of the Standard Color Map Data Structure (VAX Binding) _____	12-3
12-3	Members of the Standard Color Map Data Structure (MIT C Binding) _____	12-4
12-4	Members of the Color Definition Data Structure (VAX Binding) _____	12-6
12-5	Members of the Color Definition Data Structure (MIT C Binding) _____	12-7
12-6	Visual Information Mask Bits _____	12-33
13-1	Window and Session Font Routines _____	13-1
13-2	Members of the Font Data Structure (VAX Binding) _____	13-3
13-3	Members of the Font Data Structure (MIT C Binding) _____	13-5
13-4	Members of the Character Data Structure (VAX Binding) _____	13-7
13-5	Members of the Character Data Structure (MIT C Binding) _____	13-8
13-6	Members of the Font Property Data Structure (VAX Binding) _____	13-9
13-7	Members of the Font Property Data Structure (MIT C Binding) _____	13-9
14-1	Window and Session Cursor Routines _____	14-1
15-1	Resource Manager Routines _____	15-1
15-2	Members of the Resource Manager Value Data Structure (VAX Binding) _____	15-5
15-3	Members of the Resource Manager Value Data Structure (MIT C Binding) _____	15-6



---

# Preface

---

## Intended Audience

This document is for experienced programmers who will be creating applications with VMS DECwindows. This document assumes a knowledge of VAX calling standards and of the C programming language.

---

## Document Structure

This document is composed of an Introduction chapter and fourteen Reference chapters.

The Introduction chapter provides information on the documentation format for the VMS DECwindows graphics programming routines.

The Reference chapters provide the following information on each programming routine:

- How to call the routine according to the VMS calling standard for any VAX supported language.
- How to call the routine according to the MIT C programming standards.
- A description of each argument in the routine.
- A description of how the routine functions.
- A list of values returned by the routine.

This document is in two parts. Part I contains the Introduction chapter and the Reference chapters for the following sets of routines:

- Display routines
- Window routines
- Event routines
- Graphics context routines
- Graphics routines
- Text routines

Part II contains the Reference chapters for the following sets of routines:

- Property routines
- Region routines
- Window and session manager routines
- Pixmap and bitmap routines

## Preface

- Color routines
- Font routines
- Cursor routines
- Resource manager routines

---

## Associated Documents

Readers may find the following documents useful when programming with DECwindows Xlib routines:

- *VMS DECwindows Xlib Programming Volume*—Describes how to program DECwindows routines using the VAX and MIT C bindings
- *VMS DECwindows Guide to Application Programming*—As well as offering a guide to programming the DECwindows toolkit, this manual provides an overview of programming in the DECwindows environment
- *XUI Style Guide*—Describes the standard DECwindows user interface

---

## Conventions

The following conventions are used in this manual:

mouse	The term <i>mouse</i> is used to refer to any pointing device, such as a mouse, a puck, or a stylus.
...	In examples, a horizontal ellipsis indicates one of the following possibilities: <ul style="list-style-type: none"><li>• Additional optional arguments in a statement have been omitted.</li><li>• The preceding item or items can be repeated one or more times.</li><li>• Additional parameters, values, or other information can be entered.</li></ul>
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
()	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[]	In format descriptions, brackets indicate that whatever is enclosed is optional; you can select none, one, or all of the choices.
{}	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.

**boldface text**

Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason.

UPPERCASE TEXT

Uppercase letters indicate the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.

numbers

Unless otherwise noted, all numbers in the text are assumed to be decimal. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.





# 1

---

## Introduction to the DECwindows Xlib Routines

The VMS DECwindows Xlib programming library routines provide the programming interface to the X Window System, Version 11. Many of these routines provide input directly to or receive output directly from the X server and are therefore the lowest level of programming interface. Other sets of VMS DECwindows tools are built on top of these routines to provide higher levels of programming interface.

The VMS DECwindows Xlib library (hereafter referred to as Xlib) is a subset of the MIT X Window System Release 2 Xlib with several additions.

Xlib provides a complete implementation of the Xlib C Language X Interface (Release 2) with the exception of the X Window System, Version 10 compatibility routines and the X extension routines. The following routines and macros are *not* provided in Xlib:

X Version 10 compatibility routines:

- XCreateAssocTable
- XDeleteAssoc
- XDestroyAssocTable
- XDraw
- XDrawFilled
- XLookupAssoc
- XMakeAssocTable

X Version 11 extension routines/macros:

- Data
- GetReq
- GetReqExtra
- GetResReq
- GetEmptyReq
- LockDisplay
- PackData
- SyncHandle
- UnlockDisplay
- XAddToExtensionList
- XAllocID
- Xcalloc
- XESetCloseDisplay
- XESetCreateFont
- XESetCreateGC
- XESetCopyGC
- XESetError
- XESetErrorString
- XESetEventToWire
- XESetFlushGC
- XESetFreeGC
- XESetFreeFont

# Introduction to the DECwindows Xlib Routines

XESetWireToEvent  
XFindOnExtensionList  
XFlushGC  
Xfree  
XFreeExtensionList  
XInitExtension  
XListExtensions  
Xmalloc  
XQueryExtension  
\_XAllocScratch  
\_XFlushGCCache  
\_XRead  
\_XReply  
\_XSend

Xlib provides several additions to the MIT Xlib, including support for asynchronous programming and entry points that conform to the VAX Procedure Calling Standard. You can perform Xlib calls at Asynchronous System Trap (AST) level, and these can be mixed with other calls at non-AST level. Xlib also extends the LOOKUP STRING routine to support DIGITAL international keyboards and the ISO Latin 1 character set.

The routines are organized according to function. Within each functional group, they are organized alphabetically. The categories of Xlib routines are as follows:

- Display routines connect and disconnect the server, send and retrieve data to and from the server, and obtain default information about the display with which you are working.
- Window routines create, map, move, change, and destroy windows. This category also includes routines that obtain information about a window.
- Event and error handling routines select and check input/output events and handle error conditions.
- Graphics context routines create, change, and delete graphics context structures and attributes.
- Graphics routines draw, fill, and erase points, lines, polygons, and arcs. This category also includes routines that handle complex images.
- Text routines manipulate text in a window.
- Property routines create, use, and destroy window properties. This category also includes context routines.
- Region routines create, use, and destroy regions within a window.
- Window and session management routines, which are special purpose routines normally only used by the window manager and session manager.
- Pixmap and bitmap routines create and free pixmaps and bitmaps.
- Color routines manipulate color maps and allocate, define, and use colors. This category also includes visual information routines.

# Introduction to the DECwindows Xlib Routines

- Font routines specify, load, and free fonts.
- Cursor routines create, use, and delete cursors.
- Resource manager routines store, retrieve, and search for resources.

For each functional group of routines, an overview section at the beginning of each chapter describes the routines in general. The overview section also includes descriptions of each predefined data structure used within that functional group.

Most routines can be formatted and called according to the following bindings:

- VAX binding—Where the routine follows the VAX Procedure Calling Standard
- C binding—Where the routine follows the MIT C language binding

Xlib can be used by programs written in any language that follows the VAX Procedure Calling Standard. Xlib library definition files are provided for the following 8 languages and may be found in `SYS$LIBRARY`:

- VAX Ada
- VAX BASIC
- VAX BLISS
- VAX C
- VAX FORTRAN
- VAX MACRO
- VAX Pascal
- VAX PL/I

Xlib programs that desire to retain binary compatibility with future releases of DECwindows should not be coded to use the MIT C language macros. Instead they should use the corresponding routine calls. No commitment is made to retain binary compatibility for programs that use the macro interface.

Xlib should not be used by VAX C language programs that have been compiled with the `/G_FLOAT` qualifier. Xlib has been compiled with the default `/D_FLOAT` qualifier and should not be intermixed with VAX C `/G_FLOAT` modules.

If your program is going to use other portions of code written in another VAX language, use the VAX binding to ensure that the VAX Procedure Calling Standard is met. If the programming language you are using cannot easily support the style used in many of the MIT C binding routines (such as parameters passed by value and structure types used), you might want to use the VAX binding. If the programming language you are using easily accommodates the MIT C language binding (for example, C and perhaps VAX BLISS), use the MIT C binding.

# Introduction to the DECwindows Xlib Routines

## 1.1 Routine Documentation Format

---

### 1.1 Routine Documentation Format

Each Xlib routine is documented for both bindings using a standard template. This section describes each category in this template, the information that is presented under each category, and the format used to present the information.

The following subsections describe each part in the documentation format.

---

#### 1.1.1 Routine Name

The generic name of the routine that is used to refer to the routine for both the VAX or C bindings.

---

#### 1.1.2 Overview

A statement describing what the routine does. The overview is relevant to both the VAX and C bindings.

---

#### 1.1.3 VAX Format

The VAX routine call format. The format syntax is summarized in Table 1–1.

**Table 1–1 General Rules of Syntax**

Element	Syntax Rule
Routine entry point name	The name is shown in all capital letters with the prefix X. The routine entry point name is required.
Equals sign	For specific return values, the equals sign is required.
Parentheses	Open and close parentheses surround the argument list in a routine call. They are required.
Argument names	Argument names, including names of return arguments, are always shown in lowercase characters. All arguments not enclosed by brackets ([ ]) are required. They must be listed in the same order in your program as they appear in the format.
Spaces	One or more spaces are shown between the entry point name and the first argument, and between each argument. They are not required.
Brackets	Brackets ([ ]) surround optional arguments. Commas that appear inside brackets are optional and appear only when the optional argument appears. The brackets are not required.
Commas	Commas must appear between required arguments. When commas appear inside brackets ([ ]), they appear only when the optional argument appears.

The VAX format is followed by a table that lists information about each argument listed in the format. Arguments are described in the order they

# Introduction to the DECwindows Xlib Routines

## 1.1 Routine Documentation Format

appear in the format. The table lists the argument name, usage, data type, access, and mechanism for each argument.

The following information is provided for each argument:

### Usage

The Usage column specifies the general VAX binding argument type. For example, if the argument is a resource identifier, the Usage field is “identifier.” This field provides additional information about the argument that is helpful when declaring the argument within a program. Refer to Table 1–2 for a list of usage entries used in Xlib routines.

**Table 1–2 VAX Usage Entries**

Entry	Description
address	Unsigned longword containing the virtual address of data or code, but not of a procedure entry mask (which is a procedure entry).
array	An array with the specific description of its elements provided in the argument description.
Boolean	Unsigned longword with the predefined values of 1 for true and 0 for false.
byte	A signed byte integer.
uns byte	An unsigned byte integer.
char string	String of 0 to 65,535 8-bit characters.
cond value	Unsigned longword specifying that a predefined condition value will be returned in R0.
identifier	A value used to refer to a resource. It is originally returned by the system. In the VAX binding, an identifier is a <i>longword integer (unsigned)</i> .
longword	A signed longword integer
uns longword	An unsigned longword integer
mask longword	An unsigned longword interpreted as a bit mask.
procedure	Entry mask to a procedure.
record	A data structure with the specific description of the structure provided in the argument description.
word	A signed word integer
uns word	An unsigned word integer

### Data Type

The data type column specifies the standard VAX data type of the argument. For example, if the argument is a resource identifier, the data type is “longword (unsigned).” If the argument has a predefined value, it is provided in the description of the argument. The following are the VAX data type entries used in the Xlib routines:

- byte
- character string

# Introduction to the DECwindows Xlib Routines

## 1.1 Routine Documentation Format

- longword
- longword (unsigned), notated as uns longword
- vector longword
- word (signed)
- word (unsigned), notated as uns word
- The name of a VAX data structure. The following are the valid VAX data structure names that can appear in this column:

```
x$any_event
x$arc
x$button_event
x$char_2b
x$char_struct
x$circ_request_event
x$circulate_event
x$class_hint
x$clie_message_event
x$color
x$colormap_event
x$compose_status
x$conf_request_event
x$configure_event
x$creat_window_event
x$crossing_event
x$depth
x$destr_window_event
x$display
x$error_event
x$event
x$expose_event
x$ext_codes
x$ext_data
x$focus_change_event
x$font_prop
x$font_struct
x$frame
x$gc_struct
x$gc_values
x$graph_expose_event
x$gravity_event
x$host_address
x$icon_size
x$image
x$key_event
x$keyboard_control
x$keyboard_state
x$keymap_event
x$map_event
x$map_request_event
x$mapping_event
x$modifier_keymap
```

# Introduction to the DECwindows Xlib Routines

## 1.1 Routine Documentation Format

x\$motion\_event  
x\$no\_expose\_event  
x\$point  
x\$property\_event  
x\$rectangle  
x\$reparent\_event  
x\$resource\_req  
x\$resz\_request\_event  
x\$rm\_value  
x\$screen  
x\$screen\_format  
x\$segment  
x\$sel\_request\_event  
x\$select\_clear\_event  
x\$selection\_event  
x\$set\_win\_attributes  
x\$size\_hints  
x\$standard\_colormap  
x\$text\_item  
x\$text\_item\_16  
x\$time\_coord  
x\$unmap\_event  
x\$visibility\_event  
x\$visual  
x\$visual\_info  
x>window\_attributes  
x>window\_changes  
x\$wm\_hints

### Access

The access column specifies the way in which the called routine accesses the argument. For example, when the argument is passed as input, the access is “read only” for both bindings and when the argument is returned by the routine, the access is “write only” for both bindings. Refer to Table 1–3 for a list of access entries used in the Xlib routines.

**Table 1–3 Access Entries**

Entry	Description
read	Input data needed by the routine to perform its operation must be readable. When an argument specifies input data, the access entry is readable. The routine cannot write data back to this argument.
write	Output data returned by the routine to a specific location. When an argument specifies output data, the access entry is writable. The routine does not read the contents of the location either before or after it writes into the location.

(continued on next page)



# Introduction to the DECwindows Xlib Routines

## 1.1 Routine Documentation Format

**Table 1–3 (Cont.) Access Entries**

Entry	Description
read/write or modify	The routine reads the input data, which it uses in its operation, and then overwrites the input data with the results (the output data) of the operation. Thus, when the routine completes execution, the input data specified by the argument is lost.

### Mechanism

The mechanism column specifies the passing mechanism used by the called routine. For example, when the argument is the value itself, the mechanism is “by value” and when the argument is a pointer to the value, the mechanism is “by reference.” Refer to Table 1–4 for a complete list of mechanism entries used in the Xlib routines.

**Table 1–4 Mechanism Entries**

Entry	Description
value	The argument contains the actual data to be used by the routine. Note that because an argument is only one longword in length, only data that can be represented in one longword can be passed by value.
reference	The argument contains the address of the data to be used by the routine. The argument is a pointer to the actual data.
descriptor	The argument contains the address of a descriptor. A descriptor consists of two or more longwords (depending on the type of descriptor used), which describe the location, length, and the VAX standard data type of the data to be used by the called routine. The argument is a pointer to a descriptor that itself is a pointer to the actual data.

### 1.1.4 MIT C Format

The section contains the MIT C routine call format. The MIT C format follows the same conventions listed in Table 1–1, with the exception of the routine entry point name. In MIT C, the name is shown in initial capitals with the prefix X. The format is followed by the MIT C declarations for each argument.

### 1.1.5 Returns

Most routines return a value. This section describes what the returned value specifies. In the VAX binding, the first item in the argument table describes how to declare the return value. In the MIT C binding, the declaration for the return value is the first item in the declaration list. If a routine does not have a return value, this section is omitted.

# Introduction to the DECwindows Xlib Routines

## 1.1 Routine Documentation Format

---

### 1.1.6 Argument Information

Detailed descriptions of each argument follow the Returns section. These descriptions cover what the arguments specify and any predefined values.

---

### 1.1.7 Description

This section describes how to use the routine. Generally, the topics covered can include what the routine is used for, how it is processed, other routines to use before or after it, other routines to use to accomplish similar tasks, and any other useful routine-specific information. For a more tutorial description of how to use the Xlib routines, see *VMS DECwindows Xlib Programming Volume*.

---

### 1.1.8 X Errors

This section lists the X errors that the routine can return. An X error event is generated by the server when it detects erroneous or inconsistent information in a client request. Error events contain an error code that tells the client what problem the server has with the request; for instance, X\$C\_BAD\_VALUE or BadValue would indicate a problem with the value of some setting in the request. The X errors section contains a table that lists the VAX and MIT C names for the error and a description of the error.

The X error description sections for utility routines list only the first-level errors that the routines can generate. The utility routines may indirectly generate additional errors if they call protocol requests that fail.

---

## 1.2 Data Structures

Predefined data structures are provided for complex data structures used by Xlib routines. The data structures are presented in the overview section for each functional group of routines. They are presented in two formats:

- VAX binding format—The data structure in a generic VAX language. The generic language provides the VAX binding names for the data structure and individual fields within the data structure. It also provides the basic VAX data type for each field. The data type used for a specific language must be subsequently derived. For information on how to derive the data type for each supported VAX language, see the *Introduction to VMS System Routines* in the VMS documentation set.
- MIT C binding format—The data structure in the MIT C programming language.

For consistency, this manual refers to fields in data structures as *members*.

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

### 1.3 Protocol Request and Utility Routines

Although many Xlib routines make protocol requests directly, some do not. Xlib routines that do make protocol requests directly are called the *protocol* routines; those that do not are called the *utility* routines. The utility routines make certain programming tasks easier.

For example, you can create clip areas using two routines: SET CLIP RECTANGLES or SET REGION. SET CLIP RECTANGLES is a protocol routine because it calls the SetClipRectangles protocol request directly. SET REGION is a utility routine because it calls the Xlib routine SET CLIP RECTANGLES, which then calls the SetClipRectangles protocol request.

Table 1–5 lists each Xlib routine. If the routine makes a direct protocol request, the name of the protocol request is listed. If it is a utility routine, the term “Utility” appears. Only the Xlib routines are documented in this reference manual, not the protocol requests.

**Table 1–5 Protocol and Utility Routines**

Xlib Routine	Protocol Request or Utility Routine
ACTIVATE SCREEN SAVER	ForceScreenSaver
ADD HOST	ChangeHosts
ADD HOSTS	ChangeHosts
ADD PIXEL	Utility
ADD TO SAVE SET	ChangeSaveSet
ALLOC COLOR	AllocColor
ALLOC COLOR CELLS	AllocColorCells
ALLOC COLOR PLANES	AllocColorPlanes
ALLOC NAMED COLOR	AllocNamedColor
ALLOW EVENTS	AllowEvents
ALL PLANES	Utility
AUTO REPEAT OFF	ChangeKeyboardControl
AUTO REPEAT ON	ChangeKeyboardControl
BELL	Bell
BITMAP BIT ORDER	Utility
BITMAP PAD	Utility
BITMAP UNIT	Utility
BLACK PIXEL	Utility
BLACK PIXEL OF SCREEN	Utility
CELLS OF SCREEN	Utility
CHANGE ACTIVE POINTER GRAB	ChangeActivePointerGrab
CHANGE GC	ChangeGC

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

Xlib Routine	Protocol Request or Utility Routine
CHANGE KEYBOARD CONTROL	ChangeKeyboardControl
CHANGE KEYBOARD MAPPING	ChangeKeyboardMapping
CHANGE POINTER CONTROL	ChangePointerControl
CHANGE PROPERTY	ChangeProperty
CHANGE SAVE SET	ChangeSaveSet
CHANGE WINDOW ATTRIBUTES	ChangeWindowAttributes
CHECK IF EVENT	Utility
CHECK MASK EVENT	Utility
CHECK TYPED EVENT	Utility
CHECK TYPED WINDOW EVENT	Utility
CHECK WINDOW EVENT	Utility
CIRCULATE SUBWINDOWS	CirculateWindow
CIRCULATE SUBWINDOWS DOWN	CirculateWindow
CIRCULATE SUBWINDOWS UP	CirculateWindow
CLEAR AREA	ClearArea
CLEAR WINDOW	ClearArea
CLIP BOX	Utility
CLOSE DISPLAY	Utility
CONFIGURE WINDOW	ConfigureWindow
CONNECTION NUMBER	Utility
CONVERT SELECTION	ConvertSelection
COPY AREA	CopyArea
COPY COLORMAP AND FREE	CopyColormapAndFree
COPY GC	CopyGC
COPY PLANE	CopyPlane
CREATE BITMAP FROM DATA	CreateGC, CreatePixmap, FreeGC, PutImage
CREATE COLORMAP	CreateColormap
CREATE FONT CURSOR	CreateGlyphCursor
CREATE GC	CreateGC
CREATE GLYPH CURSOR	CreateGlyphCursor
CREATE IMAGE	Utility
CREATE PIXMAP	CreatePixmap
CREATE PIXMAP CURSOR	CreateCursor
CREATE PIXMAP FROM BITMAP DATA	CreateGC, CreatePixmap, FreeGC, PutImage
CREATE REGION	Utility

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
CREATE SIMPLE WINDOW	Create Window
CREATE WINDOW	CreateWindow
DEFAULT COLORMAP	Utility
DEFAULT COLORMAP OF SCREEN	Utility
DEFAULT DEPTH	Utility
DEFAULT DEPTH OF SCREEN	Utility
DEFAULT GC	Utility
DEFAULT GC OF SCREEN	Utility
DEFAULT ROOT WINDOW	Utility
DEFAULT SCREEN	Utility
DEFAULT SCREEN OF DISPLAY	Utility
DEFAULT VISUAL	Utility
DEFAULT VISUAL OF SCREEN	Utility
DEFINE CURSOR	ChangeWindowAttributes
DELETE CONTEXT	Utility
DELETE MODIFIER MAP ENTRY	Utility
DELETE PROPERTY	DeleteProperty
DESTROY IMAGE	Utility
DESTROY REGION	Utility
DESTROY SUBWINDOWS	DestroySubwindows
DESTROY WINDOW	DestroyWindow
DISABLE ACCESS CONTROL	SetAccessControl
DISPLAY CELLS	Utility
DISPLAY HEIGHT	Utility
DISPLAY HEIGHT MM	Utility
DISPLAY NAME	Utility
DISPLAY OF SCREEN	Utility
DISPLAY PLANES	Utility
DISPLAY STRING	Utility
DISPLAY WIDTH	Utility
DISPLAY WIDTH MM	Utility
DOES BACKING STORE	Utility
DOES SAVE UNDERS	Utility
DRAW ARC	PolyArc
DRAW ARCS	PolyArc
DRAW IMAGE STRING	ImageText8

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

Xlib Routine	Protocol Request or Utility Routine
DRAW IMAGE STRING 16	ImageText16
DRAW LINE	PolySegment
DRAW LINES	PolyLine
DRAW POINT	PolyPoint
DRAW POINTS	PolyPoint
DRAW RECTANGLE	PolyRectangle
DRAW RECTANGLES	PolyRectangle
DRAW SEGMENTS	PolySegment
DRAW STRING	PolyText8
DRAW STRING 16	PolyText16
DRAW TEXT	PolyText8
DRAW TEXT 16	PolyText16
EMPTY REGION	Utility
ENABLE ACCESS CONTROL	SetAccessControl
EQUAL REGION	Utility
EVENT MASK OF SCREEN	Utility
EVENTS QUEUED	Utility
FETCH BUFFER	Utility
FETCH BYTES	GetProperty
FETCH NAME	GetProperty
FILL ARC	PolyFillArc
FILL ARCS	PolyFillArc
FILL POLYGON	FillPoly
FILL RECTANGLE	PolyFillRectangle
FILL RECTANGLES	PolyFillRectangle
FIND CONTEXT	Utility
FLUSH	Utility
FORCE SCREEN SAVER	ForceScreenSaver
FREE	Utility
FREE COLORMAP	FreeColormap
FREE COLORS	FreeColors
FREE CURSOR	FreeCursor
FREE FONT	CloseFont
FREE FONT INFO (MIT C only)	Utility
FREE FONT NAMES (MIT C only)	Utility
FREE FONT PATH (MIT C only)	Utility

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
FREE GC	FreeGC
FREE MODIFIERMAP	Utility
FREE PIXMAP	FreePixmap
GCONTEXT FROM GC	Utility
GEOMETRY	Utility
GET ATOM NAME	GetAtomName
GET CHAR INFO (VAX only)	Utility
GET CHAR STRUCT (VAX only)	Utility
GET CLASS HINT	Utility
GET DEFAULT	Utility
GET ERROR DATABASE TEXT	Utility
GET ERROR TEXT	Utility
GET FONT PATH	GetFontPath
GET FONT PROPERTY	Utility
GET GEOMETRY	GetGeometry
GET ICON NAME	Utility
GET ICON SIZES	GetProperty
GET IMAGE	GetImage
GET INPUT FOCUS	GetInputFocus
GET KEYBOARD CONTROL	GetKeyboardControl
GET KEYBOARD MAPPING	GetKeyboardMapping
GET MODIFIER MAPPING	GetModifierMapping
GET MOTION EVENTS	GetMotionEvents
GET NORMAL HINTS	GetProperty
GET PIXEL	Utility
GET POINTER CONTROL	GetPointerControl
GET POINTER MAPPING	GetPointerMapping
GET SCREEN SAVER	SetScreenSaver
GET SELECTION OWNER	GetSelectionOwner
GET SIZE HINTS	GetProperty
GET STANDARD COLORMAP	Utility
GET SUBIMAGE	Utility
GET TRANSIENT FOR HINT	Utility
GET VISUAL INFO	Utility
GET WINDOW ATTRIBUTES	GetWindowAttributes
GET WINDOW PROPERTY	GetProperty

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
GET WM HINTS	GetProperty
GET ZOOM HINTS	GetProperty
GRAB BUTTON	GrabButton
GRAB KEY	GrabKey
GRAB KEYBOARD	GrabKeyboard
GRAB POINTER	GrabPointer
GRAB SERVER	GrabServer
HEIGHT MM OF SCREEN	Utility
HEIGHT OF SCREEN	Utility
IF EVENT	Utility
IMAGE BYTE ORDER	Utility
INSERT MODIFIERMAP ENTRY	Utility
INSTALL COLORMAP	InstallColormap
INTERN ATOM	InternAtom
INTERSECT REGION	Utility
KEYCODE TO KEYSYM	Utility
KEYSYM TO KEYCODE	Utility
KEYSYM TO STRING	Utility
KILL CLIENT	KillClient
LAST KNOWN REQUEST PROCESSED	Utility
LIST FONT (VAX only)	ListFonts
LIST FONTS	ListFonts
LIST FONT WITH INFO (VAX only)	ListFontsWithInfo
LIST FONTS WITH INFO	ListFontsWithInfo
LIST HOSTS	ListHosts
LIST INSTALLED COLORMAPS	ListInstalledColormaps
LIST PROPERTIES	ListProperties
LOAD FONT	OpenFont
LOAD QUERY FONT	OpenFont, QueryFont
LOOKUP COLOR	LookupColor
LOOKUP KEYSYM	Utility
LOOKUP STRING	Utility
LOWER WINDOW	ConfigureWindow
MAP RAISED	ConfigureWindow, MapWindow
MAP SUBWINDOWS	MapSubwindows

(continued on next page)



# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
MAP WINDOW	MapWindow
MASK EVENT	Utility
MATCH VISUAL INFO	Utility
MAX CMAPS OF SCREEN	Utility
MIN CMAPS OF SCREEN	Utility
MOVE RESIZE WINDOW	ConfigureWindow
MOVE WINDOW	ConfigureWindow
NEW MODIFIERMAP	Utility
NEXT EVENT	Utility
NEXT REQUEST	Utility
NO OP	NoOperation
OFFSET REGION	Utility
OPEN DISPLAY	CreateGC
PARSE COLOR	LookupColor
PARSE GEOMETRY	Utility
PEEK EVENT	Utility
PEEK IF EVENT	Utility
PENDING	Utility
PERMALLOC	Utility
PLANES OF SCREEN	Utility
POINT IN REGION	Utility
POLYGON REGION	Utility
PROTOCOL REVISION	Utility
PROTOCOL VERSION	Utility
PUT BACK EVENT	Utility
PUT IMAGE	PutImage
PUT PIXEL	Utility
Q LENGTH	Utility
QUERY BEST CURSOR	QueryBestSize
QUERY BEST SIZE	QueryBestSize
QUERY BEST STIPPLE	QueryBestSize
QUERY BEST TILE	QueryBestSize
QUERY COLOR	QueryColors
QUERY COLORS	QueryColors
QUERY FONT	Utility
QUERY KEYMAP	Utility

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

Xlib Routine	Protocol Request or Utility Routine
QUERY POINTER	QueryPointer
QUERY TEXT EXTENTS	QueryTextExtents
QUERY TEXT EXTENTS 16	QueryTextExtents
QUERY TREE	QueryTree
RAISE WINDOW	ConfigureWindow
READ BITMAP FILE	CreateGC, CreatePixmap, FreeGC, PutImage
REBIND KEYSYM	Utility
RECOLOR CURSOR	RecolorCursor
RECT IN REGION	Utility
REFRESH KEYBOARD MAPPING	Utility
REMOVE FROM SAVE SET	ChangeSaveSet
REMOVE HOST	ChangeHosts
REMOVE HOSTS	ChangeHosts
REPARANT WINDOW	ReparentWindow
RESET SCREEN SAVER	ForceScreenSaver
RESIZE WINDOW	ConfigureWindow
RESTACK WINDOWS	ConfigureWindow
RM GET FILE DATABASE	Utility
RM GET RESOURCE	Utility
RM GET STRING DATABASE	Utility
RM INITIALIZE	Utility
RM MERGE DATABASES	Utility
RM PARSE COMMAND	Utility
RM PUT FILE DATABASE	Utility
RM PUT LINE RESOURCE	Utility
RM PUT RESOURCE	Utility
RM PUT STRING RESOURCE	Utility
RM Q GET RESOURCE	Utility
RM Q GET SEARCH LIST	Utility
RM Q GET SEARCH RESOURCE	Utility
RM Q PUT RESOURCE	Utility
RM Q PUT STRING RESOURCE	Utility
RM QUARK TO STRING	Utility
RM STRING TO BIND QUARK LIST	Utility
RM STRING TO QUARK	Utility
RM STRING TO QUARK LIST	Utility

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
RM UNIQUE QUARK	Utility
ROOT WINDOW	Utility
ROOT WINDOW OF SCREEN	Utility
ROTATE BUFFERS	RotateProperties
ROTATE WINDOW PROPERTIES	RotateProperties
SAVE CONTEXT	Utility
SCREEN COUNT	Utility
SCREEN OF DISPLAY	Utility
SELECT ASYNC EVENT	Utility
SELECT ASYNC INPUT	Utility
SELECT INPUT	ChangeWindowAttributes
SEND EVENT	SendEvent
SERVER VENDOR	Utility
SET ACCESS CONTROL	SetAccessControl
SET AFTER FUNCTION	Utility
SET ARC MODE	ChangeGC
SET BACKGROUND	ChangeGC
SET CLASS HINT	Utility
SET CLIP MASK	ChangeGC
SET CLIP ORIGIN	ChangeGC
SET CLIP RECTANGLES	SetClipRectangles
SET CLOSE DOWN MODE	SetCloseDownMode
SET COMMAND	ChangeProperty
SET DASHES	SetDashes
SET ERROR HANDLER	Utility
SET FILL RULE	ChangeGC
SET FILL STYLE	ChangeGC
SET FONT	ChangeGC
SET FONT PATH	SetFontPath
SET FOREGROUND	ChangeGC
SET FUNCTION	ChangeGC
SET GRAPHICS EXPOSURES	ChangeGC
SET ICON NAME	Utility
SET ICON SIZES	ChangeProperty
SET INPUT FOCUS	SetInputFocus
SET IO ERROR HANDLER	Utility

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
SET LINE ATTRIBUTES	ChangeGC
SET MODIFIER MAPPING	SetModifierMapping
SET NORMAL HINTS	ChangeProperty
SET PLANE MASK	ChangeGC
SET POINTER MAPPING	SetPointerMapping
SET REGION	Utility
SET SCREEN SAVER	SetScreenSaver
SET SELECTION OWNER	SetSelectionOwner
SET SIZE HINTS	ChangeProperty
SET STANDARD COLORMAP	Utility
SET STANDARD PROPERTIES	ChangeProperty
SET STATE	ChangeGC
SET STIPPLE	ChangeGC
SET SUBWINDOW MODE	ChangeGC
SET TILE	ChangeGC
SET TRANSIENT FOR HINT	Utility
SET TS ORIGIN	ChangeGC
SET WINDOW BACKGROUND	ChangeWindowAttributes
SET WINDOW BACKGROUND PIXMAP	ChangeWindowAttributes
SET WINDOW BORDER	ChangeWindowAttributes
SET WINDOW BORDER PIXMAP	ChangeWindowAttributes
SET WINDOW BORDER WIDTH	ConfigureWindow
SET WINDOW COLORMAP	ChangeWindowAttributes
SET WM HINTS	ChangeProperty
SET ZOOM HINTS	ChangeProperty
SHRINK REGION	Utility
STORE BUFFER	ChangeProperty
STORE BYTES	ChangeProperty
STORE COLOR	StoreColors
STORE COLORS	StoreColors
STORE NAME	ChangeProperty
STORE NAMED COLOR	StoreNamedColor
STRING TO KEYSYM	Utility
SUBIMAGE	Utility
SUBTRACT REGION	Utility

(continued on next page)

# Introduction to the DECwindows Xlib Routines

## 1.3 Protocol Request and Utility Routines

**Table 1–5 (Cont.) Protocol and Utility Routines**

<b>Xlib Routine</b>	<b>Protocol Request or Utility Routine</b>
SYNC	GetInputFocus
SYNCHRONIZE	Utility
TEXT EXTENTS	Utility
TEXT EXTENTS 16	Utility
TEXT WIDTH	Utility
TEXT WIDTH 16	Utility
TRANSLATE COORDINATES	TranslateCoordinates
UNDEFINE CURSOR	ChangeWindowAttributes
UNGRAB BUTTON	UngrabButton
UNGRAB KEY	UngrabKey
UNGRAB KEYBOARD	UngrabKeyboard
UNGRAB POINTER	UngrabPointer
UNGRAB SERVER	UngrabServer
UNINSTALL COLORMAP	UninstallColormap
UNION RECT WITH REGION	Utility
UNION REGION	Utility
UNIQUE CONTEXT	Utility
UNLOAD FONT	CloseFont
UNMAP SUBWINDOWS	UnmapSubwindows
UNMAP WINDOW	UnmapWindow
VENDOR RELEASE	Utility
WARP POINTER	WarpPointer
WHITE PIXEL	Utility
WHITE PIXEL OF SCREEN	Utility
WIDTH MM OF SCREEN	Utility
WIDTH OF SCREEN	Utility
WINDOW EVENT	Utility
WRITE BITMAP FILE	Utility
XOR REGION	Utility

# 2

## Display Routines

---

This chapter includes routines that open and close a display, provide information about an open display, and manipulate buffers.

The Xlib routines allow you to perform window operations on any supported *display* in a network. You can think of a display as the connection between a client program and an instance of the X server.

Client programs can run in any CPU in the network and open explicit connections to displays to perform input and output. The X server is said to control the display because it maintains client status information, such as the state of the graphics context, and manages input and output for the connection. Client programs can open connections to one or more displays at one time; X servers can maintain multiple client connections at one time.

For example, an instructor who wants to draw a box on all displays in a class could write a program that opened a connection to each display. This program could use its event mask to accept input from one or more of the connections. For more information about creating an event mask, see Chapter 4. If the students in the class write programs that open connections to their own displays, their instance of the server prioritizes the client requests and schedules them accordingly.

Xlib routines provide functions that allow you to specify the display on which you want to perform window operations. Xlib also provides C language macros that return information about a specific display. This section describes routines that enable you to perform the following operations:

- Open (connect to) a display
- Obtain information about a display
- Close (disconnect) a display

For concepts related to display routines and information on how to use display routines, see *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 2-1.

**Table 2-1 Display Routines**

Routine Name	Description
ALL PLANES	Returns a value with all bits set on. This value can then be used in a plane argument to a procedure.

(continued on next page)

# Display Routines

**Table 2–1 (Cont.) Display Routines**

<b>Routine Name</b>	<b>Description</b>
BITMAP BIT ORDER	Returns the value of the leftmost bit in a bitmap.
BITMAP PAD	Returns a number of bits by which all scan lines must be padded.
BITMAP UNIT	Returns the size of a bitmap's unit.
BLACK PIXEL	Returns the color index (pixel value) that yields black on the specified screen.
BLACK PIXEL OF SCREEN	Returns the black pixel of the specified screen.
CELLS OF SCREEN	Returns the number of color map cells on the specified screen.
CLOSE DISPLAY	Closes a connection between a client program and the display that you specify.
CONNECTION NUMBER	Returns the connection number of the display.
DEFAULT COLORMAP	Returns the default color map for allocation on the specified screen.
DEFAULT COLORMAP OF SCREEN	Returns the default color map of the specified screen.
DEFAULT DEPTH	Returns the depth of the default root window for the specified screen.
DEFAULT DEPTH OF SCREEN	Returns the default depth of the specified screen.
DEFAULT GC	Returns the default graphics context for the root window of the specified screen.
DEFAULT GC OF SCREEN	Returns the default graphics context of the specified screen.
DEFAULT ROOT WINDOW	Returns the default root window for a specified display.
DEFAULT SCREEN	Returns the default screen referenced in OPEN DISPLAY.
DEFAULT SCREEN OF DISPLAY	Returns the default screen of the specified display.
DEFAULT VISUAL	Returns the default visual type for the specified screen.
DEFAULT VISUAL OF SCREEN	Returns the default visual of the specified screen.
DISPLAY CELLS	Returns the number of color map cells on the specified screen.
DISPLAY HEIGHT	Returns a number that specifies the height of the screen in pixels.
DISPLAY HEIGHT MM	Returns a number that specifies the height of the screen in millimeters.

(continued on next page)

**Table 2–1 (Cont.) Display Routines**

<b>Routine Name</b>	<b>Description</b>
DISPLAY NAME	Returns the name of the display that you were trying to open.
DISPLAY PLANES	Returns the number of planes on the specified screen.
DISPLAY STRING	Returns the name of the string that was passed to OPEN DISPLAY when the current display was opened.
DISPLAY WIDTH	Returns a number that specifies the width of the screen in pixels.
DISPLAY WIDTH MM	Returns a number that specifies the width of the screen in millimeters.
DOES BACKING STORE	Returns a value that indicates whether the screen supports backing stores.
DOES SAVE UNDERS	Returns a Boolean value that indicates whether the screen supports save unders.
DISPLAY OF SCREEN	Returns the display of the specified screen.
EVENT MASK OF SCREEN	Returns the initial root event mask of the specified screen.
HEIGHT MM OF SCREEN	Returns the height, in millimeters, of the specified screen.
HEIGHT OF SCREEN	Returns the height, in pixels, of the specified screen.
IMAGE BYTE ORDER	The required byte order for images for each scan line unit in XYFormat (bitmap) or for each pixel value in ZFormat.
LAST KNOWN REQUEST PROCESSED	Extracts the serial number of the last request known by Xlib to have been processed to the X server.
MAX CMAPS OF SCREEN	Returns the maximum number of color maps supported by the specified screen.
MIN CMAPS OF SCREEN	Returns the minimum number of color maps supported by the specified screen.
NEXT REQUEST	Extracts the serial number that is to be used for the next request.
OPEN DISPLAY	Opens a connection between a client program and the display that you specify.
PLANES OF SCREEN	Returns the number of planes in the specified screen.
PROTOCOL REVISION	Returns the minor protocol revision number that the X server is using.
PROTOCOL VERSION	Returns the version number of the X protocol associated with the connected display.

(continued on next page)



# Display Routines

**Table 2–1 (Cont.) Display Routines**

<b>Routine Name</b>	<b>Description</b>
Q LENGTH	Returns the length of the input queue for the connected hardware display.
ROOT WINDOW	Returns the identifier of the root window.
ROOT WINDOW OF SCREEN	Returns the root window of the specified screen.
ROTATE BUFFERS	Rotates the ring of cut buffers from 0 to 7.
SCREEN COUNT	Returns the number of available screens.
SCREEN OF DISPLAY	Returns a pointer to the screen of the specified display.
SERVER VENDOR	Returns a pointer to a string that identifies the owner of the X server implementation.
STORE BUFFER	Stores data in a specified cut buffer.
STORE BYTES	Stores data in cut buffer zero.
VENDOR RELEASE	Returns the number of the release of the X server, as assigned by the vendor.
WHITE PIXEL	Returns the color index (pixel value) that yields white on the specified screen.
WHITE PIXEL OF SCREEN	Returns the white pixel of the specified screen.
WIDTH MM OF SCREEN	Returns the width, in millimeters, of the specified screen.
WIDTH OF SCREEN	Returns the width, in pixels, of the specified screen.

## 2.1

### Display Routines

The following pages describe the Xlib display routines.

---

## ALL PLANES

Returns a value with all bits set on. This value can then be used in a plane argument to a procedure.

---

**VAX FORMAT**     *value\_return = X\$ALL\_PLANES*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
value_return	uns longword	uns longword	write	value

---



---

**MIT C FORMAT**     *value\_return = XAllPlanes*

**argument  
information**

unsigned long XAllPlanes()

---

**RETURNS**     *value\_return*  
The returned value with all bits set to 1.

---

**DESCRIPTION**     ALL PLANES returns a value with all bits set on. This value can then be used in a plane argument to a procedure.

## Display Routines

### BITMAP BIT ORDER

---

## BITMAP BIT ORDER

Returns whether the bits in a bitmap are ordered with the least significant bit first or the most significant bit first.

---

**VAX FORMAT**     *order\_return* = **X\$BITMAP\_BIT\_ORDER** (*display*)

---

**argument information**

Argument	Usage	Data Type	Access	Mechanism
order_return	longword	longword	write	value
display	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *order\_return* = **XBitmapBitOrder** (*display*)

---

**argument information**

```
int XBitmapBitOrder(display)
    Display *display;
```

---

### RETURNS

***order\_return***

The value of the leftmost bit in the bitmap, as displayed on the screen. Valid values are shown in the following table:

VAX	C	Description
X\$C_LSB_FIRST	LSBFirst	The least significant bit is displayed first.
X\$C_MSB_FIRST	MSBFirst	The most significant bit is displayed first.

---

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

---

### DESCRIPTION

BITMAP BIT ORDER returns whether the bits in a bitmap are ordered with the least significant bit first or the most significant bit first. Within each bitmap unit, the leftmost bit in the bitmap (as displayed on the screen) is either the least significant bit or the most significant bit in the bitmap unit.

---

## BITMAP PAD

Returns a number of bits by which all scan lines must be padded.

---

**VAX FORMAT**     *pad\_return = X\$BITMAP\_PAD (display)*

---

**argument  
 information**

Argument	Usage	Data Type	Access	Mechanism
pad_return	longword	longword	write	value
display	identifier	uns longword	read	reference

---



---

**MIT C FORMAT**     *pad\_return = XBitmapPad (display)*

---

**argument  
 information**

```
int XBitmapPad(display)
    Display *display;
```

---

**RETURNS**     *pad\_return*  
 The number of bits by which all scan lines must be padded.

---

**ARGUMENTS**     *display*  
 The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**     BITMAP PAD returns a multiple of bits by which all scan lines must be padded. When an image is sent to the server, the number of bits in each scan line must be a multiple of **pad\_return**. If the number of bits in the scan line is greater than the number of bits in the image, the server ignores the extra bits.

# Display Routines

## BITMAP UNIT

---

## BITMAP UNIT

Returns the size of a bitmap's unit.

---

**VAX FORMAT**     *size\_return = X\$BITMAP\_UNIT (display)*

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
size_return	uns longword	uns longword	write	value
display	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *size\_return = XBitmapUnit (display)*

---

**argument  
information**

```
int XBitmapUnit (display)
    Display *display;
```

---

**RETURNS**     *size\_return*  
The size, in bits, of the bitmap's unit.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**     BITMAP UNIT returns the size, in bits, of a bitmap's unit. This value can equal 8, 16, or 32. When an image is sent to the server, the server receives the image in units of the number of bits specified in BITMAP UNIT.

---

## BLACK PIXEL

Returns the color index (pixel value) that yields black on the specified screen.

---

**VAX FORMAT**     *color\_index\_return = X\$BLACK\_PIXEL*  
                          (*display, screen\_number*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
color_index_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference

---



---

**MIT C FORMAT**     *color\_index\_return = XBlackPixel*  
                          (*display, screen\_number*)

**argument  
information**

```
unsigned long XBlackPixel(display, screen_number)
Display *display;
int screen_number;
```

---

**RETURNS**            *color\_index\_return*  
The identifier of the color index (pixel value) that yields black on the specified screen.

---

**ARGUMENTS**        *display*  
The display information originally returned by OPEN DISPLAY.

*screen\_number*  
The number of the screen for which the black pixel is requested.

---

**DESCRIPTION**     BLACK PIXEL returns the color index (pixel value) that yields black on the specified screen.

To obtain the color index (pixel value) that yields white on a specified screen, use WHITE PIXEL.



---

## CELLS OF SCREEN

Returns the number of color map cells on the specified screen.

---

**VAX FORMAT**     *cells\_return* = **X\$CELLS\_OF\_SCREEN**  
                          (*screen\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
<i>cells_return</i>	longword	longword	longword	write
<i>screen_id</i>	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *cells\_return* = **XCellsOfScreen**  
                          (*screen\_id*)

**argument  
information**

```
int XCellsOfScreen(screen_id)
    Screen screen_id;
```

---

**RETURNS**            *cells\_return*  
The number of color map cells on the specified screen.

---

**ARGUMENTS**        *screen\_id*  
The identifier of the screen for which the number of color map cells is requested.

---

**DESCRIPTION**      CELLS OF SCREEN returns the number of color map cells on the specified screen.



## Display Routines

### CLOSE DISPLAY

---

## CLOSE DISPLAY

Closes a connection between a client program and the display that you specify.

---

### VAX FORMAT `X$CLOSE_DISPLAY` (*display*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

---

---

### MIT C FORMAT `XCloseDisplay` (*display*)

#### argument information

```
XCloseDisplay(display)
Display *display;
```

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.

---

### DESCRIPTION

CLOSE DISPLAY disconnects a client's connection to the display specified in the display identifier. OPEN DISPLAY returned a pointer to the display identifier when it opened the display.

CLOSE DISPLAY destroys all windows created by the client, resource identifiers (Window, Font, Pixmap, Colormap, Cursor, and GContext), or other resources (such as graphics contexts) that the client application has created on this display, unless the closedown mode of the resource has been changed. Therefore, these windows, resource identifiers, and other resources should never be referenced again. In addition, CLOSE DISPLAY discards any output events that have been buffered but have not yet been sent. Because these operations automatically (implicitly) occur if a process exits, you normally do not need to call CLOSE DISPLAY explicitly.

The server performs a number of automatic operations when its connection to the display hardware closes. These operations are described in *VMS DECwindows Xlib Programming Volume*.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

---

## Display Routines

### CONNECTION NUMBER

---

## CONNECTION NUMBER

Returns an integer identifying the connection.

---

**VAX FORMAT**    *int = X\$CONNECTION\_NUMBER (display)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
int	longword	longword	write	value
display	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    *int = XConnectionNumber (display)*

**argument  
information**

```
int ConnectionNumber(display)
    Display *display;
```

---

**RETURNS**        *int*  
The connection number of the specified display.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**    CONNECTION NUMBER returns an integer that identifies the connection. This routine is defined to be operating system specific by MIT, and there is no direct analogue to the Unix file descriptor in VMS.

## DEFAULT COLORMAP

Returns the default color map for allocation on the specified screen.

**VAX FORMAT**     *colormap\_id\_return = X\$DEFAULT\_COLORMAP  
(display, screen\_number)*

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
colormap_id_return	identifier	longword	write	value
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference

**MIT C FORMAT**     *colormap\_id\_return = XDefaultColormap  
(display, screen\_number)*

**argument  
information**

```
Colormap XDefaultColormap(display, screen_number)
Display *display;
int screen_number;
```

**RETURNS**     *colormap\_id\_return*  
The identifier of the default color map for the specified screen.

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

*screen\_number*  
The number of the screen for which the default color map is being requested.

**DESCRIPTION**     DEFAULT COLORMAP returns the identifier of the default color map for the specified screen. Most applications should allocate pixel values out of this color map.

## Display Routines

### DEFAULT COLORMAP OF SCREEN

---

## DEFAULT COLORMAP OF SCREEN

Returns the default color map of the specified screen.

---

**VAX FORMAT**     *colormap\_id\_return* =  
**X\$DEFAULT\_COLORMAP\_OF\_SCREEN**  
                  (*screen\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
colormap_id_return	identifier	uns longword	write	value
screen_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *colormap\_id\_return* = **XDefaultColormapOfScreen**  
                  (*screen\_id*)

**argument  
information**

---

```
Colormap XDefaultColormapOfScreen(screen_id)
Screen *screen_id;
```

---

**RETURNS**            ***colormap\_id\_return***  
The identifier of the default color map of the specified screen.

---

**ARGUMENTS**        ***screen\_id***  
The identifier of the screen for which the colormap is requested.

---

**DESCRIPTION**     **DEFAULT COLORMAP OF SCREEN** returns the default color map of the specified screen.









# Display Routines

## DEFAULT GC OF SCREEN

---

### DEFAULT GC OF SCREEN

Returns the default graphics context of the specified screen.

---

**VAX FORMAT**     *gc\_return* =X\$DEFAULT\_GC\_OF\_SCREEN  
                  (*screen\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
<i>gc_return</i>	identifier	uns longword	write	value
<i>screen_id</i>	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *gc\_return* =XDefaultGCOfScreen  
                  (*screen\_id*)

**argument  
information**

```
GC XDefaultGCOfScreen(screen_id)
Screen *screen_id;
```

---

**RETURNS**             *gc\_return*  
The identifier of the default graphics context of the specified screen.

---

**ARGUMENTS**          *screen\_id*  
The identifier of the screen for which the default graphics context is requested.

---

**DESCRIPTION**        DEFAULT GC OF SCREEN returns the default graphics context of the specified screen.





---

## DEFAULT SCREEN OF DISPLAY

Returns the default screen of the specified display.

---

**VAX FORMAT**     *screen\_id\_return* =  
**X\$DEFAULT\_SCREEN\_OF\_DISPLAY**  
                  (*display*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
screen_id_return	record	x\$screen	write	value
display	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *screen\_id\_return* = **XDefaultScreenOfDisplay**  
                  (*display*)

**argument  
information**

Screen \*XDefaultScreenOfDisplay(*display*)  
Display \**display*;

---

**RETURNS**             *screen\_id\_return*  
The identifier of the default screen for the specified display.

---

**ARGUMENTS**         *display*  
The display information originally returned by the OPEN DISPLAY routine.

---

**DESCRIPTION**        **DEFAULT\_SCREEN\_OF\_DISPLAY** returns the default screen of the specified display.

## Display Routines

### DEFAULT VISUAL

---

## DEFAULT VISUAL

Returns a visual structure for the specified screen.

---

**VAX FORMAT**     *status\_return = X\$DEFAULT\_VISUAL*  
                          (*display, screen\_number, visual\_return*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference
visual_return	record	x\$visual	write	reference

---

---

**MIT C FORMAT**     *visual\_return = XDefaultVisual*  
                          (*display, screen\_number*)

**argument  
information**

```
Visual *XDefaultVisual(display, screen_number)
Display *display;
int screen_number;
```

---

**RETURNS**     *status\_return (VAX only)*  
                  Specifies whether the routine completed successfully.

*visual\_return (MIT C only)*  
                  The returned visual structure.

---

**ARGUMENTS**     *display*  
                  The display information originally returned by OPEN DISPLAY.

*screen\_number*  
                  The number of the screen for which you are requesting the visual structure.

*visual\_return (VAX only)*  
                  The returned visual structure.

---

**DESCRIPTION**     DEFAULT VISUAL returns the default visual structure for the specified screen.

---

## DEFAULT VISUAL OF SCREEN

Returns the default visual of the specified screen.

---

**VAX FORMAT**    **X\$DEFAULT\_VISUAL\_OF\_SCREEN**  
*(screen\_id, visual\_return)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
screen_id	identifier	uns longword	read	reference
visual_return	record	x\$visual	write	reference

---

---

**MIT C FORMAT**    *visual\_return = XDefaultVisualOfScreen*  
*(screen\_id)*

**argument  
information**

```
Visual *XDefaultVisualOfScreen(screen_id)  
Screen *screen_id;
```

---

**RETURNS**    ***visual\_return (MIT C only)***  
The default visual structure for the specified screen.

---

**ARGUMENTS**    ***screen\_id***  
The identifier of the screen for which the default visual is requested.

***visual\_return (VAX only)***  
The returned visual structure.

---

**DESCRIPTION**    **DEFAULT VISUAL OF SCREEN** returns the default visual structure of the specified screen.

# Display Routines

## DISPLAY CELLS

---

## DISPLAY CELLS

Returns the number of color map cells on the specified screen.

---

**VAX FORMAT**     *cells\_return = X\$DISPLAY\_CELLS*  
                          (*display, screen\_number*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
cells_return	longword	longword	write	value
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**     *cells\_return = XDisplayCells*  
                          (*display, screen\_number*)

**argument  
information**

```
int XDisplayCells(display, screen_number)
    Display *display;
    int screen_number;
```

---

**RETURNS**            *cells\_return*  
The number of color map cells on the specified screen.

---

**ARGUMENTS**        *display*  
The display information originally returned by OPEN DISPLAY.

*screen\_number*  
The number of the screen for which the number of cells is to be determined.

---

**DESCRIPTION**      DISPLAY CELLS returns the number of color map cells present on the specified screen.







---

## DISPLAY NAME

Returns the name of the display that you were trying to open.

---

**VAX FORMAT**     *status = X\$DISPLAY\_NAME*  
                           (*disp\_string, disp\_name\_return [,disp\_len\_return]*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status	cond_value	uns longword	write	value
disp_string	char string	char string	read	descriptor
disp_name_return	char string	char string	write	descriptor
disp_len_return	longword	uns longword	write	reference

---



---

**MIT C FORMAT**     *char = XDisplayName*  
                           (*disp\_string*)

**argument  
information**

```
char *XDisplayName(disp_string)
char *string;
```

---

**RETURNS**     *status (VAX only)*  
 Specifies whether or not the routine completed successfully.

*char (MIT C only)*  
 The name of the display that you are currently using is returned to this argument.

---

**ARGUMENTS**     *disp\_string*  
 The name of the display that you are trying to open. If you specify a null string, DISPLAY NAME looks in the environment for the display and returns the name of the display that OPEN DISPLAY would attempt to use.

**VAX only**

The **string** argument is the address of a character string descriptor that points to the string.

**MIT C only**

The **string** argument is a pointer to the null-terminated character string.

## Display Routines

### DISPLAY NAME

#### *disp\_name\_return (VAX only)*

The address of a character string descriptor that points to the string. DISPLAY NAME returns the display name to this argument.

#### *disp\_len\_return (VAX only)*

DISPLAY NAME returns the length of the display name string to this optional argument.

---

### DESCRIPTION

DISPLAY NAME returns the name of the display that you were trying to open. Errors are usually reported relative to a display. If your client program fails in an attempt to open a connection to a display, you can use DISPLAY NAME to find out the name of the display.

---

## DISPLAY OF SCREEN

Returns the display of the specified screen.

---

**VAX FORMAT**    **X\$DISPLAY\_OF\_SCREEN**  
                  (*screen\_id*, *display\_return*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
screen_id	identifier	uns longword	read	reference
display_return	record	x\$display	write	value

---

---

**MIT C FORMAT**    *display\_return* = **XDisplayOfScreen**  
                                  (*screen\_id*)

**argument  
information**

```
Display *XDisplayOfScreen(screen_id)
Screen *screen_id;
```

---

**RETURNS**            ***display\_return (MIT C only)***  
                          The display of the specified screen.

---

**ARGUMENTS**        ***screen\_id***  
                          The identifier of the screen for which the value is requested.

***display\_return (VAX only)***  
                          The display of the specified screen.

---

**DESCRIPTION**      DISPLAY OF SCREEN returns the display of the specified screen.

## Display Routines

### DISPLAY PLANES

---

## DISPLAY PLANES

Returns the number of planes on the specified screen.

---

**VAX FORMAT**     *planes\_return* = **X\$DISPLAY\_PLANES**  
                          (*display*, *screen\_number*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
<i>planes_return</i>	longword	longword	write	value
<i>display</i>	identifier	uns longword	read	reference
<i>screen_number</i>	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**     *planes\_return* = **XDisplayPlanes**  
                          (*display*, *screen\_number*)

**argument  
information**

```
int XDisplayPlanes(display, screen_number)
    Display *display;
    int screen_number;
```

---

**RETURNS**             *planes\_return*  
                          The number of planes on the specified screen.

---

**ARGUMENTS**         *display*  
                          The display information originally returned by OPEN DISPLAY.

*screen\_number*  
                          The number of the screen for which you want to obtain the number of planes.

---

**DESCRIPTION**        DISPLAY PLANES returns the number of planes on the specified screen.



## Display Routines

### DISPLAY STRING

---

**DESCRIPTION**    `DISPLAY STRING` returns the name of the string that was passed to `OPEN DISPLAY` when the current hardware display was opened. If the string was null, `DISPLAY STRING` returns the value of the logical name when the current display was opened.

`DISPLAY STRING` is useful to applications that want to open a new connection from the child process to the same hardware display.









## Display Routines

### DOES SAVE UNDERS

---

## DOES SAVE UNDERS

Returns a Boolean value that indicates whether the screen supports save unders.

---

**VAX FORMAT**    *Bool = X\$DOES\_SAVE\_UNDERS (screen\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
Bool	Boolean	uns longword	write	value
screen_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    *Bool = XDoesSaveUnders (screen\_id)*

**argument  
information**

```
ret XDoesSaveUnders(screen_id)
    Screen *screen_id;
```

---

**RETURNS**

***Bool***

A Boolean value that, when true, indicates that the screen supports save unders. If this value is false, the screen does not support save unders.

---

**ARGUMENTS**

***screen\_id***

The identifier of the screen for which the value is requested.

---

**DESCRIPTION**

DOES SAVE UNDERS indicates whether the screen supports save unders.



# Display Routines

## FREE

---

### FREE

Frees a data buffer that was created by an Xlib routine.

---

**VAX FORMAT**    **X\$FREE** (*buff\_ptr* [,*buff\_len*])

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
buff_ptr	byte	v. uns byte	read	reference
buff_len	longword	longword	read	reference

---

---

**C FORMAT**    **XFree** (*buff\_ptr*)

**argument  
information**

XFree (*buff\_ptr*)  
char \**buff\_ptr*

---

**ARGUMENTS**    ***buff\_ptr***  
A pointer to the data buffer that is to be freed.

***buff\_len (MIT C Only)***  
The size, in bytes, of the buffer. This argument is optional; if it is not specified, ***buff\_ptr*** is assumed to contain a null-terminated byte stream.

---

**DESCRIPTION**    FREE frees a data buffer that was created by an Xlib routine.









## Display Routines

### LAST KNOWN REQUEST PROCESSED

---

## LAST KNOWN REQUEST PROCESSED

Extracts the serial number of the last request known by Xlib to have been processed to the X server.

---

**VAX FORMAT**     *request\_return =*  
**X\$LAST\_KNOWN\_REQUEST\_PROCESSED**  
                  (*display*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
request_return	uns longword	uns longword	write	value
display	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *request\_return =* **XLastKnownRequestProcessed**  
                  (*display*)

**argument  
information**

---

```
int XLastKnownRequestProcessed(display)
    Display *display;
```

---

**RETURNS**            *request\_return*  
The serial number of the last request known by Xlib to have been processed to the X server.

---

**ARGUMENTS**        *display*  
The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**      LAST KNOWN REQUEST PROCESSED extracts the serial number of the last request known by Xlib to have been processed to the X server. This number is automatically set by Xlib when replies, events, and errors are received.





---

## NEXT REQUEST

Extracts the serial number that is to be used for the next request.

---

**VAX FORMAT**     *request\_return = X\$NEXT\_REQUEST (display)*

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
request_return	uns longword	uns longword	write	value
display	identifier	uns longword	read	reference

---



---

**MIT C FORMAT**     *request\_return = XNextRequest (display)*

---

**argument  
information**

```
int XNextRequest(display)
    Display *display;
```

---

**RETURNS**     *request\_return*  
The serial number that Xlib is to use for the next request.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**     NEXT REQUEST extracts the serial number that is to be used for the next request. Serial numbers are maintained separately for each display connection.

# Display Routines

## NO OP

---

## NO OP

Sends a no-operation protocol request to the server.

---

### VAX FORMAT **X\$NO\_OP** (*display*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

---

---

### MIT C FORMAT **XNoOp** (*display*)

#### argument information

XNoOp(*display*)  
Display \**display*;

---

### ARGUMENTS ***display***

The *display* information originally returned by OPEN DISPLAY.

---

### DESCRIPTION

NO OP generates a no-operation server protocol request. It does not explicitly flush the output buffer.



## Display Routines

### OPEN DISPLAY

screen            The number of the screen on that server. An X server can control multiple screens on one display. The screen sets an internal variable that can be accessed by using the DefaultScreen macro or the DEFAULT\_SCREEN function.

#### VAX only

The **display\_name** argument is the address of the character string descriptor that points to the string.

#### C only

The **display\_name** argument is a pointer to a null-terminated character string.

---

## DESCRIPTION

OPEN DISPLAY connects a client program to the specified display. You pass to OPEN DISPLAY the name of the display to which you want to connect. If successful, OPEN DISPLAY establishes the connection to the display and returns a pointer to a display identifier. If OPEN DISPLAY does not succeed, it returns a null value in the MIT C binding and a zero in the VAX binding.

After you successfully connect the client to the display, you can obtain data from the display using other Xlib routines. However, OPEN DISPLAY should be called before calling any other Xlib routines.









## Display Routines

### Q LENGTH

---

## Q LENGTH

Returns the length of the input event queue for the connected hardware display.

---

**VAX FORMAT**     *length\_return = X\$Q\_LENGTH (display)*

---

#### argument information

Argument	Usage	Data Type	Access	Mechanism
length_return	longword	longword	write	value
display	identifier	uns longword	read	reference

---

**MIT C FORMAT**     *length\_return = XQLength (display)*

---

#### argument information

```
int XQLength(display)
    Display *display;
```

---

**RETURNS**     *length\_return*  
The length of the input event queue for the connected hardware display.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**     Q LENGTH returns the length of the input event queue for the connected hardware display. See also the EVENTS QUEUED routine.

---

---

## ROOT WINDOW

Returns the identifier of the root window.

---

**VAX FORMAT**     *root\_window\_id\_return = X\$ROOT\_WINDOW*  
                          (*display, screen\_number*)

argument  
information

---

Argument	Usage	Data Type	Access	Mechanism
root_window_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference

---



---

**MIT C FORMAT**     *root\_window\_id\_return = XRootWindow*  
                          (*display, screen\_number*)

argument  
information

```
Window XRootWindow(display, screen_number)
Display *display;
int screen_number;
```

---

**RETURNS**            *root\_window\_id\_return*  
The identifier of the root window.

---

**ARGUMENTS**        *display*  
The display information originally returned by OPEN DISPLAY.

*screen\_number*  
The number of the screen for which the root window is to be obtained.

---

**DESCRIPTION**      ROOT WINDOW obtains the identifier of the root window. ROOT WINDOW is useful with other Xlib routines that take a parent window as an argument.



---

## ROTATE BUFFERS

Rotates the ring of cut buffers from 0 to 7.

---

### VAX FORMAT **X\$ROTATE\_BUFFERS** (*display, rotate*)

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
rotate	longword	longword	read	reference

---

### MIT C FORMAT **XRotateBuffers** (*display, rotate*)

#### argument information

```
XRotateBuffers(display, rotate)
Display *display;
int rotate;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***rotate***

Specifies how much to rotate the cut buffers.

---

### DESCRIPTION

ROTATE BUFFERS rotates the cut buffers based on the value that you specify. Buffer 0 becomes buffer  $n$ , buffer 1 becomes buffer  $n+1 \text{ MOD } 8$ , and so on. This method of numbering cut buffers is global to the display. Note that ROTATE BUFFERS generates an error if any of the eight buffers have not been created.

## Display Routines

### ROTATE BUFFERS

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

---

## SCREEN COUNT

Returns the number of screens in the specified display.

---

**VAX FORMAT**     *count\_return = X\$SCREEN\_COUNT (display)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
count_return	longword	longword	write	value
display	identifier	uns longword	read	reference

---



---

**MIT C FORMAT**     *count\_return = XScreenCount (display)*

**argument  
information**

```
int XScreenCount (display)
    Display *display;
```

---

**RETURNS**     *request\_return*  
The number of screens in the specified display.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**     SCREEN COUNT returns the number of screens in the specified display.



## Display Routines

### SCREEN OF DISPLAY

---

## SCREEN OF DISPLAY

Returns the identifier of the screen of the specified display.

---

**VAX FORMAT**     **X\$SCREEN\_OF\_DISPLAY**  
*(display, screen\_number, screen\_return)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
screen_number	uns longword	uns longword	read	reference
screen_return	record	x\$screen	write	reference

---

---

**MIT C FORMAT**     *screen\_id\_return =XScreenOfDisplay*  
*(display, screen\_number)*

**argument  
information**

```
Screen *XScreenOfDisplay(display, screen_number)
Display *display;
int screen_number;
```

---

**RETURNS**     ***screen\_id\_return (MIT C only)***  
The identifier of the screen associated with the specified display.

---

**ARGUMENTS**     ***display***  
The display information originally returned by the OPEN DISPLAY routine.

***screen\_number***  
The number of the returned screen.

***screen\_return (VAX only)***  
The identifier of the screen associated with the specified display.

---

**DESCRIPTION**     SCREEN OF DISPLAY returns the identifier of the screen associated with the specified display.

---

## SERVER VENDOR

Returns a pointer to a string that identifies the owner of the X server implementation.

---

**VAX FORMAT**    **X\$SERVER\_VENDOR**  
(*display, vendor\_name\_return, [len\_return]*)

**argument information**

Argument	Usage	Data Type	Access	Mechanism
string_name	char string	char string	read	descriptor
display	identifier	uns longword	read	reference
vendor_name_return	char string	char_string	write	descriptor
len_return	word	uns word	write	reference

---

**MIT C FORMAT**    *vendor\_name\_return = XServerVendor*  
(*display*)

**argument information**

```
char *XServerVendor(display)
    Display *display;
```

---

**RETURNS**

***vendor\_name\_return (MIT C only)***

The name of the string that identifies the owner of the X server implementation.

The **vendor\_name\_return** argument is a pointer to the null-terminated character string.

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***vendor\_name\_return (VAX only)***

The name of the string that identifies the owner of the X server implementation.

The **vendor\_name\_return** argument is the address of a character string descriptor that points to the string.

***len\_return (VAX only)***

The length of the returned string. This argument is optional.

## Display Routines

### SERVER VENDOR

---

**DESCRIPTION**    `SERVER VENDOR` returns a pointer to a character string that identifies the owner of the X server implementation.

---

## STORE BUFFER

Stores data in a specified cut buffer.

---

**VAX FORMAT**    **X\$STORE\_BUFFER**  
(*display, bytes, num\_bytes, buffer*)

argument  
information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
bytes	array	byte	read	reference
num_bytes	longword	longword	read	reference
buffer	longword	longword	read	reference

---

**MIT C FORMAT**    **XStoreBuffer**  
(*display, bytes, num\_bytes, buffer*)

argument  
information

---

```
XStoreBuffer(display, bytes, num_bytes, buffer)
Display *display;
char bytes[];
int num_bytes;
int buffer;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***bytes***

A pointer to the array of bytes that you want to store. The length of the array is specified by **num\_bytes**. The byte string is not necessarily ASCII or null terminated.

#### ***num\_bytes***

The number of bytes that you want to store in the **bytes** argument.

#### ***buffer***

The buffer in which you want to store the byte string. Valid entries are 0 through 7.

---

### DESCRIPTION

STORE BUFFER stores the string of bytes in the buffer that you specify. Clients can retrieve the contents of the cut buffer by calling FETCH BUFFER.

## Display Routines

### STORE BUFFER

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## STORE BYTES

Stores data in cut buffer zero.

---

**VAX FORMAT**    **X\$STORE\_BYTES**  
(*display, bytes, num\_bytes*)

argument  
information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
bytes	array	byte	read	reference
num_bytes	longword	longword	write	reference

---

**MIT C FORMAT**    **XStoreBytes**  
(*display, bytes, num\_bytes*)

argument  
information

```
XStoreBytes(display, bytes, num_bytes)
Display *display;
char bytes[];
int num_bytes;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***bytes***  
A pointer to the array of bytes that you want to store. The byte string is not necessarily ASCII text or a null-terminated string. The length of the array is specified by **num\_bytes**.

***num\_bytes***  
The number of bytes in the string that you want to store.

---

**DESCRIPTION**    STORE BYTES returns the number of bytes to be stored to the **num\_bytes** argument.

Clients can retrieve the contents of the cut buffer by calling FETCH BYTES.

## Display Routines

### STORE BYTES

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.









## Display Routines

### WIDTH MM OF SCREEN

---

## WIDTH MM OF SCREEN

Returns the width, in millimeters, of the specified screen.

---

**VAX FORMAT**     *width\_return = X\$WIDTH\_MM\_OF\_SCREEN*  
                          (*screen\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
width_return	longword	longword	write	value
screen_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *width\_return = XWidthMMOfScreen*  
                          (*screen\_id*)

**argument  
information**

---

```
width XWidthMMOfScreen(screen_id)
Screen *screen_id;
```

---

**RETURNS**            *width\_return*  
The width, in millimeters, of the specified screen.

---

**ARGUMENTS**        *screen\_id*  
The identifier of the screen for which the width is requested.

---

**DESCRIPTION**      WIDTH MM OF SCREEN returns the width, in millimeters, of the specified screen.

When you want to obtain the width of a screen in pixels, use WIDTH OF SCREEN.

---

## WIDTH OF SCREEN

Returns the width, in pixels, of the specified screen.

---

**VAX FORMAT**     *width\_return = X\$WIDTH\_OF\_SCREEN*  
                          (*screen\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
<i>width_return</i>	longword	longword	write	value
<i>screen_id</i>	identifier	uns longword	read	reference

---



---

**MIT C FORMAT**     *width\_return = XWidthOfScreen*  
                          (*screen\_id*)

**argument  
information**

```
long XWidthOfScreen (screen_id)
Screen *screen_id;
```

---

**RETURNS**            ***width\_return***  
The width, in pixels, of the specified screen.

---

**ARGUMENTS**        ***screen\_id***  
The identifier of the screen for which the width is requested.

---

**DESCRIPTION**     WIDTH OF SCREEN returns the width, in pixels, of the specified screen.  
When you want to obtain the width of a screen in millimeters, use WIDTH  
MM OF SCREEN.



# 3

## Window Routines

---

The Xlib window routines enable you to perform the following operations:

- Create windows
- Destroy windows
- Map windows
- Unmap windows
- Configure windows
- Raise and lower windows
- Change window attributes
- Provide window information
- Translate window coordinates

For concepts related to window routines and information on how to use window routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 3–1.

**Table 3–1 Window Routines**

<b>Routine Name</b>	<b>Description</b>
CHANGE WINDOW ATTRIBUTES	Sets the attributes of a specified window.
CIRCULATE SUBWINDOWS	Circulates the appropriate child of a window in a specified direction.
CIRCULATE SUBWINDOWS DOWN	Lowers the highest mapped child window of a window that occludes another window.
CIRCULATE SUBWINDOWS UP	Raises the lowest mapped child of an occluded window.
CONFIGURE WINDOW	Reconfigures a window's size, position, border, and stacking order.
CREATE SIMPLE WINDOW	Creates a subwindow of a specified parent window from an application.
CREATE WINDOW	Creates a subwindow of a parent window and defines the attributes of the subwindow.
DESTROY SUBWINDOWS	Destroys all subwindows of a window.
DESTROY WINDOW	Destroys a window and all of its subwindows from an application.
GET GEOMETRY	Obtains the current geometry of the specified drawable.

(continued on next page)

# Window Routines

**Table 3–1 (Cont.) Window Routines**

<b>Routine Name</b>	<b>Description</b>
GET WINDOW ATTRIBUTES	Obtains the attributes of a specified window.
LOWER WINDOW	Lowers a window so that it does not obscure any sibling windows.
MAP RAISED	Maps the specified window and raises it to the top of the stack.
MAP SUBWINDOWS	Maps all subwindows of a specified window.
MAP WINDOW	Maps a window and all of its subwindows that have had map requests.
MOVE RESIZE WINDOW	Changes the location and the size of a specified window.
MOVE WINDOW	Moves and raises a window without changing its size.
QUERY POINTER	Obtains the root window where the pointer is currently located and the pointer coordinates relative to the root window's origin.
QUERY TREE	Lists the parent, the children, and the number of children for a window.
RAISE WINDOW	Raises a window so that no sibling window obscures it.
RESIZE WINDOW	Changes the size of a window.
RESTACK WINDOWS	Restacks an array of windows from top to bottom.
SET WINDOW BACKGROUND	Sets the background of a specified window to the specified pixel.
SET WINDOW BACKGROUND PIXMAP	Changes the background tile of a specified window.
SET WINDOW BORDER	Sets a window's border to a specified pixel.
SET WINDOW BORDER PIXMAP	Changes and repaints the border tile of a specified window.
SET WINDOW BORDER WIDTH	Changes the border width of a window.
TRANSLATE COORDINATES	Transfers coordinates from the coordinate space of one window to another.
UNMAP SUBWINDOWS	Unmaps all subwindows of a specified window.
UNMAP WINDOW	Unmaps a window.

The routines use the following data structures:

- Set window attributes
- Window changes
- Window attributes

### 3.1 Set Window Attributes Data Structure

The set window attributes data structure specifies and receives information about window attributes. The data structure for the VAX binding is shown in Figure 3–1, and information about members in the data structure is described in Table 3–2.

Figure 3–1 Set Window Attributes Data Structure (VAX Binding)

x\$_swda_background_pixmap	0
x\$_swda_background_pixel	4
x\$_swda_border_pixmap	8
x\$_swda_border_pixel	12
x\$_swda_bit_gravity	16
x\$_swda_win_gravity	20
x\$_swda_backing_store	24
x\$_swda_backing_planes	28
x\$_swda_backing_pixel	32
x\$_swda_save_under	36
x\$_swda_event_mask	40
x\$_swda_do_not_propagate_mask	44
x\$_swda_override_redirect	48
x\$_swda_colormap	52
x\$_swda_cursor	56

Table 3–2 Members of the Set Window Attributes Data Structure (VAX Binding)

Member Name	Contents
X\$_SWDA_BACKGROUND_PIXMAP	Defines the window background of an input-output window. This member can assume one of three possible values: pixmap identifier, the constant x\$_c_none (default), or the constant x\$_c_parent_relative.

(continued on next page)



# Window Routines

## 3.1 Set Window Attributes Data Structure

**Table 3–2 (Cont.) Members of the Set Window Attributes Data Structure (VAX Binding)**

Member Name	Contents
X\$_SWDA_BACKGROUND_PIXEL	Specifying a value for the X\$_SWDA_BACKGROUND_PIXEL member causes the server to override the X\$_SWDA_BACKGROUND_PIXMAP member. This is equivalent to a specifying pixmap of any size filled with the background pixel and used to paint the window background.
X\$_SWDA_BORDER_PIXMAP	Defines the window border of an input-output window. This member can be either X\$_C_COPY_FROM_PARENT or a pixmap identifier.
X\$_SWDA_BORDER_PIXEL	Specifying a value for X\$_SWDA_BORDER_PIXEL causes the server to override the X\$_SWDA_BORDER_PIXMAP member. This is equivalent to specifying a pixmap of any size filled with the border pixel and used to paint the window border.
X\$_SWDA_BIT_GRAVITY	Defines how window contents should be moved when an input-only or input-output window is resized. By default, the server does not retain window contents.
X\$_SWDA_WIN_GRAVITY	Defines how the server should reposition the newly-created input-only or input-output window when its parent window is resized. By default, the server does not move the newly created window.
X\$_SWDA_BACKING_STORE	Provides a hint to the server about how the client wants it to manage obscured portions of the window. In this release clients must maintain window contents.
X\$_SWDA_BACKING_PLANES	Indicates (with bits set to one) which bit planes of the window hold dynamic data that must be preserved if the window obscures or is obscured by another window. In this release clients must maintain data to be preserved.
X\$_SWDA_BACKING_PIXEL	Defines what values to use in planes not specified by the X\$_SWDA_BACKING_PLANES member. In this release clients must maintain values.
X\$_SWDA_SAVE_UNDER	Setting the X\$_SWDA_SAVE_UNDER member to true informs the server that the client would like the contents of the screen saved when an input-output window obscures them. Clients must maintain the contents of screens.
X\$_SWDA_EVENT_MASK	Defines which types of events associated with an input-only or input-output window the server should report to the client. A complete table appears in Chapter 4.
X\$_SWDA_DO_NOT_PROPAGATE_MASK	Defines which kinds of events should not be propagated to ancestors.

(continued on next page)

## 3.1 Set Window Attributes Data Structure

**Table 3–2 (Cont.) Members of the Set Window Attributes Data Structure (VAX Binding)**

Member Name	Contents
X\$_SWDA_OVERRIDE_REDIRECT	Specifies whether calls to map and configure an input-only or input-output window should override a request by another client to redirect those calls. Typically, this is used to inform a window manager not to tamper with the window, for example when the client is creating and mapping a menu.
X\$_SWDA_COLORMAP	Specifies the color map, if any, that best reflects the colors of an input-output window. The color map must have the same visual type as the window. If it does not, the server issues an error.
X\$_SWDA_CURSOR	Specifying a value for the cursor member causes the server to use a particular cursor when the pointer is in an input-only or input-output window.

Table 3–3 lists default values for the SET WINDOW ATTRIBUTES structure.

**Table 3–3 Default Values of the Set Window Attributes Structure**

Member	Default Value
X\$_SWDA_BACKGROUND_PIXMAP	None
X\$_SWDA_BACKGROUND_PIXEL	Undefined
X\$_SWDA_BORDER_PIXMAP	Copied from the parent window
X\$_SWDA_BORDER_PIXEL	Undefined
X\$_SWDA_BIT_GRAVITY	Window contents not retained
X\$_SWDA_WIN_GRAVITY	Window not moved
X\$_SWDA_BACKING_STORE	Window contents not retained
X\$_SWDA_BACKING_PLANES	All 1s
X\$_SWDA_BACKING_PIXEL	0
X\$_SWDA_SAVE_UNDER	False
X\$_SWDA_EVENT_MASK	Empty set
X\$_SWDA_DO_NOT_PROPAGATE_MASK	Empty set
X\$_SWDA_OVERRIDE_REDIRECT	False
X\$_SWDA_COLORMAP	Copied from parent
X\$_SWDA_CURSOR	None

The data structure for the MIT C binding is shown in Figure 3–2, and information about members in the data structure is described in Table 3–4.

# Window Routines

## 3.1 Set Window Attributes Data Structure

**Figure 3–2 Set Window Attributes Data Structure (MIT C Binding)**

---

```
typedef struct {
    Pixmap background_pixmap;
    unsigned long background_pixel;
    Pixmap border_pixmap;
    unsigned long border_pixel;
    int bit_gravity;
    int win_gravity;
    int backing_store;
    unsigned long backing_planes;
    unsigned long backing_pixel;
    Bool save_under;
    long event_mask;
    long do_not_propagate_mask;
    Bool override_redirect;
    Colormap colormap;
    Cursor cursor;
} XSetWindowAttributes
```

---

**Table 3–4 Members of the Set Window Attributes Structure (MIT C Binding)**

---

<b>Member Name</b>	<b>Contents</b>
background_pixmap	Defines the window background. The background_pixmap member can assume one of three possible values: pixmap identifier, the constant None (default), or the constant ParentRelative.
background_pixel	Specifying a value for the background_pixel member causes the server to override the background_pixmap member. This is equivalent to a specifying pixmap of any size filled with the background pixel and used to paint the window background.
border_pixmap	Defines the window border. This can either be CopyFromParent or a Pixmap.
border_pixel	Specifying a value for border_pixel causes the server to override the border_pixmap member. This is equivalent to specifying a pixmap of any size filled with the border pixel and used to paint the window border.
bit_gravity	Defines how the contents of the window should be moved when the window is resized. By default, the server does not retain window contents.
win_gravity	Defines how the server should reposition the newly-created window when its parent window is resized. By default, the server does not move the newly created window.

---

(continued on next page)

**Table 3–4 (Cont.) Members of the Set Window Attributes Structure (MIT C Binding)**

Member Name	Contents
backing_store	<p>Provides a hint to the server about how the client wants it to manage obscured portions of the window. The hint does not guarantee the server will maintain window contents. The following are possible:</p> <ul style="list-style-type: none"> <li>• The constant <code>WhenMapped</code>—Advises the server that the client wants it to maintain the contents of obscured regions when the window is mapped.</li> <li>• The constant <code>Always</code>—Advises the server that the client wants it to maintain the contents of obscured regions even when the window is unmapped. The server might maintain complete contents of a window when part of the window is obscured. If the server maintains the contents, exposure events are not generated, but the server may stop maintaining contents at any time.</li> <li>• The constant <code>NotUseful</code>—Advises the server that it does not need to maintain the contents of the window.</li> </ul>
backing_planes	<p>Indicates (with bits set to one) which bit planes of the window hold dynamic data that must be preserved if the window obscures or is obscured by another window.</p>
backing_pixel	<p>Defines what values to use in planes not specified by the <code>backing_planes</code> member. The server is free to save only specified bit planes and to regenerate the remaining planes with the specified pixel value. Bits that extend beyond the number per pixel of the window are ignored.</p>
save_under	<p>Setting the <code>save_under</code> member to true informs the server that the client would like the contents of the screen saved when the window obscures them. Saving the contents of obscured portions of the screen is not guaranteed.</p>
event_mask	<p>Defines which types of events associated with the window the server should report to the client. See the complete table in Chapter 4.</p>
do_not_propagate_mask	<p>Defines which kinds of events should not be propagated to ancestors.</p>
override_redirect	<p>Specifies whether calls to map and configure the window should override a request by another client to redirect those calls. Typically, this is used to inform a window manager not to tamper with the window, for example when the client is creating and mapping a menu.</p>
colormap	<p>Specifies the color map, if any, that best reflects the colors of the window. The color map must have the same visual type as the window. If it does not, the server issues an error.</p>
cursor	<p>Specifying a value for the <code>cursor</code> member causes the server to use a particular cursor when the pointer is in the window.</p>

## 3.2 Window Changes Data Structure

The window changes data structure specifies and receives information about a window. You use the window changes data structure in the `CONFIGURE WINDOW` routine, when you want to reconfigure a window's size, position, border, and stacking order.

## Window Routines

### 3.2 Window Changes Data Structure

The data structure for the VAX binding is shown in Figure 3–3, and information about members in the data structure is described in Table 3–5.

**Figure 3–3 Window Changes Data Structure (VAX Binding)**

---

x\$l_wchg_x	0
x\$l_wchg_y	4
x\$l_wchg_width	8
x\$l_wchg_height	12
x\$l_wchg_border_width	16
x\$l_wchg_sibling	20
x\$l_wchg_stack_mode	24

---

**Table 3–5 Members of the Window Changes Data Structure (VAX Binding)**

---

Member Name	Contents
X\$L_WCHG_X	Defines the x-coordinate of the new location of the window relative to the origin of its parent. The x and y coordinates specify the upper left outside corner of the window.
X\$L_WCHG_Y	Defines the y-coordinate of the new location of the window relative to the origin of its parent. The x and y coordinates specify the upper left outside corner of the window.
X\$L_WCHG_WIDTH	Defines the new width of the window, excluding the border.
X\$L_WCHG_HEIGHT	Defines the new height of the window, excluding the border.
X\$L_WCHG_BORDER_WIDTH	Specifies the new window border in pixels.
X\$L_WCHG_SIBLING	Specifies the sibling window for stacking order.
X\$L_WCHG_STACK_MODE	Defines how the window is restacked.

---

The data structure for the MIT C binding is shown in Figure 3–4, and information about members in the data structure is described in Table 3–6.

# Window Routines

## 3.2 Window Changes Data Structure

**Figure 3–4 Window Changes Data Structure (MIT C Binding)**

```
typedef struct {
    int x,y;
    int width, height;
    int border_width;
    Window sibling;
    int stack_mode;
} XWindowChanges
```

**Table 3–6 Members of the Window Changes Data Structure (MIT C Binding)**

Member Name	Contents
x	Defines, with the y member, the new location of the window relative to the origin of its parent.
y	Defines, with the x member, the new location of the window relative to the origin of its parent.
width	Defines the new width of the window, excluding the border.
height	Defines the new height of the window, excluding the border.
border_width	Specifies the new window border in pixels.
sibling	Specifies the sibling window for stacking order.
stack_mode	Defines how the window is restacked.

### 3.3 Window Attributes Data Structure

The window attributes data structure specifies and receives information about a window. Use this data structure when you want to obtain the current attributes of a window with the GET WINDOW ATTRIBUTES routine.

The data structure for the VAX binding is shown in Figure 3–5, and information about members in the data structure is described in Table 3–7.

**Figure 3–5 Window Attributes Data Structure (VAX Binding)**

x\$_l_wdat_x	0
x\$_l_wdat_y	4
x\$_l_wdat_width	8
x\$_l_wdat_height	12
x\$_l_wdat_border_width	16

(continued on next page)

## Window Routines

### 3.3 Window Attributes Data Structure

Figure 3–5 (Cont.) Window Attributes Data Structure (VAX Binding)

x\$_l_wdat_depth	20
x\$_l_wdat_visual	24
x\$_l_wdat_root	28
x\$_l_wdat_class	32
x\$_l_wdat_bit_gravity	36
x\$_l_wdat_win_gravity	40
x\$_l_wdat_backing_store	44
x\$_l_wdat_backing_planes	48
x\$_l_wdat_backing_pixel	52
x\$_l_wdat_save_under	56
x\$_l_wdat_colormap	60
x\$_l_wdat_map_installed	64
x\$_l_wdat_map_state	68
x\$_l_wdat_all_event_masks	72
x\$_l_wdat_your_event_mask	76
x\$_l_wdat_not_propagate_mask	80
x\$_l_wdat_override_redirect	84
x\$_l_wdat_screen	88

Table 3–7 Members of the Window Attributes Data Structure (VAX Binding)

Member Name	Contents
X\$_L_WDAT_X	Specifies the x-coordinate of the upper left outside corner of the window relative to its parent.
X\$_L_WDAT_Y	Specifies the y-coordinate of the upper left outside corner of the window relative to its parent.
X\$_L_WDAT_WIDTH	Specifies the width of the window, excluding the window border, in pixels.

(continued on next page)

## Window Routines

### 3.3 Window Attributes Data Structure

**Table 3–7 (Cont.) Members of the Window Attributes Data Structure (VAX Binding)**

Member Name	Contents
X\$_WDAT_HEIGHT	Specifies the height of the window, excluding the window border, in pixels.
X\$_WDAT_BORDER_WIDTH	Specifies the width of the window border in pixels.
X\$_WDAT_DEPTH	Specifies the bits per pixel of the window.
X\$_WDAT_VISUAL	The VISUAL structure associated with the window. The VISUAL structure specifies how displays should treat color resources.
X\$_WDAT_ROOT	Identifies the screen with which the window is associated.
X\$_WDAT_CLASS	Specifies whether the window accepts input and output, or input only.
X\$_WDAT_BIT_GRAVITY	Specifies how pixels should be moved when the window is resized.
X\$_WDAT_WIN_GRAVITY	Specifies how the window should be repositioned when its parent is resized.
X\$_WDAT_BACKING_STORE	Indicates whether or not the server should maintain a record of portions of a window that are obscured when the window is mapped. In this release clients must maintain window contents.
X\$_WDAT_BACKING_PLANES	Indicates (with bits set to 1) which bit planes of the window hold dynamic data that must be preserved in backing stores and during save unders. In this release clients must maintain their own data.
X\$_WDAT_BACKING_PIXEL	Defines what values to use in planes not specified by X\$_WDAT_BACKING_PLANES. In this release clients must maintain their own values.
X\$_SWDA_SAVE_UNDER	Setting this member to true informs the server that the client would like the contents of the screen saved when the window obscures them. Saving the contents of obscured portions of the screen is not guaranteed.
X\$_WDAT_COLORMAP	Specifies the color map, if any, that best reflects the colors of the window. The color map must have the same visual type as the window. If it does not, an error occurs.
X\$_WDAT_MAP_INSTALLED	If set to true, indicates that the color map is currently installed and the window is being displayed in its correct colors.

(continued on next page)



## Window Routines

### 3.3 Window Attributes Data Structure

**Table 3–7 (Cont.) Members of the Window Attributes Data Structure (VAX Binding)**

Member Name	Contents								
X\$_WDAT_MAP_STATE	Indicates whether the window is mapped and viewable. Clients can specify the following constants: <table border="1"><thead><tr><th>Constant Name</th><th>Description</th></tr></thead><tbody><tr><td>x\$c_is_unmapped</td><td>Indicates that the window is not mapped.</td></tr><tr><td>x\$c_is_unviewable</td><td>Indicates that the window is mapped, but that one of its ancestors is unmapped, causing the window to be unviewable.</td></tr><tr><td>x\$c_is_viewable</td><td>Indicates that the window is mapped and viewable.</td></tr></tbody></table>	Constant Name	Description	x\$c_is_unmapped	Indicates that the window is not mapped.	x\$c_is_unviewable	Indicates that the window is mapped, but that one of its ancestors is unmapped, causing the window to be unviewable.	x\$c_is_viewable	Indicates that the window is mapped and viewable.
Constant Name	Description								
x\$c_is_unmapped	Indicates that the window is not mapped.								
x\$c_is_unviewable	Indicates that the window is mapped, but that one of its ancestors is unmapped, causing the window to be unviewable.								
x\$c_is_viewable	Indicates that the window is mapped and viewable.								
X\$_WDAT_ALL_EVENTS_MASK	Indicates the set of events in which all applications have an interest. X\$_WDAT_ALL_EVENTS_MASK is the inclusive-OR of all event masks set for the window.								
X\$_WDAT_YOUR_EVENT_MASK	Indicates the events about which the querying client is interested in receiving notice.								
X\$_WDAT_DO-NOT_PROPAGATE_MASK	Defines which events should not be propagated to its ancestors when no application has the event type selected in the window.								
X\$_WDAT_OVERRIDE_REDIRECT	Specifies whether requests to map and configure the window should override a request by another client to redirect those calls. Typically, this mask, which informs the window manager not to tamper with the window, should be used only on subwindows such as menus.								
X\$_WDAT_SCREEN	Specifies the screen on which the window is mapped.								

The data structure for the MIT C binding is shown in Figure 3–6, and information about members in the data structure is described in Table 3–8.

## Window Routines

### 3.3 Window Attributes Data Structure

**Figure 3–6 Window Attributes Data Structure (MIT C Binding)**

---

```

typedef struct {
    int x,y;
    int width,height;
    int border_width;
    int depth;
    Visual *visual
    Window root;
    int class;
    int bit_gravity;
    int win_gravity;
    int backing_store;
    unsigned long backing_planes;
    unsigned long backing_pixel;
    Bool save_under;
    Colormap colormap;
    Bool map_installed;
    int map_state;
    long all_event_masks;
    long your_event_mask;
    long do_not_propagate_mask;
    Bool override_redirect;
    Screen *screen
} XWindowAttributes

```

---

**Table 3–8 Members of the Window Attributes Data Structure (MIT C Binding)**

Member Name	Contents
x	Specifies, with the y member, the coordinates of the upper left corner of the window relative to its parent.
y	Specifies, with the x member, the coordinates of the upper left corner of the window relative to its parent.
width	Specifies the width of the window, excluding the window border, in pixels.
height	Specifies the height of the window, excluding the window border, in pixels.
border_width	Specifies the width of the window border in pixels.
depth	Specifies the bits per pixel of the window.
visual	A pointer to a VISUAL structure associated with the window. The VISUAL structure specifies how displays should treat color resources.
root	Identifies the screen with which the window is associated.
class	Specifies whether the window accepts input and output, or input only.
bit_gravity	Specifies how pixels should be moved when the window is resized.
win_gravity	Specifies how the window should be repositioned when its parent is resized.
backing_store	Indicates whether or not the server should maintain a record of portions of a window that are obscured when the window is mapped. In this release clients must maintain contents of obscured windows.

(continued on next page)

## Window Routines

### 3.3 Window Attributes Data Structure

**Table 3–8 (Cont.) Members of the Window Attributes Data Structure (MIT C Binding)**

Member Name	Contents
backing_planes	Indicates (with bits set to 1) which bit planes of the window hold dynamic data that must be preserved in backing stores and during save unders. In this release clients must maintain data to be preserved.
backing_pixel	Defines what values to use in planes not specified by the backing_planes member. In this release clients must maintain values to be saved.
save_under	Setting the save_under member to true informs the server that the client would like the contents of the screen saved when the window obscures them. Saving the contents of obscured portions of the screen is not guaranteed.
colormap	Specifies the color map, if any, that best reflects the colors of the window. The color map must have the same visual type as the window. If it does not, an error occurs.
map_installed	If set to true, the map_installed member indicates that the color map is currently installed and the window is being displayed in its correct colors.
map_state	Indicates whether the window is mapped. Clients can specify the following constants:
<hr/>	
<b>Constant</b>	
<b>Name</b>	<b>Description</b>
IsUnmapped	Indicates that the window is not mapped
IsUnviewable	Indicates that the window is mapped, but that one of its ancestors is unmapped, causing the window to be unviewable
IsViewable	Indicates that the window is mapped and viewable
<hr/>	
all_events_mask	Indicates the set of events in which all applications have an interest. The all_events_mask member is the inclusive-OR of event masks set for the window.
your_event_mask	Indicates the events about which the querying application is interested in receiving notice.
do_not_propagate	Defines which events should not be propagated to its ancestors when no application has the event type selected in the window.
override_redirect	Specifies whether requests to map and configure the window should override a request by another client to redirect those calls. Typically, this mask, which informs the window manager not to tamper with the window, should be used only on subwindows such as menus.
screen	Specifies the screen on which the window is mapped.

## 3.4 Window Routines

The following pages describe the Xlib window routines.

## CHANGE WINDOW ATTRIBUTES

Sets the attributes of a specified window.

**VAX FORMAT**    **X\$CHANGE\_WINDOW\_ATTRIBUTES**  
*(display, window\_id, attributes\_mask, attributes)*

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
attributes_mask	uns longword	uns longword	read	reference
attributes	record	x\$set_win_ attributes	read	reference

**MIT C FORMAT**    **XChangeWindowAttributes**  
*(display, window\_id, attributes\_mask, attributes)*

**argument  
information**

```
XChangeWindowAttributes(display, window_id, attributes_mask,
                        attributes)
Display *display;
Window window_id;
unsigned long attributes_mask;
XSetWindowAttributes *attributes;
```

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window whose attributes are to be changed. The identifier of the window was originally specified by any CREATE WINDOW request.

***attributes\_mask***

The window attributes defined in the **attributes** argument that will be changed. If the value of **attributes\_mask** is 0, the **attributes** argument is ignored and is not referenced. See Table 3-9.

## Window Routines

### CHANGE WINDOW ATTRIBUTES

**Table 3–9 CHANGE WINDOW ATTRIBUTES Flags**

Flag Name	Member
x\$m_cw_back_pixmap	X\$_SWDA_BACKGROUND_PIXMAP
x\$m_cw_background_pixel	X\$_SWDA_BACKGROUND_PIXEL
x\$m_cw_border_pixmap	X\$_SWDA_BORDER_PIXMAP
x\$m_cw_border_pixel	X\$_SWDA_BORDER_PIXEL
x\$m_cw_bit_gravity	X\$_SWDA_BIT_GRAVITY
x\$m_cw_win_gravity	X\$_SWDA_WIN_GRAVITY
x\$m_cw_backing_store	X\$_SWDA_BACKING_STORE
x\$m_cw_backing_planes	X\$_SWDA_BACKING_PLANES
x\$m_cw_backing_pixel	X\$_SWDA_BACKING_PIXEL
x\$m_cw_override_redirect	X\$_SWDA_OVERRIDE_REDIRECT
x\$m_cw_save_under	X\$_SWDA_SAVE_UNDER
x\$m_cw_event_mask	X\$_SWDA_EVENT_MASK
x\$m_cw_dont_propagate	X\$_SWDA_DO_NOT_PROPAGATE_MASK
x\$m_cw_colormap	X\$_SWDA_COLORMAP
x\$m_cw_cursor	X\$_SWDA_CURSOR

#### ***attributes***

A pointer to the attribute data structure containing values to be changed.

For more information about the set window attributes data structure, see Section 3.1.

#### **DESCRIPTION**

CHANGE WINDOW ATTRIBUTES sets specific window attributes for a specified window.

Depending on which attributes are specified in **attributes**, CHANGE WINDOW ATTRIBUTES can affect the specified window as follows:

- Changing the background does not change the window contents.
- Changing the background of a root window to None or ParentRelative restores the default background pixmap.
- Changing the window gravity does not affect the current window position.
- Changing the color map of a window generates a Colormap Notify event.
- Changing the cursor of a root window to None restores the default cursor.
- If a client attempts to select the Substructure Redirect event, the Resize Redirect event, or the Button Press event and another client has already selected it, the X server generates a Bad Access error.

# Window Routines

## CHANGE WINDOW ATTRIBUTES

### X ERRORS

VAX	C	Description
X\$C_BAD_ACCESS	BadAccess	<p>Possible causes are as follows:</p> <ul style="list-style-type: none"> <li>• An attempt to grab a key/button combination that has already been grabbed by another client</li> <li>• An attempt to free a color map entry that was not allocated by the client</li> <li>• An attempt to store in to a read-only or unallocated color map entry</li> <li>• An attempt to modify the access control list from other than the local host</li> <li>• An attempt to select an event type that at most one client can select at a time, when another client has already selected it</li> </ul>
X\$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X\$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.
X\$C_BAD_MATCH	BadMatch	<p>Possible causes are as follows:</p> <ul style="list-style-type: none"> <li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>• An input-only window is used as a drawable.</li> <li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>• An input-only window lacks this attribute.</li> </ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### CIRCULATE SUBWINDOWS

---

## CIRCULATE SUBWINDOWS

Circulates the appropriate child of a window in a specified direction.

---

### VAX FORMAT X\$CIRCULATE\_SUBWINDOWS

*(display, window\_id, direction)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
direction	longword	longword	read	reference

---

---

### MIT C FORMAT XCirculateSubwindows

*(display, window\_id, direction)*

#### argument information

```
XCirculateSubwindows(display, window_id, direction)
    Display *display;
    Window window_id;
    int direction;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***window\_id***

The identifier of the parent of the window that you want to circulate. The identifier of the window was originally returned by any CREATE WINDOW request.

### ***direction***

The direction in which you want to circulate the window. Valid values are listed as follows:

---

VAX	C	Description
X\$_RAISE_LOWEST	RaiseLowest	The lowest window in the same stack as the specified window is raised.
X\$_LOWER_HIGHEST	LowerHighest	The highest window in the same stack as the specified window is lowered.

---

## Window Routines

### CIRCULATE SUBWINDOWS

---

#### DESCRIPTION

CIRCULATE SUBWINDOWS circulates children of a window in a specified direction.

If the specified direction is Raise Lowest, CIRCULATE SUBWINDOWS raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. If the specified direction is Lower Highest, CIRCULATE SUBWINDOWS lowers the highest mapped child (if any) that occludes another child to the bottom of the stack.

If another client selects the Substructure Redirect event on the window specified in **window\_id**, CIRCULATE SUBWINDOWS generates a circulate request event and performs no further processing.

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.



## Window Routines

### CIRCULATE SUBWINDOWS DOWN

---

## CIRCULATE SUBWINDOWS DOWN

Lowers the highest mapped child window of a window that occludes another window.

---

**VAX FORMAT**     **X\$CIRCULATE\_SUBWINDOWS\_DOWN**  
*(display, window\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     **XCirculateSubwindowsDown**  
*(display, window\_id)*

**argument  
information**

```
XCirculateSubwindowsDown(display, window_id)
Display *display;
Window window_id;
```

---

**ARGUMENTS**

***display***

The *display* information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the parent window of the highest mapped child window. The identifier of the parent window was originally returned by any CREATE WINDOW request.

---

**DESCRIPTION**

CIRCULATE SUBWINDOWS DOWN lowers the highest mapped child of a window that partially or completely occludes another child window.

Lowering the child window generates exposure events on any window that the child window had occluded.

If another client has selected Substructure Redirect on the window specified in **window\_id**, CIRCULATE SUBWINDOWS DOWN generates a circulate request event and performs no further processing.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### CIRCULATE SUBWINDOWS UP

---

## CIRCULATE SUBWINDOWS UP

Raises the lowest mapped child of an occluded window.

---

**VAX FORMAT**    **X\$CIRCULATE\_SUBWINDOWS\_UP**  
*(display, window\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XCirculateSubwindowsUp**  
*(display,window\_id)*

**argument  
information**

```
XCirculateSubwindowsUp(display, window_id)
    Display *display;
    Window window_id;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the parent of the lowest mapped child window. The identifier of the window was originally returned by any CREATE WINDOW request.

---

### DESCRIPTION

CIRCULATE SUBWINDOWS UP raises the lowest mapped child of a partially or completely occluded window.

Raising the child window generates exposure events on portions of the child window that had been occluded, as well as on any descendants of the child window. CIRCULATE SUBWINDOWS UP also generates a Circulate Notify event.

If another client has selected Substructure Redirect on the window specified in **window\_id**, CIRCULATE SUBWINDOWS UP generates a circulate request event and performs no further processing.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### CONFIGURE WINDOW

---

## CONFIGURE WINDOW

Reconfigures a window's size, position, border, and stacking order.

---

**VAX FORMAT**     **X\$CONFIGURE\_WINDOW**  
(*display, window\_id, change\_mask, values*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
change_mask	mask_uns longword	uns longword	read	reference
values	record	x\$window_ changes	read	reference

---

---

**MIT C FORMAT**     **XConfigureWindow**  
(*display, window\_id, change\_mask, values*)

**argument  
information**

```
XConfigureWindow(display, window_id, change_mask, values)
Display *display;
Window window_id;
unsigned int change_mask;
XWindowChanges *values;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be reconfigured. The identifier of the window was originally returned by any CREATE WINDOW request.

***change\_mask***

A bit mask that specifies the values that are to be set using the information in the data structure that the **values** argument points to.

Table 3-10 lists each bit for the **change\_mask** argument, its predefined value, and its description.

## Window Routines CONFIGURE WINDOW

**Table 3–10 Change Mask Bits**

Bit	VAX Value	C Value	Meaning when Set
0	X\$_CW_X	CWY	Change x-coordinate
1	X\$_CW_Y	CWY	Change y-coordinate
2	X\$_CW_WIDTH	CWWidth	Change width
3	X\$_CW_HEIGHT	CWHeight	Change height
4	X\$_CW_BORDER_WIDTH	CWBWidth	Change border width
5	X\$_CW_SIBLING	CWSibling	Change sibling
6	X\$_CW_STACK_MODE	CWStackMode	Change stack mode
7	NONE	None	Reserved
8	NONE	None	Reserved

### *values*

A pointer to the window changes data structure.

For more information about the window changes data structure, see Section 3.2.

---

## DESCRIPTION

CONFIGURE WINDOW reconfigures a window's size, position, border, and stacking order.

A Bad Match error is generated if a sibling is specified without a stack mode or if the window is not actually a sibling. Note that the computations for stack mode are performed with respect to the window's final geometry, not its initial geometry. Any backing store contents of the window, its inferiors, and other newly visible windows are either discarded or changed to reflect the current screen contents.

---

## X ERRORS

VAX	C	Description
X\$_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"> <li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>• An input-only window is used as a drawable.</li> <li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>• An input-only window lacks this attribute.</li> </ul>

## Window Routines

### CONFIGURE WINDOW

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## CREATE SIMPLE WINDOW

Creates a subwindow of a specified parent window.

---

**VAX FORMAT**     *window\_id\_return = X\$CREATE\_SIMPLE\_WINDOW*  
*(display, parent\_id, x\_coord, y\_coord, width, height,*  
*border\_width, border\_id, background\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
window_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
parent_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference
border_width	uns longword	uns longword	read	reference
border_id	identifier	uns longword	read	reference
background_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *window\_id\_return = XCreateSimpleWindow*  
*(display, parent\_id, x\_coord, y\_coord, width, height,*  
*border\_width, border\_id, background\_id)*

**argument  
information**

```
Window XCreateSimpleWindow(display, parent_id, x_coord, y_coord,  
                           width, height, border_width,  
                           border_id, background_id)  
  
Display *display;  
Window parent_id;  
int x_coord, y_coord;  
unsigned int width, height, border_width;  
unsigned long border_id;  
unsigned long background_id;
```

---

**RETURNS**

***window\_id\_return***  
The identifier of the new subwindow.



## Window Routines

### CREATE SIMPLE WINDOW

---

#### ARGUMENTS

##### ***display***

The display information originally returned by OPEN DISPLAY.

##### ***parent\_id***

The identifier of the parent window.

##### ***x\_coord***

The x-coordinate of the new subwindow. This coordinate is relative to the inside of the parent window. The x- and y-coordinates define the upper left corner of the new subwindow.

##### ***y\_coord***

The y-coordinate of the new subwindow. This coordinate is relative to the inside of the parent window. The x- and y-coordinates define the upper left corner of the new subwindow.

##### ***width***

The width of the new subwindow. The width and height represent the outline of the new subwindow.

Width and height represent the new subwindow's inside dimensions; they do not include the new subwindow's borders, which are outside the window.

##### ***height***

The height of the new subwindow. The height and width represent the outline of the new subwindow.

Height and width represent the new subwindow's inside dimensions; they do not include the new subwindow's borders, which are outside the window.

##### ***border\_width***

The width, in pixels, of the new subwindow's border.

##### ***border\_id***

The identifier of the pixmap used to specify the border pattern.

##### ***background\_id***

The identifier of the pixmap that specifies the background.

---

#### DESCRIPTION

CREATE SIMPLE WINDOW creates a subwindow of a specified parent window from an application. The identifiers of the pixmaps were originally returned by CREATE PIXMAP.

CREATE SIMPLE WINDOW returns the window identifier of the new subwindow. All subsequent operations that are performed on the new subwindow use this window identifier as the identifier of that window. The window identifier remains associated with the subwindow until you destroy it.

## Window Routines

### CREATE SIMPLE WINDOW

The outline of the new subwindow is defined by the arguments **x\_coord**, **y\_coord**, **width**, and **height**. The x- and y-coordinates are relative to the inside of the parent window and define the upper left corner of the new subwindow. The width and height determine the area inside the window of the border. The width and height must be nonzero; otherwise a Bad Value error is returned.

If the border width is specified, the x- and y-coordinates specify the origin at the border, and the origin of the window is offset by the border width.

When you create a window with the CREATE SIMPLE WINDOW routine, the new window inherits the depth, class, and visual identifier of the parent window. You cannot change these attributes. Upon creation, the new subwindow does not have an associated icon window in the property list. The name of the new subwindow is the null string.

Once you create a window with the CREATE SIMPLE WINDOW routine, you must then call the MAP WINDOW routine to display the window on the screen. In order for the window to appear on the screen, all of the window's ancestors must also be mapped, and the window cannot be obscured by any of its ancestors. The new subwindow does not have its own cursor at creation time; it uses the parent window's cursor until its own cursor is registered.

The CREATE SIMPLE WINDOW routine generates a Create Notify event.

To create a subwindow and specify its attributes, use the CREATE WINDOW routine.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

## Window Routines

### CREATE SIMPLE WINDOW

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## CREATE WINDOW

Creates a subwindow of a parent window and defines the attributes of the subwindow.

**VAX FORMAT**     *window\_id\_return = X\$CREATE\_WINDOW*  
                           (*display, parent\_id, x\_coord, y\_coord, width,*  
                           *height, border\_width, depth, class, visual\_struct,*  
                           *attributes\_mask, attributes)*

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
window_id_return	identifier	uns longword	write	reference
display	identifier	uns longword	read	reference
parent_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference
border_width	uns longword	uns longword	read	reference
depth	longword	longword	read	reference
class	longword	longword	read	reference
visual_struct	record	x\$visual	read	reference
attributes_mask	uns longword	uns longword	read	reference
attributes	record	x\$set_win_attributes	read	reference

**MIT C FORMAT**     *window\_id\_return = XCreateWindow*  
                           (*display, parent\_id, x\_coord, y\_coord, width,*  
                           *height, border\_width, depth, class, visual\_struct,*  
                           *attributes\_mask, attributes)*

## Window Routines

### CREATE WINDOW

---

#### argument information

```
Window XCreateWindow(display, parent_id, x_coord, y_coord, width,
                    height, border_width, depth, class,
                    visual_struct, attributes_mask, attributes)
Display *display;
Window parent_id;
int x_coord, y_coord;
unsigned int width, height;
unsigned int border_width;
int depth;
unsigned int class;
Visual *visual_struct;
unsigned long attributes_mask;
XSetWindowAttributes *attributes;
```

---

#### RETURNS

##### *window\_id\_return*

The identifier of the new subwindow that is created.

---

#### ARGUMENTS

##### *display*

The display information originally returned by OPEN DISPLAY.

##### *parent\_id*

The identifier of the window that is the parent of the new subwindow.

##### *x\_coord*

The x-coordinate of the new subwindow. This coordinate is relative to the inside of the parent window. The x- and y-coordinates define the upper left corner of the new subwindow.

##### *y\_coord*

The y-coordinate of the new subwindow. This coordinate is relative to the inside of the parent window. The x- and y-coordinates define the upper left corner of the new subwindow.

##### *width*

The width, in pixels, of the new subwindow. The width and height determine the area of the new subwindow.

The **width** and the **height** arguments represent the new subwindow's inside dimensions; they do not include the new subwindow's borders, which are outside the window.

##### *height*

The height, in pixels, of the new subwindow. The height and width determine the area of the new subwindow.

The **height** and the **width** arguments represent the new subwindow's inside dimensions; they do not include the new subwindow's borders, which are outside the window.

##### *border\_width*

The width, in pixels, of the new subwindow's border. For input-only windows, **border\_width** must be zero, otherwise a Bad Match error is generated.

#### ***depth***

The depth, in bits per pixel, of the new subwindow. If the value of **depth** equals zero and the **class** argument is either input output or Copy From Parent, the depth of the new subwindow is the same as the depth of the parent window. For input only windows, **depth** must be zero.

#### ***class***

The class of the new subwindow.

If the parent window is input only, the class of the subwindow cannot be input output.

A window with a class of input only cannot be used as output.

For Copy From Parent windows, the class of the new subwindow is the same as the parent window.

#### ***visual\_struct***

A pointer to a visual structure associated with the window.

#### ***attributes\_mask***

The window attributes to be specified in the **attributes** argument. For any attributes not specified, default values are used. If the value of **attributes\_mask** is zero, the attributes argument is ignored and is not referenced.

#### ***attributes***

A pointer to the set window attributes data structure for the window.

---

## DESCRIPTION

CREATE WINDOW creates a subwindow of a specified parent window and sets the attributes of the new subwindow. CREATE WINDOW returns the window identifier of the new subwindow. All subsequent operations that are performed on the new subwindow use this window identifier as the identifier of that window. The window identifier remains associated with the new subwindow until you destroy the new subwindow.

The outline of the new subwindow is defined by the arguments **x\_coord**, **y\_coord**, **width**, and **height**. The x- and y-coordinates are relative to the inside of the parent window and define the upper left corner of the new subwindow. The width and height determine the area inside the border. The width and height must be nonzero, otherwise a Bad Value error is returned.

If the border width is specified, the x- and y-coordinates specify the origin at the border, and the origin of the window is offset by the border width.

An input-only window cannot process graphics requests, exposure events, or visibly notify events. In addition, an input-only window cannot be used as a drawable (a source or destination for graphics requests). Input-only and input-output windows act identically in all other respects. Only the following window attributes are defined for input-only windows:

- Window gravity
- Event mask
- Do not propagate mask

## Window Routines

### CREATE WINDOW

- Override redirect
- Cursor

If you specify any other attribute for an input only window, a Bad Match error is generated.

The CREATE WINDOW routine generates a Create Notify event.

CREATE WINDOW differs from CREATE SIMPLE WINDOW because it lets you specify the new subwindow's attributes, such as class, visual type, and attributes contained in the set window attributes data structure. With CREATE SIMPLE WINDOW, these attributes are automatically inherited from the parent window. You specify the attributes of the new subwindow by setting the appropriate bits in **attributes\_mask** to indicate the attributes that are set in **attributes**. For more information about the set window attributes data structure, see Section 3.1.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X\$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"> <li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>• An input-only window is used as a drawable.</li> <li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>• An input-only window lacks this attribute.</li> </ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## DESTROY SUBWINDOWS

Destroys all subwindows of a window.

---

**VAX FORMAT**     **X\$DESTROY\_SUBWINDOWS**  
                          (*display, window\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     **XDestroySubwindows**  
                          (*display, window\_id*)

**argument  
information**

```
XDestroySubwindows (display, window_id)
    Display *display;
    Window window_id;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window for which all subwindows are to be destroyed. The identifier of the window was originally returned by any CREATE WINDOW request.

---

**DESCRIPTION**

DESTROY SUBWINDOWS destroys all subwindows (children) of a specified window in bottom-to-top stacking order.

After you destroy a subwindow, you cannot reference that subwindow again. If any mapped subwindows are destroyed, DESTROY SUBWINDOWS generates exposure events on the window specified in **window\_id**.

When you want to destroy not only the subwindows of a given window, but also the window itself, use DESTROY WINDOW.



## Window Routines

### DESTROY SUBWINDOWS

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## DESTROY WINDOW

Destroys a window and all of its subwindows.

**VAX FORMAT**    **X\$DESTROY\_WINDOW**    (*display, window\_id*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

**MIT C FORMAT**    **XDestroyWindow**    (*display, window\_id*)

**argument  
information**

```
XDestroyWindow(display, window_id)
    Display *display;
    Window window_id;
```

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be destroyed. The identifier of the window was originally returned by any CREATE WINDOW request.

### DESCRIPTION

DESTROY WINDOW unmaps and destroys the specified window and all of its subwindows.

If the specified window was mapped before calling DESTROY WINDOW, the window is unmapped automatically. The window and all subwindows are then destroyed, and a destroy notify event is generated for each window. Destroy notify events are first generated for the inferiors or ancestors of the window before being generated on the window itself.

If you specify a root window, no windows are destroyed.

Once you destroy a window, you cannot reference that window again. Destroying a mapped window generates exposure events on other windows that the destroyed window had obscured.

When you want to destroy only the subwindows of a given window, use DESTROY SUBWINDOWS.

## Window Routines

### DESTROY WINDOW

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## GET GEOMETRY

Obtains the current geometry of the specified drawable.

---

**VAX FORMAT**     *status\_return = X\$GET\_GEOMETRY*  
                           (*display, drawable\_id [,window\_id\_return]*  
                           *[,x\_coord\_return][,y\_coord\_return][,width\_return]*  
                           *[,height\_return][,border\_width\_return][,depth\_return]*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
window_id_return	identifier	uns longword	write	reference
x_coord_return	longword	longword	write	reference
y_coord_return	longword	longword	write	reference
width_return	uns longword	uns longword	write	reference
height_return	uns longword	uns longword	write	reference
border_width_return	uns longword	uns longword	write	reference
depth_return	longword	longword	write	reference

---



---

**MIT C FORMAT**     *status\_return = XGetGeometry*  
                           (*display, drawable\_id, window\_id\_return,*  
                           *x\_coord\_return, y\_coord\_return, width\_return,*  
                           *height\_return, border\_width\_return, depth\_return)*

**argument  
information**

---

```
Status XGetGeometry(display, drawable_id, window_id_return,
                    x_coord_return, y_coord_return, width_return,
                    height_return, border_width_return,
                    depth_return)
Display *display;
Drawable drawable_id;
Window *window_id_return;
int *x_coord_return, *y_coord_return;
unsigned int *width_return, *height_return;
unsigned int *border_width_return;
unsigned int *depth_return;
```

## Window Routines

### GET GEOMETRY

---

#### RETURNS

##### ***status\_return***

Specifies whether or not the routine completed successfully.

---

#### ARGUMENTS

##### ***display***

The display information originally returned by OPEN DISPLAY.

##### ***drawable\_id***

The identifier of the drawable for which the current geometry is to be obtained. This can be either a window or a pixmap.

##### ***window\_id\_return***

The identifier of the root window of the specified drawable.

##### **VAX only**

This argument is optional.

##### ***x\_coord\_return***

If the drawable is a window, ***x\_coord\_return*** specifies the x-coordinate of the window. The x- and y-coordinates define the upper left corner of the window, relative to the origin of the parent window. If the drawable is a pixmap, ***x\_coord\_return*** and ***y\_coord\_return*** equal zero.

##### **VAX only**

This argument is optional.

##### ***y\_coord\_return***

If the drawable is a window, ***y\_coord\_return*** specifies the y-coordinate of the window. The x- and y-coordinates define the upper left corner of the window, relative to the origin of the parent window. If the drawable is a pixmap, ***x\_coord\_return*** and ***y\_coord\_return*** equal zero.

##### **VAX only**

This argument is optional.

##### ***width\_return***

The width, in pixels, of the drawable. GET GEOMETRY returns the width of the drawable to this argument. The width and height represent the inside area of the drawable, not including the border of the drawable.

##### **VAX only**

This argument is optional.

#### ***height\_return***

The height, in pixels, of the drawable. GET GEOMETRY returns the height of the drawable to this argument. The width and height represent the inside area of the drawable, not including the border of the drawable.

#### **VAX only**

This argument is optional.

#### ***border\_width\_return***

The width, in pixels, of the new subwindow's border. If the drawable is a window, GET GEOMETRY returns the width of the window's border to this argument. If the drawable is a pixmap, GET GEOMETRY returns zero to this argument.

#### **VAX only**

This argument is optional.

#### ***depth\_return***

The depth of the pixmap. The depth must be supported by the root window of the specified drawable. GET GEOMETRY returns the depth, in bits per pixel, to this argument.

#### **VAX only**

This argument is optional.

---

## DESCRIPTION

GET GEOMETRY obtains the current geometry of the specified drawable.

If the specified drawable is a pixmap, GET GEOMETRY returns the width, height, and depth of the pixmap. GET GEOMETRY returns zero for the **x\_coord\_return**, **y\_coord\_return**, and **border\_width\_return** arguments when the specified drawable is a pixmap.

If the specified drawable is a window, GET GEOMETRY returns the x- and y-coordinates of the upper left corner of the window (relative to its parent's origin), along with the width and height of the window and the width of the window's border. GET GEOMETRY returns zero for the **depth\_return** argument when the drawable is a window.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.

## Window Routines

### GET WINDOW ATTRIBUTES

---

## GET WINDOW ATTRIBUTES

Obtains the attributes of a specified window.

---

**VAX FORMAT**     *status\_return = X\$GET\_WINDOW\_ATTRIBUTES*  
                          (*display, window\_id, window\_attributes\_return*)

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
window_attributes_ return	array	x\$window_ attributes	write	reference

---

---

**MIT C FORMAT**     *status\_return = XGetWindowAttributes*  
                          (*display, window\_id, window\_attributes\_return*)

---

**argument  
information**

```
Status XGetWindowAttributes(display, window_id,  
                             window_attributes_return)  
Display *display;  
Window window_id;  
XWindowAttributes *window_attributes_return;
```

---

### RETURNS

***status\_return***

Return value that specifies whether the routine completed successfully.

**C only**

This argument returns 1 if the routine completes successfully, and zero if it does not complete successfully.

**VAX only**

This argument returns one of the following values:

## Window Routines

### GET WINDOW ATTRIBUTES

---

Value	Description
X\$_ERRORREPLY	Error received from the server—window no longer exists.
SS\$_NORMAL	Success.
Otherwise	Failure for reason given.

---

---

#### ARGUMENTS

##### *display*

The display information originally returned by OPEN DISPLAY.

##### *window\_id*

The identifier of the window whose attributes are to be obtained. The identifier of the window was originally returned by any CREATE WINDOW request.

##### *window\_attributes\_return*

A pointer to an array of attribute data structures, in which each element defines a window attribute. For more information about the window attributes data structure, see Section 3.3.

---

#### DESCRIPTION

GET WINDOW ATTRIBUTES obtains the current attributes for a window.

GET WINDOW ATTRIBUTES returns a status value that specifies whether or not the routine completed successfully. This value is zero if the routine fails and nonzero if the routine succeeds.

---

#### X ERRORS

---

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---



## Window Routines

### LOWER WINDOW

---

## LOWER WINDOW

Lowers a window so that it does not obscure any sibling windows.

---

**VAX FORMAT**    **X\$LOWER\_WINDOW**    (*display, window\_id*)

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XLowerWindow**    (*display, window\_id*)

---

**argument  
information**

```
XLowerWindow(display, window_id)
    Display *display;
    Window window_id;
```

---

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window that you want to lower. The identifier of the window was originally returned by any CREATE WINDOW request.

---

**DESCRIPTION**

LOWER WINDOW lowers a window so that it does not obscure any sibling windows.

Lowering a window does not change the x- and y-coordinates of the window. Lowering a mapped window generates exposure events for any windows that had been obscured.

If the override attribute of the window is false and another client has selected the Substructure Redirect event on the parent window, LOWER WINDOW generates a configure request event and performs no further processing.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### MAP RAISED

---

## MAP RAISED

Maps the specified window and raises it to the top of the stack.

---

**VAX FORMAT**    **X\$MAP\_RAISED**    (*display, window\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XMapRaised**    (*display, window\_id*)

**argument  
information**

```
XMapRaised(display, window_id)
    Display *display;
    Window window_id;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be mapped and raised. The identifier of the window was originally returned by any CREATE WINDOW request.

---

### DESCRIPTION

MAP RAISED maps and raises a specified window.

MAP RAISED is basically the same routine as MAP WINDOW, except that it also raises the specified window to the top of the stack.

---

### X ERRORS

---

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

---

## MAP SUBWINDOWS

Maps all subwindows of a specified window.

---

**VAX FORMAT**    **X\$MAP\_SUBWINDOWS**    (*display, window\_id*)

argument  
information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

**MIT C FORMAT**    **XMapSubwindows**    (*display, window\_id*)

argument  
information

```
XMapSubwindows (display, window_id)
    Display *display;
    Window  window_id;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window for which all subwindows are to be mapped. The identifier of the window was originally returned by any CREATE WINDOW request.

---

### DESCRIPTION

MAP SUBWINDOWS maps all subwindows for a specified window in top-to-bottom stacking order.

MAP SUBWINDOWS generates an exposure event on each newly displayed window.

When you want to map not only the subwindows of a window, but also the window itself, use MAP WINDOW.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### MAP WINDOW

---

## MAP WINDOW

Maps a window and all of its subwindows that have had mapping requests.

---

**VAX FORMAT**    **X\$MAP\_WINDOW**    (*display, window\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XMapWindow**    (*display, window\_id*)

**argument  
information**

```
XMapWindow(display, window_id)
Display *display;
Window window_id;
```

---

**ARGUMENTS**    ***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be mapped. The identifier of the window was originally returned by any CREATE WINDOW request.

If the window specified in **window\_id** has an unmapped ancestor, it is not displayed on the screen until the ancestor is also mapped.

If the window specified in **window\_id** is opaque, MAP WINDOW generates expose events on each opaque window that becomes displayed.

---

**DESCRIPTION**

MAP WINDOW maps a window and all of its subwindows that have had mapping requests.

If you first map a window, then paint it, and then process input events, the window will be painted twice. To avoid painting a window twice, call SELECT INPUT for exposure events, then map the window, and then process input events.

To map all subwindows of a given window, use MAP SUBWINDOWS. Using MAP WINDOW repeatedly is less efficient than using MAP SUBWINDOWS. To map a window and raise it to the top of a stack, use MAP RAISED.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### MOVE RESIZE WINDOW

---

## MOVE RESIZE WINDOW

Changes the location and the size of a specified window.

---

**VAX FORMAT**     **X\$MOVE\_RESIZE\_WINDOW**  
(*display, window\_id, x\_coord, y\_coord, width, height*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**     **XMoveResizeWindow**  
(*display, window\_id, x\_coord, y\_coord, width, height*)

**argument  
information**

```
XMoveResizeWindow(display, window_id, x_coord, y_coord, width,
                  height)
Display *display;
Window window_id;
int x_coord, y_coord;
unsigned int width, height;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window that will be changed in size and location. The identifier of the window was originally returned by any CREATE WINDOW request.

***x\_coord***

The x-coordinate of the window's new location. This coordinate is relative to the parent window. The x- and y-coordinates define the new location of the window's upper left corner.

## Window Routines

### MOVE RESIZE WINDOW

#### *y\_coord*

The y-coordinate of the window's new location. This coordinate is relative to the parent window. The x- and y-coordinates define the new location of the window's upper left corner.

#### *width*

The new width, in pixels, of the window. The width and height define the interior area of the window.

#### *height*

The new height, in pixels, of the window. The width and height define the interior area of the window.

---

### DESCRIPTION

MOVE RESIZE WINDOW changes the size and the location of a specified window.

Configuring a mapped window causes the window to lose its contents and generates an exposure event. Configuring a window also generates exposure events on any other windows that the window had formerly obscured.

If the override attribute of the window is false and another client has selected substructure redirect on the parent window, MOVE RESIZE WINDOW generates a configure request event and performs no further processing.

When you want to change only the size of a window, use RESIZE WINDOW. When you want to change only the location of a window, use MOVE WINDOW.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>



## Window Routines

### MOVE RESIZE WINDOW

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## MOVE WINDOW

Moves and raises a window without changing its size.

**VAX FORMAT**     **X\$MOVE\_WINDOW**  
                           (*display, window\_id, x\_coord, y\_coord*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference

**MIT C FORMAT**    **XMoveWindow**  
                           (*display, window\_id, x\_coord, y\_coord*)

**argument  
information**

```
XMoveWindow(display, window_id, x_coord, y_coord)
Display *display;
Window window_id;
int x_coord, y_coord;
```

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be moved. The identifier of the window was originally returned by any CREATE WINDOW request.

***x\_coord***

The x-coordinate of the window's new location. The x- and y-coordinates define the new location of the upper left corner of the window's border (or the window itself if it has no border).

***y\_coord***

The y-coordinate of the window's new location. The x- and y-coordinates define the new location of the upper left corner of the window's border (or the window itself if it has no border).

## Window Routines

### MOVE WINDOW

---

#### DESCRIPTION

MOVE WINDOW moves and raises a window without changing its size.

MOVE WINDOW does not change the mapping state of the window.

Moving a mapped window generates exposure events on any windows that were obscured. However, if you move a mapped window, it may lose its contents if no backing store exists for the window and the window is obscured by windows other than its subwindows. If the window loses its contents, exposure events are generated for the window and any of its mapped subwindows.

If the override attribute of the window is false and another client has selected substructure redirect on the parent window, MOVE WINDOW generates a configure request event and performs no further processing. The stacking order does not change.

If you want to move a window and change its size, use CONFIGURE WINDOW or MOVE RESIZE WINDOW.

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## QUERY POINTER

Obtains the root window where the pointer is currently located and the pointer coordinates relative to the root window's origin.

### VAX FORMAT

*result\_return = X\$QUERY\_POINTER*  
*(display, window\_id, root\_id\_return [,child\_id\_return]*  
*[,root\_x\_coord\_return] [,root\_y\_coord\_return]*  
*[,win\_x\_coord\_return] [,win\_y\_coord\_return]*  
*[,state\_mask\_return])*

### argument information

Argument	Usage	Data Type	Access	Mechanism
result_return	longword	longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
root_id_return	identifier	uns longword	write	reference
child_id_return	identifier	uns longword	write	reference
root_x_coord_return	longword	longword	write	reference
root_y_coord_return	longword	longword	write	reference
win_x_coord_return	longword	longword	write	reference
win_y_coord_return	longword	longword	write	reference
state_mask_return	uns longword	uns longword	write	reference

### MIT C FORMAT

*result\_return = XQueryPointer*  
*(display, window\_id, root\_id\_return, child\_id\_return,*  
*root\_x\_coord\_return, root\_y\_coord\_return,*  
*win\_x\_coord\_return, win\_y\_coord\_return,*  
*state\_mask\_return)*

### argument information

```

Bool XQueryPointer(display, window_id, root_id_return,
                  child_id_return, root_x_coord_return,
                  root_y_coord_return, win_x_coord_return,
                  win_y_coord_return, state_mask_return)

Display *display;
Window window_id;
Window *root_id_return, *child_id_return;
int *root_x_coord_return, *root_y_coord_return;
int *win_x_coord_return, *win_y_coord_return;
unsigned int *state_mask_return;

```

## Window Routines

### QUERY POINTER

---

#### RETURNS

##### *result\_return*

This argument specifies whether or not the pointer is on the same screen as the window specified in **window\_id**. When true, this argument is a nonzero value. When false, this argument is zero.

---

#### ARGUMENTS

##### *display*

The display information originally returned by OPEN DISPLAY.

##### *window\_id*

The identifier of the window; this window is relative to the pointer to be queried. The identifier of the window was originally returned by any CREATE WINDOW request.

##### *root\_id\_return*

The identifier of the root window of the window specified in **window\_id**. QUERY POINTER returns the root window where the pointer is currently located to this argument.

##### *child\_id\_return*

The identifier of the child window that contains the pointer coordinates. QUERY POINTER returns the child window where the pointer is located (if any) to this argument.

##### VAX only

This argument is optional.

##### *root\_x\_coord\_return*

The x-coordinate of the pointer. This coordinate is relative to the root window's origin. The x- and y-coordinates define the location of the pointer. QUERY POINTER returns the pointer coordinates (relative to the root window's origin) to this argument.

##### VAX only

This argument is optional.

##### *root\_y\_coord\_return*

The y-coordinate of the pointer. This coordinate is relative to the root window's origin. The x- and y-coordinates define the location of the pointer. QUERY POINTER returns the pointer coordinates (relative to the root window's origin) to this argument.

##### VAX only

This argument is optional.

***win\_x\_coord\_return***

The x-coordinate of the window. This coordinate is relative to the root window's origin. The x- and y-coordinates define the location of the pointer. QUERY POINTER returns the pointer coordinates (relative to the root window's origin) to this argument.

**VAX only**

This argument is optional.

***win\_y\_coord\_return***

The y-coordinate of the pointer. This coordinate is relative to the root window's origin. The x- and y-coordinates define the location of the pointer. QUERY POINTER returns the pointer coordinates (relative to the root window's origin) to this argument.

**VAX only**

This argument is optional.

***state\_mask\_return***

The current state of the modifier keys and the buttons. QUERY POINTER returns the current state to this argument.

**VAX only**

This argument is optional.

---

**DESCRIPTION**

QUERY POINTER obtains the root window where the pointer is currently located and the pointer coordinates relative to the root window's origin.

If QUERY POINTER returns zero, then ***win\_x\_coord\_return*** and ***win\_y\_coord\_return*** are equal to zero, because the pointer is not on the same screen as the root window and the child window. If QUERY POINTER returns a nonzero value, then ***win\_x\_coord\_return*** and ***win\_y\_coord\_return*** are the pointer coordinates relative to the origin of the ***window\_id***.

---

**X ERRORS**

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### QUERY TREE

---

## QUERY TREE

Lists the parent, the children, and the number of children for a window.

---

### VAX FORMAT

```
status_return = X$QUERY_TREE  
(display, window_id [,root_id_return]  
[,parent_id_return] [,children_return]  
[,num_children_return] [,children_size_return]  
[,children_buff_return])
```

### argument information

---

Argument	Usage	Data Type	Access	Mechanism
<i>status_return</i>	longword	longword	write	value
<i>display</i>	identifier	uns longword	read	reference
<i>window_id</i>	identifier	uns longword	read	reference
<i>root_id_return</i>	identifier	uns longword	write	reference
<i>parent_id_return</i>	identifier	uns longword	write	reference
<i>children_return</i>	address	uns longword	write	reference
<i>num_children_return</i>	longword	longword	write	reference
<i>children_size_return</i>	longword	longword	write	reference
<i>children_buff_return</i>	identifier	uns longword	write	reference

---

---

### MIT C FORMAT

```
status_return = XQueryTree  
(display, window_id, root_id_return, parent_id_return,  
children_return, num_children_return)
```

### argument information

```
Status XQueryTree(display, window_id, root_id_return,  
                parent_id_return, children_return,  
                num_children_return)  
Display *display;  
Window window_id;  
Window *root_id_return;  
Window *parent_id_return;  
Window **children_return;  
unsigned int *num_children_return;
```

---

**RETURNS**

***status\_return***

Specifies whether the routine completed successfully.

**C only**

This argument returns 1 if the routine completes successfully, and zero if it does not complete successfully.

**VAX only**

This argument returns one of the following values:

Value	Description
X\$_ERRORREPLY	Error received from the server—window no longer exists.
SS\$_NORMAL	Success.
Otherwise	Failure for reason given.

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window for which the window tree information is to be obtained. The identifier of the window was originally returned by any CREATE WINDOW request.

***root\_id\_return***

The identifier of the root window of the specified window.

**VAX only**

This argument is optional.

***parent\_id\_return***

The identifier of the parent window of the specified window.

**VAX only**

This argument is optional.

***children\_return***

The virtual address of a pointer to an array of child windows, returned by the routine and residing in space reserved by Xlib. Each element in the array is a child window of the specified window. The number of elements in the array is specified by **num\_children\_return**.

**VAX only**

This argument is optional.



## Window Routines

### QUERY TREE

#### ***num\_children\_return***

The number of children associated with the specified window.

#### **VAX only**

This argument is optional.

#### ***children\_size\_return (VAX only)***

The size of the buffer containing the child windows, specified in **children\_buff\_return**.

#### ***children\_buff\_return (VAX only)***

A pointer to a data buffer, residing in space you have reserved, where each entry is one child window identifier. The size of the buffer is specified by **children\_size\_return**. This data is returned by the routine.

---

## DESCRIPTION

QUERY TREE lists the parent, the children, and the number of children for a specified window.

QUERY TREE lists the children in bottom-to-top stacking order. QUERY TREE returns a nonzero value if it completes successfully, and zero if it does not.

To specify arguments that describe the child window identifiers returned by the routine, use **children\_return** and **num\_children\_return** to access data owned by Xlib, or use **children\_size\_return** and **children\_buff\_return** to obtain a private copy of the data. To free the storage returned by this routine, use the display routine FREE.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## RAISE WINDOW

Raises a window so that no sibling window obscures it.

### VAX FORMAT `X$RAISE_WINDOW` (*display, window\_id*)

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

### MIT C FORMAT `XRaiseWindow` (*display, window\_id*)

#### argument information

```
XRaiseWindow(display, window_id)
Display *display;
Window window_id;
```

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***window\_id***

The identifier of the window to be raised. The identifier of the specified window was originally returned by any CREATE WINDOW request.

### DESCRIPTION

RAISE WINDOW raises a window so that no sibling window obscures it.

Raising a window does not change the x- and y-coordinates of the window.

Raising a mapped window generates exposure events for that window and for any mapped subwindows that were obscured.

If the override attribute of the window is false and another client has selected substructure redirect on the parent window, RAISE WINDOW generates a configure request event and performs no further processing.

### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### RESIZE WINDOW

---

## RESIZE WINDOW

Changes the size of a window.

---

**VAX FORMAT**     **X\$RESIZE\_WINDOW**  
*(display, window\_id, width, height)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**     **XResizeWindow**  
*(display, window\_id, width, height)*

**argument  
information**

```
XResizeWindow(display, window_id, width, height)
Display *display;
Window window_id;
unsigned int width, height;
```

---

**ARGUMENTS**     ***display***  
The display information originally returned by OPEN DISPLAY.

***window\_id***  
The identifier of the window. The identifier of the window was originally returned by any CREATE WINDOW request.

***width***  
The new width, in pixels, for the window.

***height***  
The new height, in pixels, for the window.

---

**DESCRIPTION**     RESIZE WINDOW changes the size of a specified window.

The width and height define the inside area of the window, not including the window's border.

## Window Routines

### RESIZE WINDOW

Changing the size of a mapped window can cause the window to lose its contents. If you decrease the size of a mapped window, exposure events are generated on any windows that were obscured.

If the `override` attribute of the window is `false` and another client has selected `Substructure Redirect` on the parent window, `RESIZE WINDOW` generates a `configure` request event and performs no further processing.

If you want to change the size of a window and move the window, use `MOVE RESIZE WINDOW`.

---

## X ERRORS

VAX	C	Description
<code>X\$C_BAD_VALUE</code>	<code>BadValue</code>	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
<code>X\$C_BAD_WINDOW</code>	<code>BadWindow</code>	A value that you specified for a window argument does not name a defined window.

## Window Routines

### RESTACK WINDOWS

---

## RESTACK WINDOWS

Restacks an array of windows from top to bottom.

---

**VAX FORMAT**    **X\$RESTACK\_WINDOWS**  
(*display, window\_ids, num\_windows*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_ids	record	uns longword	read	reference
num_windows	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XRestackWindows**  
(*display, window\_ids, num\_windows*)

**argument  
information**

```
XRestackWindows(display, window_ids, num_windows);  
Display *display;  
Window window_ids[];  
int num_windows;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***window\_ids***  
A pointer to an array of windows. Each element is the identifier of a window that will be restacked. The length of the array is specified by **num\_windows**.

***num\_windows***  
The total number of window entries. This value specifies the number of elements in **window\_ids**.

---

**DESCRIPTION**    RESTACK WINDOWS restacks an array of windows from top to bottom. The stacking order of the first window in the array is unaffected, but all other windows in the array are stacked underneath it in the order of the array.

All windows in the specified array must have a common parent.

## Window Routines

### RESTACK WINDOWS

If the override attribute of a window in the array is false and another client has selected substructure redirect on the parent window, RESTACK WINDOWS generates a Configure Request event for each array window that has an override attribute of false and performs no further processing.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## Window Routines

### SET WINDOW BACKGROUND

---

## SET WINDOW BACKGROUND

Sets the background of a specified window to the specified pixel.

---

### VAX FORMAT `X$SET_WINDOW_BACKGROUND` (*display, window\_id, pixel*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
pixel	uns longword	uns longword	read	reference

---

---

### MIT C FORMAT `XSetWindowBackground` (*display, window\_id, pixel*)

#### argument information

```
XSetWindowBackground(display, window_id, pixel)
Display *display;
Window window_id;
unsigned long pixel;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***window\_id***

The identifier of the window. The identifier of the window was originally returned by any CREATE WINDOW request.

#### ***pixel***

The entry in the color map to be used for the window's background.

---

### DESCRIPTION

SET WINDOW BACKGROUND sets the background of a specified window to a specified pixel. SET WINDOW BACKGROUND uses the color map entry specified in **pixel** for the window's background.

SET WINDOW BACKGROUND PIXMAP can be performed only on an opaque window. An attempt to perform SET WINDOW BACKGROUND PIXMAP on an input-only window generates an error.

## Window Routines

### SET WINDOW BACKGROUND

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---



## Window Routines

### SET WINDOW BACKGROUND PIXMAP

---

## SET WINDOW BACKGROUND PIXMAP

Changes the background tile of a specified window.

---

**VAX FORMAT**    **X\$SET\_WINDOW\_BACKGROUND\_PIXMAP**  
(*display, window\_id, background\_pixmap\_id*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
background_pixmap_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetWindowBackgroundPixmap**  
(*display, window\_id, background\_pixmap\_id*)

**argument  
information**

```
XSetWindowBackgroundPixmap(display, window_id,  
                             background_pixmap_id)  
  
Display *display;  
Window window_id;  
Pixmap background_pixmap_id;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window that will have its background tile changed. The identifier of the window was originally returned by any CREATE WINDOW request.

***background\_pixmap\_id***

The identifier of the pixmap to be used as the background tile.

---

**DESCRIPTION**

SET WINDOW BACKGROUND PIXMAP changes the background tile of a specified window.

SET WINDOW BACKGROUND PIXMAP uses the pixmap specified in **background\_pixmap\_id** as a background. If you do not specify a background pixmap, SET WINDOW BACKGROUND PIXMAP uses the background pixmap of the parent window. If you do not specify a background pixmap for the root window, SET WINDOW BACKGROUND PIXMAP restores the default background.

## Window Routines

### SET WINDOW BACKGROUND PIXMAP

SET WINDOW BACKGROUND PIXMAP does not repaint the background and does not change the current contents of the window. After you use SET WINDOW BACKGROUND PIXMAP, you can then clear and repaint the screen; this will not repaint the background you just set.

The background pixmap can be freed immediately if no further explicit references to it are made.

SET WINDOW BACKGROUND PIXMAP can be performed only on an opaque window. An attempt to perform SET WINDOW BACKGROUND PIXMAP on an input-only window generates a Bad Match error.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### SET WINDOW BORDER

---

## SET WINDOW BORDER

Sets a window's border to a specified pixel.

---

**VAX FORMAT**    **X\$SET\_WINDOW\_BORDER**  
*(display, window\_id, pixel)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
pixel	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetWindowBorder**  
*(display, window\_id, pixel)*

**argument  
information**

```
XSetWindowBorder(display, window_id, pixel)
Display *display;
Window window_id;
unsigned long pixel;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window for which the border is to be changed and repainted. The identifier of the window was originally specified by any CREATE WINDOW request.

***pixel***

The entry in the color map to be used for painting the border.

---

**DESCRIPTION**

SET WINDOW BORDER sets the border of a window to a specified pixel. SET WINDOW BORDER uses the color map entry specified in **pixel** for the window's border.

SET WINDOW BORDER cannot be performed on an input-only window; this generates a Bad Match error.

To set the border tile of a window to a specified pixel, use SET WINDOW BORDER TILE.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### SET WINDOW BORDER PIXMAP

---

## SET WINDOW BORDER PIXMAP

Sets the border tile of a specified window.

---

**VAX FORMAT**    **X\$SET\_WINDOW\_BORDER\_PIXMAP**  
*(display, window\_id, border\_pixmap\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
border_pixmap_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetWindowBorderPixmap**  
*(display, window\_id, border\_pixmap\_id)*

**argument  
information**

---

```
XSetWindowBorderPixmap(display, window_id, border_pixmap_id)
Display *display;
Window window_id;
Pixmap border_pixmap_id;
```

---

## ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window that will have its border tile changed and repainted. The identifier of the window was originally returned by any CREATE WINDOW request.

***border\_pixmap\_id***

The identifier of the pixmap to be used as a background.

---

## DESCRIPTION

SET WINDOW BORDER PIXMAP sets the border tile of a specified window.

SET WINDOW BORDER PIXMAP uses the pixmap specified in **border\_pixmap\_id** for the border. SET WINDOW BORDER PIXMAP does not repaint the background; CLEAR can be used to repaint the background.

The border pixmap can be freed immediately if no further explicit references to it are made.

## Window Routines

### SET WINDOW BORDER PIXMAP

SET WINDOW BORDER PIXMAP can be performed only on an input output window that has a border. An attempt to perform SET WINDOW BORDER PIXMAP on an input-only window generates a Bad Match error.

To set the border of a window to a specified pixel, use SET WINDOW BORDER.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### SET WINDOW BORDER WIDTH

---

## SET WINDOW BORDER WIDTH

Changes the border width of a window.

---

**VAX FORMAT**    **X\$SET\_WINDOW\_BORDER\_WIDTH**  
*(display, window\_id, width)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
width	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetWindowBorderWidth**  
*(display, window\_id, width)*

**argument  
information**

---

```
XSetWindowBorderWidth(display, window_id, width)
Display *display;
Window window_id;
unsigned int width;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***window\_id***  
The identifier of the window to be raised. The identifier of the window was originally returned by any CREATE WINDOW request.

***width***  
The new width, in pixels, of the window border.

---

**DESCRIPTION**    SET WINDOW BORDER WIDTH changes the border of a window.

---

### X ERRORS

---

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

---

## TRANSLATE COORDINATES

Transfers coordinates from the coordinate space of one window to another window.

---

**VAX FORMAT**    *bool = X\$TRANSLATE\_COORDINATES*  
                   (*display, src\_window\_id, dst\_window\_id,*  
                   *src\_x\_coord, src\_y\_coord [,dst\_x\_coord\_return]*  
                   *[,dst\_y\_coord\_return] [,child\_id\_return]*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
bool	longword	longword	write	value
display	identifier	uns longword	read	reference
src_window_id	identifier	uns longword	read	reference
dst_window_id	identifier	uns longword	read	reference
src_x_coord	longword	longword	read	reference
src_y_coord	longword	longword	read	reference
dst_x_coord_return	longword	longword	write	reference
dst_y_coord_return	longword	longword	write	reference
child_id_return	identifier	uns longword	write	reference

---



---

**MIT C FORMAT**    *bool = XTranslateCoordinates*  
                   (*display, src\_window\_id, dst\_window\_id, src\_x\_coord,*  
                   *src\_y\_coord, dst\_x\_coord\_return, dst\_y\_coord\_return,*  
                   *child\_id\_return*)

**argument  
information**

---

```
int XTranslateCoordinates(display, src_window_id, dst_window_id,
                        src_x_coord, src_y_coord,
                        dst_x_coord_return, dst_y_coord_return,
                        child_id_return)

Display *display;
Window src_window_id, dst_window_id;
int src_x_coord, src_y_coord;
int *dst_x_coord_return, *dst_y_coord_return;
Window *child_id_return;
```



## Window Routines

### TRANSLATE COORDINATES

---

#### RETURNS

##### *bool*

If TRANSLATE COORDINATES returns zero, this means that the source window and the destination window are on different screens, and that **dst\_x\_coord** and **dst\_y\_coord** are zero. If the destination coordinates are in a mapped child of the destination window, the identifier of that child window is returned in **child\_id\_return**. Otherwise, this argument is set to none.

---

#### ARGUMENTS

##### *display*

The display information originally returned by OPEN DISPLAY.

##### *src\_window\_id*

The identifier of the source window, which was originally returned by any CREATE WINDOW request.

##### *dst\_window\_id*

The identifier of the destination window.

##### *src\_x\_coord*

The x-coordinate of the source window. This coordinate is relative to the origin of the source window. The x- and y-coordinates define the upper left corner of the source window.

##### *src\_y\_coord*

The y-coordinate of the source window. This coordinate is relative to the origin of the source window. The x- and y-coordinates define the upper left corner of the source window.

##### *dst\_x\_coord\_return*

The x-coordinate of the destination window. TRANSLATE COORDINATES returns the source coordinates relative to the origin of the source window to the **dst\_x\_coord\_return** and **dst\_y\_coord\_return** arguments, which are relative to the destination window's origin.

##### **VAX only**

This argument is optional.

##### *dst\_y\_coord\_return*

The y-coordinate of the destination window. TRANSLATE COORDINATES returns the source coordinates relative to the origin of the source window to the **dst\_x\_coord\_return** and **dst\_y\_coord\_return** arguments, which are relative to the destination window's origin.

##### **VAX only**

This argument is optional.

## Window Routines

### TRANSLATE COORDINATES

#### ***child\_id\_return***

If the destination coordinates are contained in a mapped child of the destination window, the identifier of that child window is returned in **child\_id\_return**.

#### **VAX only**

This argument is optional.

---

#### **DESCRIPTION**

TRANSLATE COORDINATES transfers coordinates from the coordinate space of one window to another window. Using TRANSLATE COORDINATES for this purpose frees the server from the need to perform this task.

---

#### **X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Window Routines

### UNMAP SUBWINDOWS

---

## UNMAP SUBWINDOWS

Unmaps all subwindows of a specified window.

---

### VAX FORMAT `X$UNMAP_SUBWINDOWS` (*display, window\_id*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---

---

### MIT C FORMAT `XUnmapSubwindows` (*display, window\_id*)

#### argument information

```
XUnmapSubwindows(display, window_id)
Display *display;
Window window_id;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***window\_id***

The identifier of the window for which all subwindows are to be unmapped. The identifier of the window was originally returned by any CREATE WINDOW request.

---

### DESCRIPTION

UNMAP SUBWINDOWS unmaps all subwindows of a specified window, in bottom-to-top stacking order.

UNMAP SUBWINDOWS generates an Unmap Window event on each subwindow and generates exposure events on windows that were obscured.

When you want to unmap a single window, use UNMAP WINDOW.

---

### X ERRORS

---

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## UNMAP WINDOW

Unmaps a window.

**VAX FORMAT**    **X\$UNMAP\_WINDOW**    (*display, window\_id*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

**MIT C FORMAT**    **XUnmapWindow**    (*display, window\_id*)

**argument  
information**

```
XUnmapWindow(display, window_id)
Display *display;
Window window_id;
```

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be unmapped. The identifier of the window was originally returned by any CREATE WINDOW request.

### DESCRIPTION

UNMAP WINDOW unmaps a window.

When you unmap a window using UNMAP WINDOW, any subwindows of the specified window remain mapped, but they will not be visible until the parent window is mapped again.

Unmapping a window generates exposure events on any windows that were obscured by the window and its children. UNMAP WINDOW also generates an Unmap Window event for the window specified in **window\_id**, unless the server implements backing stores.

If the specified window is already unmapped, UNMAP WINDOW has no effect.

When you want to unmap all subwindows of a specified window, use UNMAP SUBWINDOWS. Using UNMAP SUBWINDOWS is much more efficient than using UNMAP WINDOW repeatedly.

## Window Routines

### UNMAP WINDOW

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

# 4

## Event Routines

---

An event is a report of either a change in the state of a device or the execution of a routine called by a client.

The server reports events related to changes in the state of devices such as keyboards and pointers, changes in windows, graphics operations, client communications, and errors. Xlib maintains an event queue for each connection. The transport buffers events on a per-connection basis and delivers them to the Xlib event queues. An event may wait in a transport buffer until it is added to the appropriate Xlib event queues. It is therefore possible for an event to be available to a connection even though it has not yet been queued in the Xlib event queue.

The event routines allow client programs to select the types of events that they want to be informed about and to manage event processing.

The routines described in this chapter allow you to perform the following operations:

- Select the event types to send to a window
- Send an event to a window
- Manipulate the event queue
- Enable or disable synchronization
- Handle I/O errors
- Handle server errors
- Return error messages

For information on how to use the event routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 4–1.

**Table 4–1 Event Routines**

<b>Routine Name</b>	<b>Description</b>
CHECK IF EVENT	Checks the event queue for the event that you specify and flushes the output buffer if the event is not found.
CHECK MASK EVENT	Removes the next matching event from the queue.
CHECK TYPED EVENT	Returns the next event in the Xlib event queue that matches the event type.

(continued on next page)

## Event Routines

**Table 4–1 (Cont.) Event Routines**

<b>Routine Name</b>	<b>Description</b>
CHECK TYPED WINDOW EVENT	Returns the next matched event in the queue for the specified window.
CHECK WINDOW EVENT	Removes the next matching event from the queue and returns immediately to indicate the status.
EVENTS QUEUED	Checks the number of events in the event queue.
FLUSH	Flushes the Xlib output buffer for a client connection.
GET ERROR DATABASE TEXT	Returns error messages from the error database.
GET ERROR TEXT	Returns a text string that describes the error code that you specify.
GET MOTION EVENTS	Returns pointer motion events for a window based on the time coordinates that you supply.
IF EVENT	Checks the event queue for the event that you specify in the predicate procedure and removes the event if the events match.
MASK EVENT	Removes the next matching event from the queue or blocks until a matching event is received.
NEXT EVENT	Gets the next event from the event queue, then removes the event.
PEEK EVENT	Peeks at and copies an event from the event queue.
PEEK IF EVENT	Checks the event queue for the event that you specify, but does not remove the event from the event queue.
PENDING	Returns the number of pending events.
PUT BACK EVENT	Copies an event back onto the head of the client's Xlib event queue.
SELECT ASYNC EVENT	Specifies an action routine and argument to be called when an event occurs.
SELECT ASYNC INPUT	Specifies an action routine and arguments to be called when some subset of events occurs.
SELECT INPUT	Selects the event types to send to a window.
SEND EVENT	Sends an event to a window without the window requesting the event.
SET AFTER FUNCTION	The synchronization handler to call before returning from an Xlib routine.

(continued on next page)

**Table 4–1 (Cont.) Event Routines**

Routine Name	Description
SET ERROR HANDLER	A user-written nonfatal error handler to be called when an XError event is received.
SET IO ERROR HANDLER	A user-written routine to handle fatal I/O errors.
SYNC	Flushes the client's Xlib output buffer and waits for all requests to be received and processed by the server.
SYNCHRONIZE	Enables or disables synchronization for a display and returns the status of the previous state.
WINDOW EVENT	Removes the next matching event from the queue for the specified window.

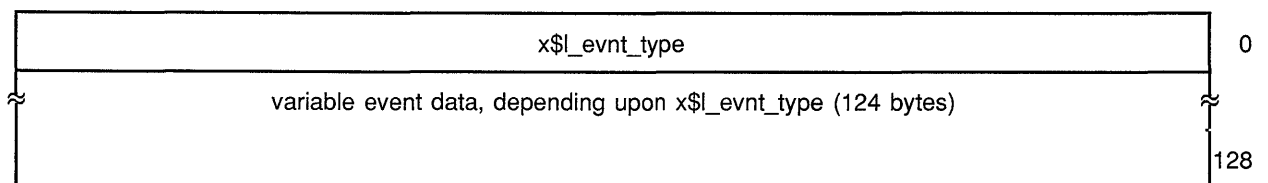
## 4.1 Event Data Structure

The event data structure is a union of structures that stores values for input event attributes. Client programs can use the event mask to select the input event attributes for which they receive notification.

**Note:** See the *VMS DECwindows Xlib Programming Volume* for a complete description of the structures that form the event data structure. The *VMS DECwindows Xlib Programming Volume* includes descriptions of the structure members for both the VAX binding and the MIT C binding.

The VAX binding event structure is shown in Figure 4–1.

**Figure 4–1 Event Data Structure (VAX Binding)**



The members of the structures that comprise the VAX binding event data structure are described in the *VMS DECwindows Guide to Xlib Programming: VAX Binding*.

The MIT C binding event structure is shown in Figure 4–2.



## Event Routines

### 4.1 Event Data Structure

Figure 4–2 Event Data Structure (MIT C Binding)

---

```
typedef union _XEvent {
    int type;
    XAnyEvent xany;
    XKeyEvent xkey;
    XButtonEvent xbutton;
    XMotionEvent xmotion;
    XCrossingEvent xcrossing;
    XFocusChangeEvent xfocus;
    XExposeEvent xexpose;
    XGraphicsExposeEvent xgraphicsexpose;
    XNoExposeEvent xnoexpose;
    XVisibilityEvent xvisibility;
    XCreateWindowEvent xcreatewindow;
    XDestroyWindowEvent xdestroywindow;
    XUnmapEvent xunmap;
    XMapEvent xmap;
    XMapRequestEvent xmaprequest;
    XReparentEvent xreparent;
    XConfigureEvent xconfigure;
    XGravityEvent xgravity;
    XResizeRequestEvent xresizerequest;
    XConfigureRequestEvent xconfigurerequest;
    XCirculateEvent xcirculate;
    XCirculateRequestEvent xcirculaterequest;
    XPropertyEvent xproperty;
    XSelectionClearEvent xselectionclear;
    XSelectionRequestEvent xselectionrequest;
    XSelectionEvent xselection;
    XColormapEvent xcolormap;
    XClientMessageEvent xclient;
    XMappingEvent xmapping;
    XErrorEvent xerror;
    XKeymapEvent xkeymap;
    long pad[24];
} XEvent;
```

---

The members of the structures that comprise the MIT C binding event data structure are described in the *VMS DECwindows Guide to Xlib Programming: MIT C Binding*.

---

## 4.2 The Event Mask

The event routines require that you use the **event\_mask** argument to define the events for which a client receives notice. To do this, you pass an event mask to an Xlib event-handling function that takes an **event\_mask** argument. The event mask name describes the events that you want the server to return to the client.

Table 4–2 lists the elements of the event mask and their descriptions.

## Event Routines

### 4.2 The Event Mask

**Table 4–2 Event Mask Elements**

Bit	VAX Value	MIT C Value	Description
1	X\$_KEY_PRESS	KeyPressMask	Keyboard down events wanted
2	X\$_KEY_RELEASE	KeyReleaseMask	Keyboard up events wanted
3	X\$_BUTTON_PRESS	ButtonPressMask	Pointer button down events wanted
4	X\$_BUTTON_RELEASE	ButtonReleaseMask	Pointer button up events wanted
5	X\$_ENTER_WINDOW	EnterWindowMask	Pointerwindow entry events wanted
6	X\$_LEAVE_WINDOW	LeaveWindowMask	Pointerwindow leave events wanted
7	X\$_POINTER_MOTION	PointerMotionMask	Pointer motion events wanted
8	X\$_POINTER_MOTION_HINT	PointerMotionHintMask	Pointer motion hints wanted
9	X\$_BUTTON1_MOTION	Button1MotionMask	Pointer motion while button 1 down
10	X\$_BUTTON2_MOTION	Button2MotionMask	Pointer motion while button 2 down
11	X\$_BUTTON3_MOTION	Button3MotionMask	Pointer motion while button 3 down
12	X\$_BUTTON4_MOTION	Button4MotionMask	Pointer motion while button 4 down
13	X\$_BUTTON5_MOTION	Button5MotionMask	Pointer motion while button 5 down
14	X\$_BUTTON_MOTION	ButtonMotionMask	Pointer motion while any button down
15	X\$_KEYMAP_STATE	KeymapStateMask	Keyboard state wanted at window entry and focus in
16	X\$_EXPOSURE	ExposureMask	Any exposure wanted
17	X\$_VISIBILITY_CHANGE	VisibilityChangeMask	Any change in visibility wanted
18	X\$_STRUCTURE_NOTIFY	StructureNotifyMask	Any change in window structure wanted
19	X\$_RESIZE_REDIRECT	ResizeRedirectMask	Redirect resize of this window
20	X\$_SUBSTRUCTURE_NOTIFY	SubstructureNotifyMask	Substructure notification wanted
21	X\$_SUBSTRUCTURE_REDIRECT	SubstructureRedirectMask	Redirect substructure of window
22	X\$_FOCUS_CHANGE	FocusChangeMask	Any change in input focus wanted
23	X\$_PROPERTY_CHANGE	PropertyChangeMask	Any change in property wanted
24	X\$_COLORMAP_CHANGE	ColormapChangeMask	Any change in color map wanted
25	X\$_OWNER_GRAB_BUTTON	OwnerGrabButtonMask	Automatic grabs should activate with owner_events set to true

You can also specify X\$\_NO\_EVENT or NoEventMask in **event\_mask** to indicate that no events are wanted.

## Event Routines

### 4.3 The Predicate Procedure

---

#### 4.3 The Predicate Procedure

Xlib provides routines that allow you to see if the next event on the Xlib event queue is the one that your client program wants. These routines require you to provide a predicate procedure, which determines if the next event in the queue matches the one your client program wants.

Your predicate procedure must decide only if the event is useful (for example, by checking the *type* member of the event data structure) and must not call Xlib routines. Xlib routines lock the event queue when they access it; the predicate procedure is called from within an Xlib routine and must not attempt to lock the queue itself.

The format of the predicate procedure in the MIT C binding is as follows:

```
Bool (*predicate) (display, event, args)
    Display *display;
    XEvent *event;
    char *args;
```

The arguments of the predicate procedure are defined as follows:

display	A pointer to the display data structure for which you want to check the input event queue. OPEN DISPLAY returned a pointer to the display structure when it opened the display.
event	A pointer to the event data structure. The event data structure is a union of the individual structures declared for each event type.
args	A pointer to the arguments passed in from IF EVENT, CHECK IF EVENT, or PEEK IF EVENT.

The predicate procedure should return a nonzero (true) or zero (false) value to indicate if the event is the one that should be returned. Xlib calls the predicate procedure once for each event in the queue until it finds a match between the event in the queue and the event specified by the corresponding routine.

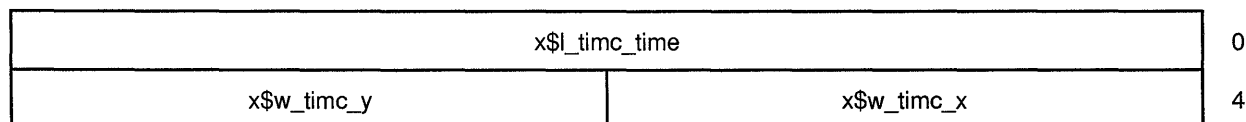
---

#### 4.4 Time Coordinate Data Structure

The time coordinate data structure defines the data structure that is returned by the GET MOTION EVENTS routine.

The VAX binding time coordinate data structure is shown in Figure 4-3.

Figure 4-3 Time Coordinate Data Structure (VAX Binding)



The members of the VAX binding time coordinate structure are described in Table 4-3.

## 4.4 Time Coordinate Data Structure

**Table 4–3 Members of the Time Coordinate Data Structure (VAX Binding)**

Member Name	Contents
X\$C_TIMC_TIME	Set to the time, in milliseconds
X\$C_TIMC_X	Set to the x-coordinate of the pointer cursor and is reported relative to the origin of the specified window
X\$C_TIMC_Y	Set to the y-coordinate of the pointer cursor and is reported relative to the origin of the specified window

The MIT C binding time coordinate data structure is shown in Figure 4–4.

**Figure 4–4 Time Coordinate Data Structure (MIT C Binding)**

```
typedef struct _XTimeCoord {
    Time time;
    short x, y;
} XTimeCoord;
```

The members of the MIT C binding time coordinate structure are described in Table 4–4.

**Table 4–4 Members of the Time Coordinate Data Structure (MIT C Binding)**

Member Name	Contents
time	Set to the time, in milliseconds
x	Set to the x-coordinate of the pointer cursor and are reported relative to the origin of the specified window
y	Set to the y-coordinate of the pointer cursor and are reported relative to the origin of the specified window

## 4.5 Error Handling

There are two default Xlib error handlers: one to handle typically fatal conditions, such as losing the connection to a display, and one to handle error events from the server. The default error handlers print an explanatory message and exit. You can substitute your own user-written error handling routines for the default error handlers at any time. If either of the error handling routines (SET ERROR HANDLER or SET IO ERROR HANDLER) is passed a null pointer, Xlib reinvokes the default error handler.

# Event Routines

## 4.5 Error Handling

Xlib calls the default or user-written error handler whenever an error event is received. The error event is assumed to be nonfatal, and it is acceptable for the error handler to generate a return. The error handler should not perform any operations, either directly or indirectly, on the display.

The Xlib routines can return the error codes described in Table 4–5.

**Table 4–5 Xlib Error Codes**

VAX	MIT C	Description
X\$C_BAD_ACCESS	BadAccess	Possible causes are as follows: <ul style="list-style-type: none"> <li>• An attempt to grab a key/button combination that has already been grabbed by another client</li> <li>• An attempt to free a color map entry that was not allocated by the client</li> <li>• An attempt to store in a read-only or unallocated color map entry</li> <li>• An attempt to modify the access control list from other than the local host</li> <li>• An attempt to select an event type that at most one client can select at a time, when another client has already selected it</li> </ul>
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_ATOM	BadAtom	The value that you specified in an atom argument does not name a defined atom.
X\$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X\$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_ID_CHOICE	BadIDChoice	The value that you chose for a resource identifier is either not included in the range assigned to the client, or it is already in use. Under normal circumstances this cannot occur and should be considered a server or Xlib error.
X\$C_BAD_IMPLEMENTATION	BadImplementation	The server does not implement some aspect of the request. This error is most likely caused by a server extension; a server that generates this error for a core protocol request is deficient. As such, this error is not listed for any particular request. Client programs should be prepared to receive this type of error and either handle or discard it.

(continued on next page)

**Table 4–5 (Cont.) Xlib Error Codes**

VAX	MIT C	Description
X\$C_BAD_LENGTH	BadLength	The length of a request is shorter or longer than required to minimally contain the arguments. This error usually indicates an internal Xlib or server error. The length of a request exceeds the maximum length accepted by the server.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"> <li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>• An input-only window is used as a drawable.</li> <li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>• An input-only window lacks this attribute.</li> </ul>
X\$C_BAD_NAME	BadName	The font or color that you specified does not exist.
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_REQUEST	BadRequest	The major or minor opcode that you specified does not indicate a valid request. This is usually an Xlib or server error.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a Window argument does not name a defined window.

**Note: The Bad Atom, Bad Color, Bad Cursor, Bad Drawable, Bad Font, Bad Pixmap, and Bad Window errors are also used when the argument type is extended by a set of fixed alternatives.**

## 4.6 Error Event Data Structure

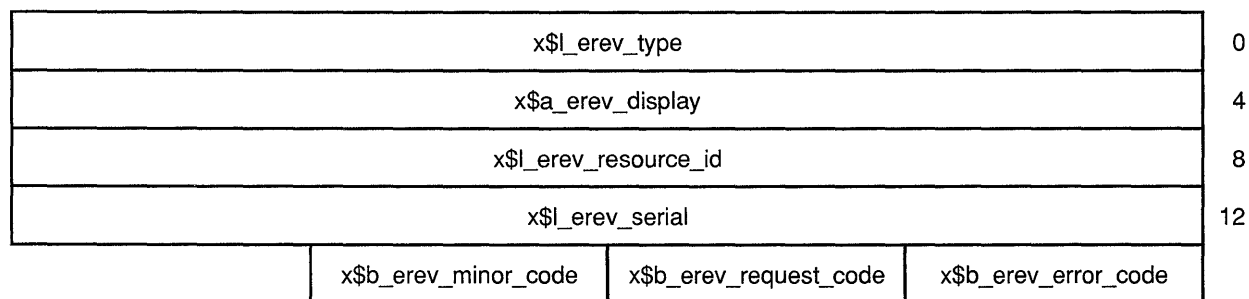
The error event data structure defines the format of errors reported to the default or user-supplied error handlers.

The VAX binding error event data structure is shown in Figure 4–5.

## Event Routines

### 4.6 Error Event Data Structure

Figure 4–5 Error Event Data Structure (VAX Binding)



The members of the VAX binding error event data structure are described in Table 4–6.

Table 4–6 Members of the Error Event Data Structure (VAX Binding)

Member Name	Contents
X\$L_EREV_TYPE	The type of event.
X\$A_EREV_DISPLAY	The display information originally returned by OPEN DISPLAY.
X\$L_EREV_RESOURCE_ID	The resource identifier associated with the error.
X\$L_EREV_SERIAL	The serial number of the request processed by the server immediately before the failing call was made. Xlib and the server use the serial number to identify the protocol request that caused the error.
X\$B_EREV_ERROR_CODE	The error code of the failed request. The error codes are described in Table 4–5.
X\$B_EREV_REQUEST_CODE	The major operation code of the failed request.
X\$B_EREV_MINOR_CODE	The minor operation code of the failed request.

The MIT C binding error event data structure is shown in Figure 4–6.

Figure 4–6 Error Event Data Structure (MIT C Binding)

---

```
typedef struct _XErrorEvent {
    int type;
    Display *display;
    XID resourceid;
    unsigned long serial;
    unsigned char error_code;
    unsigned char request_code;
    unsigned char minor_code;
}XErrorEvent;
```

---

## Event Routines

### 4.6 Error Event Data Structure

The members of the MIT C binding error event data structure are described in Table 4–7.

**Table 4–7 Members of the Error Event Data Structure (MIT C Binding)**

<b>Member Name</b>	<b>Contents</b>
type	The type of event.
display	The display information originally returned by OPEN DISPLAY.
resourceid	The resource identifier associated with the error.
serial	The serial number of the request processed by the server immediately before the failing call was made. Xlib and the server use the serial number to identify the protocol request that caused the error.
error_code	The error code of the failed request. The error codes are described in Table 4–5.
request_code	The major operation code of the failed request.
minor_code	The minor operation code of the failed request.

---

## 4.7 Event Routines

The following pages describe the Xlib event routines.





***predicate***

A pointer to a user-supplied procedure that determines if the next event in the event queue matches the event you specify in the **args** argument. When true, **predicate** returns a nonzero value. When false, **predicate** returns zero. Xlib calls the predicate procedure once for each event in the queue until it returns true.

For more information about the predicate procedure see Section 4.3.

***arg***

The user-specified arguments that are passed to the predicate procedure.

---

**DESCRIPTION**

CHECK IF EVENT returns true when the specified predicate procedure returns true for the next event in the queue that matches the specified event. If the predicate procedure finds a match, CHECK IF EVENT copies the matched event into the client-supplied event structure and returns true. The event is removed from the queue. All earlier events in the queue are not discarded.

If the predicate procedure finds no match, CHECK IF EVENT returns false and flushes the output buffer. CHECK IF EVENT does not block.

The event data structure is shown in Section 4.1.

## Event Routines

### CHECK MASK EVENT

---

## CHECK MASK EVENT

Removes the next matching event from the queue and immediately returns a value to indicate the status.

---

**VAX FORMAT**     *present\_return = X\$CHECK\_MASK\_EVENT*  
                          (*display, event\_mask, event\_return*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
present_return	Boolean	longword	write	value
display	identifier	uns longword	read	reference
event_mask	mask_longword	longword	read	reference
event_return	record	x\$event	write	reference

---

---

**MIT C FORMAT**     *present\_return = XCheckMaskEvent*  
                          (*display, event\_mask, event\_return*)

**argument  
information**

```
Bool XCheckMaskEvent (display, event_mask, event_return)
    Display *display;
    long event_mask;
    XEvent *event_return;
```

---

**RETURNS**     *present\_return*  
A Boolean value that specifies whether the routine completed successfully. When **present\_return** is false, no matching events were found. When **present\_return** is true, a matching event was found.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

*event\_mask*  
A bitmask that specifies the event types for which you want to remove an event. The mask is an inclusive OR of one or more of the event mask elements described in Table 4-2.

*event\_return*  
A pointer to an event structure to which the matching event is returned. CHECK MASK EVENT copies the matched event's associated data structure into this client-supplied data structure. The event data structure is shown in Section 4.1.

---

**DESCRIPTION**

CHECK MASK EVENT searches the event queue, and the server connection, for the event that matches the specified mask. If it finds a match, CHECK MASK EVENT removes the event, copies it into an event structure supplied by the caller and returns true. Earlier events in the queue are not discarded.

If no matching event has been queued, CHECK MASK EVENT flushes the client's output buffer and returns false.

Note that CHECK MASK EVENT differs from MASK EVENT in that CHECK MASK EVENT returns immediately while MASK EVENT blocks until a match is found. CHECK MASK EVENT also returns a Boolean value indicating if the event was returned.

The event data structure is shown in Section 4.1.

## Event Routines

### CHECK TYPED EVENT

---

## CHECK TYPED EVENT

Returns and removes the next event in the event queue that matches the event type.

---

**VAX FORMAT**     *present\_return = X\$CHECK\_TYPED\_EVENT*  
                          (*display, event\_type, event\_return*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
present_return	Boolean	longword	write	value
display	identifier	uns longword	read	reference
event_type	longword	longword	read	reference
event_return	record	x\$event	write	reference

---

---

**MIT C FORMAT**     *present\_return = XCheckTypedEvent*  
                          (*display, event\_type, event\_return*)

**argument  
information**

```
Bool XCheckTypedEvent(display, event_type, event_return)
    Display *display;
    int event_type;
    XEvent *event_return;
```

---

**RETURNS**

***present\_return***

A Boolean value that specifies whether the routine completed successfully. When the value of **present\_return** is false, no matching events were found. When the value of **present\_return** is true, a matching event was found.

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***event\_type***

The event type to be compared.

***event\_return***

A pointer to an event structure to which the matching event is returned. CHECK TYPED EVENT copies the matched event's associated structure into this client-supplied structure.

---

**DESCRIPTION**

CHECK TYPED EVENT searches the event queue for the first event that matches the specified event type. If CHECK TYPED EVENT does not find a match, Xlib searches the available events for the connection that have yet to be queued. If CHECK TYPED EVENT finds a match, it removes the event, copies the associated event structure to the event structure specified by **event\_return**, and returns true. Earlier events in the queue are not discarded.

If no match is found, CHECK TYPED EVENT flushes the output buffer and returns false.



---

## Event Routines

### CHECK TYPED WINDOW EVENT

#### ***event\_return***

A pointer to an event structure to which the matching event is returned. CHECK TYPED WINDOW EVENT returns the matched event's associated structure into this client-supplied structure.

---

#### **DESCRIPTION**

CHECK TYPED WINDOW EVENT searches the event queue for the events associated with the specified type and window. If CHECK TYPED WINDOW EVENT does not find a match, Xlib searches the available events for the connection that have yet to be queued. If it finds a match, CHECK TYPED WINDOW EVENT removes the event from the queue, copies it into the event data structure specified by ***event\_return***, and returns true. Earlier events in the queue are not discarded. If no match is found, CHECK TYPED WINDOW EVENT flushes the output buffer and returns false.



## Event Routines

### CHECK WINDOW EVENT

---

## CHECK WINDOW EVENT

Removes the next matching event from the queue and indicates status.

---

**VAX FORMAT**     *present\_return = X\$CHECK\_WINDOW\_EVENT*  
                          (*display, window\_id, event\_mask, event\_return*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
present_return	Boolean	longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
event_mask	mask_longword	uns longword	read	reference
event_return	record	x\$event	write	reference

---

---

**MIT C FORMAT**     *present\_return = XCheckWindowEvent*  
                          (*display, window\_id, event\_mask, event\_return*)

#### argument information

```
Bool XCheckWindowEvent(display, window_id, event_mask,  
                        event_return)  
Display *display;  
Window window_id;  
long event_mask;  
XEvent *event_return;
```

---

### RETURNS

#### *present\_return*

A Boolean value that specifies whether the routine completed successfully. When the value of **present\_return** is false, no matching events were found. When the value of **present\_return** is true, a matching event was found.

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.

#### *window\_id*

The identifier of the window for which you want to remove the event from the event queue.

## Event Routines

### CHECK WINDOW EVENT

#### *event\_mask*

A bitmask that specifies the events types for which you want to remove an event. This mask is an inclusive OR of one or more of the event mask elements described in Table 4-2.

#### *event\_return*

A pointer to an event structure to which the matching event is returned. CHECK WINDOW EVENT copies the matched event's associated structure to this client-supplied structure. The event data structure is shown in Section 4.1.

---

#### DESCRIPTION

CHECK WINDOW EVENT searches the event queue for the next event that matches both **window\_id** and **event\_mask**. If CHECK WINDOW EVENT does not find a match, Xlib searches the available events for the connections that have yet to be queued.

If it finds a match, CHECK WINDOW EVENT removes the event, copies it into the event structure specified in **event\_return**, and returns true. Earlier events on the queue are not discarded. If no matching event is found, CHECK WINDOW EVENT flushes the output buffer and returns false.

Note that CHECK WINDOW EVENT differs from WINDOW EVENT in that CHECK WINDOW EVENT returns immediately while WINDOW EVENT blocks until a match is found.

The event data structure is shown in Section 4.1.

## Event Routines

### EVENTS QUEUED

---

## EVENTS QUEUED

Checks the number of events in the event queue.

---

**VAX FORMAT**     *count = X\$EVENTS\_QUEUED (display, mode)*

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
count	longword	longword	write	value
display	identifier	uns longword	read	reference
mode	longword	longword	read	reference

---

---

**MIT C FORMAT**     *count = XEventsQueued (display, mode)*

---

**argument  
information**

```
int XEventQueued(display, mode)
    Display *display;
    int mode;
```

---

**RETURNS**     *count*  
The number of events in the event queue.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

*mode*  
The mode by which you want to return the number of queued events. The predefined values for **mode** are as follows:

## Event Routines

### EVENTS QUEUED

VAX	MIT C	Description
X\$C_QUEUED_ALREADY	QueuedAlready	Returns the number of events already in the queue.
X\$C_QUEUED_AFTER_FLUSH	QueuedAfterFlush	Returns the number of events already in the queue, if it is nonzero. If there are no events in the queue, it flushes the output buffer, attempts to read more events for the applications's connection, and returns the number read.
X\$C_QUEUED_AFTER_READING	QueuedAfterReading	Returns the number of events already in the queue, if it is nonzero. If there are no events in the queue, it attempts to read more events for the applications's connection, without flushing the output buffer, and returns the number read.

#### DESCRIPTION

EVENTS QUEUED checks the number of events in the event queue, including those that have yet to be queued. Xlib maintains an event queue for each connection. The transport buffers events on a per-connection basis and delivers them to the Xlib event queues. An event may wait in a transport buffer until it is added to the appropriate Xlib event queues. It is therefore possible for an event to be available to a connection even though it has not yet been queued in the Xlib event queue.

EVENTS QUEUED always returns immediately without I/O if there are events in the queue. EVENTS QUEUED with a mode of Queued After Flush is identical to PENDING. EVENTS QUEUED with a mode of Queued Already is identical to the Q LENGTH display routine.

## Event Routines

### FLUSH

---

## FLUSH

Flushes the Xlib output buffer for a client connection.

---

### VAX FORMAT **X\$FLUSH** (*display*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference

---

---

### MIT C FORMAT **XFlush** (*display*)

#### argument information

---

```
XFlush(display)
Display *display;
```

---

### ARGUMENT

#### ***display***

The display information originally returned by OPEN DISPLAY.

---

### DESCRIPTION

FLUSH flushes all output requests that have been buffered in the client output buffer but have not yet been sent to the server. Because flushing is done as needed, the next time the client calls PENDING, NEXT EVENT, or WINDOW EVENT, most client programs do not need to call the FLUSH routine.

---

## GET ERROR DATABASE TEXT

Returns error messages from the error database.

---

**VAX FORMAT**     **X\$GET\_ERROR\_DATABASE\_TEXT**  
*(display, appl\_name, message\_name,  
default\_message\_name, buff\_return [,length])*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
appl_name	char string	character string	read	descriptor
message_name	char string	character string	read	descriptor
default_message_ name	char string	character string	read	descriptor
buff_return	char string	character string	write	descriptor
length	word	uns word	write	reference

---

---

**MIT C FORMAT**     **XGetErrorDatabaseText**  
*(display, appl\_name, message\_name,  
default\_message\_name, buff\_return, length)*

**argument  
information**

```
XGetErrorDatabaseText(display, appl_name, message_name,  
                      default_message_name, buff_return,  
                      length)  
  
Display *display;  
char *appl_name, *message_name;  
char *default_message_name;  
char *buff_return;  
int length;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***appl\_name***

The name of the application for which you want to obtain error messages.

## Event Routines

### GET ERROR DATABASE TEXT

#### *message\_name*

The type of the error message that you want. Xlib uses the following message types to report errors:

VAX Binding	MIT C Binding	Description
X\$C_PROTO_ERROR	XProtoError	The protocol error number is used as a string for <b>message_name</b> .
X\$C_XLIB_MESSAGE	XlibMessage	These are the message strings that are used internally by Xlib.
X\$C_REQUEST	XRequest	The major protocol request number is used for the message argument. If no string is found in the error database, the default string is returned to <b>buff_return</b> .

#### VAX only

The **message\_name** argument is the address of a character string descriptor that points to the string.

#### MIT C only

The **message\_name** argument is a pointer to the null-terminated character string.

#### *default\_message\_name*

The default error message if no message is found in the database.

#### *buff\_return*

The error message.

#### *length*

The size of the buffer that is passed in **buff\_return**.

#### VAX only

This argument is optional.

---

## DESCRIPTION

GET ERROR DATABASE TEXT returns a message (or the default message) from the error message database. Xlib uses this function internally to look up error messages.

---

## GET ERROR TEXT

Returns a text string that describes the error code that you specify.

---

**VAX FORMAT**    **X\$GET\_ERROR\_TEXT**  
                   (*display, code, buff\_return [, len\_return]*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
code	longword	longword	read	reference
buff_return	char string	character string	write	descriptor
len_return	longword	uns longword	write	reference

---



---

**MIT C FORMAT**    **XGetErrorText**  
                   (*display, code, buff\_return, length*)

**argument  
information**

```
XGetErrorText(display, code, buff_return, length)
    Display *display;
    int code;
    char *buff_return;
    int length;
```

---

**ARGUMENTS**    ***display***  
 The display information originally returned by OPEN DISPLAY.

***code***  
 The error code for which you want to obtain a textual description.



## Event Routines

### GET ERROR TEXT

#### ***buff\_return***

A pointer to the buffer to which GET ERROR TEXT returns the description.

#### **VAX only**

The **buff\_return** argument is the address of a character string descriptor that points to the string.

#### **MIT C only**

The **buff\_return** argument is a pointer to a client-allocated buffer.

#### ***len\_return (VAX only)***

The size of the text string that GET ERROR TEXT returns. This argument is optional in the VAX binding.

#### ***length (MIT C only)***

The size of the **buff\_return** buffer.

---

## DESCRIPTION

GET ERROR TEXT returns a textual description of the error code that you specify. You should use this routine to obtain an error description because extensions to Xlib can define their own error codes and strings.

---

## GET MOTION EVENTS

Returns pointing device motion events for a window based on the time coordinates that you supply.

---

**VAX FORMAT**     *status\_return = X\$GET\_MOTION\_EVENTS*  
                           *(display, window\_id, start, stop, num\_events\_return*  
                           *[,time\_return] [,time\_size] [,time\_buff\_return])*

**argument information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
start	longword	uns longword	read	reference
stop	longword	uns longword	read	reference
num_events_return	longword	longword	write	reference
time_return	address	uns longword	write	reference
time_size	longword	longword	read	reference
time_buff_return	array	uns longword	write	reference

---



---

**MIT C FORMAT**     *timecoord\_return = XGetMotionEvents*  
                           *(display, window\_id, start, stop, num\_events\_return)*

**argument information**

```
XTimeCoord *XGetMotionEvents(display, window_id, start, stop,
                               num_events_return)
Display *display;
Window window_id;
Time start, stop;
int *num_events_return;
```

---

**RETURNS**

***status\_return (VAX only)***  
 Possible status values returned by the VAX binding are as follows:

---

Value	Description
X\$_ERRORREPLY	An error was received from the server.

---

## Event Routines

### GET MOTION EVENTS

---

Value	Description
X\$_TRUNCATED	The user buffer specified in <b>time_buff_return</b> was not large enough.
SS\$_NORMAL	The routine completed successfully.

---

#### ***timecoord\_return (MIT C only)***

All the events in the motion history buffer that fall within the specified starting and stopping times (inclusive), and that have coordinates that lie within the specified window (including borders) at its current placement, are returned.

If the starting time is later than the stopping time, or if the starting time is in the future, no events are returned.

The time coordinate data structure is shown in Section 4.4.

---

## ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***window\_id***

The identifier of the window for which you want to receive motion events.

#### ***start***

The beginning of the time interval, in milliseconds, for which the events are returned from the motion history buffer. You can pass a time-stamping, expressed in milliseconds, or the predefined values X\$\_CURRENT\_TIME or CurrentTime. If the stopping time is in the future, it is equivalent to specifying Current Time.

#### ***stop***

The end of the time interval, in milliseconds, for which the events are returned from the motion history buffer. You can pass a time-stamping, expressed in milliseconds, or the predefined values X\$\_CURRENT\_TIME or CurrentTime. If the stopping time is in the future, it is equivalent to specifying Current Time.

#### ***num\_events\_return***

A pointer to the number of events returned from the motion history buffer.

#### ***time\_return (VAX only)***

The address of the time coordinate data structure. This argument is optional. You can specify either **time\_return** to receive the address of the time coordinate data structure, or the **time\_size** and **time\_buff\_return** arguments to receive an array of time coordinates in a buffer.

#### ***time\_size (VAX only)***

The length of the time coordinate buffer. This argument is optional.

#### ***time\_buff\_return (VAX only)***

A pointer to an array of time coordinates. The length of the array is specified by **time\_size**. This argument is optional.

---

**DESCRIPTION**

GET MOTION EVENTS returns a pointer to all of the events in the pointing device motion history queue that fall between the starting and stopping times (inclusive) that you specify and that have coordinates that lie within (including borders) the specified window at its present placement.

If the starting time is later than the stopping time, or if the starting time is in the future, no events are returned. If the stopping time is in the future, it is equivalent to specifying the predefined value Current Time, which is reserved for use in requests to represent the current server time.

GET MOTION EVENTS does not request notification of pointing device motion history events from the Xlib input event queue; it receives this information in the form of a reply directly from the server.

The time coordinate data structure is shown in Section 4.4.

The *time* member of the time coordinate data structure is set to the time in milliseconds. The *x* and *y* members are set to the coordinates of the pointer and are reported relative to the origin of the specified window.

Clients should use FREE to free the data that is returned from this call.

**Note:** Some X11 server implementations maintain a history of pointer motion by storing the pointer position at each pointer hardware interrupt in a motion history buffer. GET MOTION EVENTS returns a pointer to the events in the motion history buffer. The VMS DECwindows server does not support the motion history buffer.

---

**X ERRORS**

VAX	MIT C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Event Routines

### IF EVENT

---

## IF EVENT

Checks the input event queue for the event that you specify in the predicate procedure, and removes the event from the queue if the events match.

---

### VAX FORMAT

#### **X\$IF\_EVENT**

*(display, event\_return, predicate, arg)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_return	record	x\$event	write	reference
predicate	procedure	proc entry mask	read	reference
arg	longword	uns longword	read	value

---

---

### MIT C FORMAT

#### **XIfEvent**

*(display, event\_return, predicate, arg)*

#### argument information

```
XIfEvent(display, event_return, predicate, arg)
    Display *display;
    XEvent *event_return;
    Bool (*predicate)();
    char *arg;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***event\_return***

A pointer to the event structure to which the matching event is returned. IF EVENT copies the matched event's associated structure into this client-supplied structure. The event data structure is shown in Section 4.1.

#### ***predicate***

A pointer to a user-supplied procedure that determines if the next event in the event queue matches the event you specify in the **arg** argument. When true, **predicate** returns a nonzero value. When false, **predicate** returns zero. You call the predicate procedure once for each event in the queue until it returns a nonzero value.

For more information about the predicate procedure see Section 4.3.

#### ***arg***

The user-specified arguments that are passed to the predicate procedure.

---

**DESCRIPTION**

IF EVENT checks the input event queue for the event that you specify in the predicate procedure, and removes the event from the queue if the events match. IF EVENT completes only when the predicate procedure returns true for an event. You supply a predicate procedure that determines if the next event in the event queue matches the event you specify in **arg**. The predicate procedure is called each time an event is added to the queue.

If the predicate procedure returns true, IF EVENT removes the event from the queue and, when it returns, copies the event data structure into the data structure specified in **event\_return**.

If IF EVENT does not find a match, it flushes the output buffer and waits for an event. The CHECK IF EVENT routine differs in that it returns false and does not block if it does not find a match.

The event data structure is shown in Section 4.1.

## Event Routines

### MASK EVENT

---

## MASK EVENT

Removes and copies the next matching event from the queue or blocks until a matching event is received.

---

**VAX FORMAT**    **X\$MASK\_EVENT**  
*(display, event\_mask, event\_return)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_mask	mask_longword	longword	read	reference
event_return	record	x\$event	write	reference

---

---

**MIT C FORMAT**    **XMaskEvent**  
*(display, event\_mask, event\_return)*

**argument  
information**

```
XMaskEvent(display, event_mask, event_return)
    Display *display;
    long event_mask;
    XEvent *event_return;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***event\_mask***  
A bitmask that specifies the events types for which you want to remove an event. The event mask is described in Table 4-2.

***event\_return***  
A pointer to the event data structure to which the matching event is returned. MASK EVENT copies the matched event's associated structure to this client-supplied structure. The event data structure is shown in Section 4.1.

---

**DESCRIPTION**    MASK EVENT removes the next event in the queue that matches the mask that you specified in **event\_mask**. MASK EVENT copies the event into the event structure specified in **event\_return**. Earlier events in the queue are not discarded.

If no matching event is found, MASK EVENT flushes the client's output buffer and then blocks until one is received.

---

## Event Routines

### MASK EVENT

The event data structure is shown in Section 4.1.

Note that MASK EVENT differs from CHECK MASK EVENT in that CHECK MASK EVENT returns immediately while MASK EVENT blocks until a match is found. CHECK MASK EVENT also returns a Boolean value indicating if the event was returned.



## Event Routines

### NEXT EVENT

---

## NEXT EVENT

Gets the next event and removes it from the queue. NEXT EVENT can flush the client's Xlib output buffer.

---

### VAX FORMAT **X\$NEXT\_EVENT** (*display, event\_return*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_return	record	x\$event	write	reference

---

---

### MIT C FORMAT **XNextEvent** (*display, event\_return*)

#### argument information

```
XNextEvent(display, event_return)
Display *display;
XEvent *event_return;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***event\_return***

A pointer to the event structure to which the next event is returned. NEXT EVENT copies the next event's associated structure to this client-supplied storage. The event data structure is shown in Section 4.1.

---

## DESCRIPTION

NEXT EVENT copies the next event's associated structure to the event structure specified in **event\_return** and removes the event from the top of the event queue. For example, if a Create Notify event is at the top of the queue, NEXT EVENT removes it and then copies the create window event structure into the event structure specified in **event\_return**.

If the event queue is empty, NEXT EVENT flushes the client's output buffer and blocks until an event is received.

NEXT EVENT is similar to PEEK EVENT, except that NEXT EVENT removes the event from the queue and PEEK EVENT does not.

For more information about the event data structure, see Section 4.1.

## PEEK EVENT

Looks at and copies an event from the event queue.

**VAX FORMAT**    **X\$PEEK\_EVENT**    (*display, event\_return*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_return	record	x\$event	write	reference

**MIT C FORMAT**    **XPeekEvent**    (*display, event\_return*)

**argument  
information**

```
XPeekEvent(display, event_return)
    Display *display;
    XEvent *event_return;
```

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***event\_return***

A pointer to the event structure to which the next event is returned. PEEK EVENT copies the event's associated structure to this structure. The event data structure is shown in Section 4.1.

### DESCRIPTION

PEEK EVENT looks at the event at the top of the event queue and copies the associated structure to the event structure specified in **event\_return**. PEEK EVENT does not remove the event from the event queue. For example, if a Create Notify event is at the top of the queue, PEEK EVENT copies the create window event structure into **event\_return**.

If the event queue is empty, PEEK EVENT flushes the client's output buffer then blocks until an event is received.

PEEK EVENT is similar to NEXT EVENT, except that NEXT EVENT removes the event from the queue and PEEK EVENT does not.

The event data structure is shown in Section 4.1.

## Event Routines

### PEEK IF EVENT

---

## PEEK IF EVENT

Checks the event queue for the event that you specify in the predicate procedure; does not remove the event from the event queue.

---

**VAX FORMAT**    **X\$PEEK\_IF\_EVENT**  
(*display, event\_return, predicate, arg*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event_return	record	x\$event	write	reference
predicate	procedure	proc entry mask	read	reference
arg	longword	uns longword	read	value

---

---

**MIT C FORMAT**    **XPeekIfEvent**  
(*display, event\_return, predicate, arg*)

**argument  
information**

```
XPeekIfEvent(display, event_return, predicate, arg)
Display *display;
XEvent *event_return;
Bool (*predicate)();
char *arg;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***event\_return***

A pointer to an event structure to which the matched event is returned. PEEK IF EVENT copies the matched event's associated structure into this client-supplied structure. The event data structure is shown in Section 4.1.

### ***predicate***

A pointer to the user-supplied procedure that determines if the next event in the event queue matches the event you specify in the **arg** argument. When true, **predicate** returns a nonzero value. When false, **predicate** returns zero. Xlib calls the predicate procedure once for each event in the queue until it returns a nonzero value.

For more information on the predicate procedure, see Section 4.3.

### ***arg***

The user-specified arguments that are passed to the predicate procedure.

---

**DESCRIPTION**

PEEK IF EVENT checks the event queue for the event that you specify in the **arg** argument. You supply a predicate procedure that determines if the next event in the event queue matches the event you specify in the **arg** argument. This predicate procedure is called each time that an event is added to the queue. PEEK IF EVENT returns the event in the event argument only when the specified predicate procedure returns true.

When it returns, PEEK IF EVENT does not remove the event from the event queue, but instead copies it to the event structure specified in **event\_return**.

If there is no match, PEEK IF EVENT flushes the Xlib output buffer and blocks until a matching event is received.

The event data structure is shown in Section 4.1.

## Event Routines

### PENDING

---

## PENDING

Returns the number of pending input events.

---

**VAX FORMAT**     *count\_return = X\$PENDING (display)*

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
count_return	longword	longword	write	value
display	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *count\_return = XPending (display)*

---

**argument  
information**

```
int XPending(display)
    Display *display;
```

---

**RETURNS**     *count\_return*

The number of events that have been received by Xlib but are not yet removed from the event queue.

---

**ARGUMENTS**     *display*

The display information originally returned by OPEN DISPLAY.

---

**DESCRIPTION**

PENDING returns the number of events that have been received by Xlib but are not yet removed from the event queue.

PENDING is identical to EVENTS QUEUED with the mode Queued After Flush.

## PUT BACK EVENT

Pushes an event back to the top of the client's event queue.

### VAX FORMAT **X\$PUT\_BACK\_EVENT** (*display, event*)

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
event	record	x\$event	read	reference

### MIT C FORMAT **XPutBackEvent** (*display, event*)

#### argument information

```
XPutBackEvent (display, event)
    Display *display;
    XEvent *event;
```

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***event***

A pointer to the event structure from which you want to return the event to the event queue. The event data structure is shown in Section 4.1.

### DESCRIPTION

PUT BACK EVENT copies an event to the top of the event queue. If you use a routine such as IF EVENT to remove an event from the event queue, you can use PUT BACK EVENT to return the event to the queue.

There is no limit to the number of times that a client can successively call PUT BACK EVENT.

The event data structure is shown in Section 4.1.

## Event Routines

### SELECT ASYNC EVENT

---

## SELECT ASYNC EVENT

Specifies an action routine and argument to be called when an event occurs.

---

**VAX FORMAT**    **X\$SELECT\_ASYNC\_EVENT**  
*(display, window\_id, event\_type, ast\_routine,  
ast\_userarg)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
event_type	longword	uns longword	read	reference
ast_routine	procedure	proc entry mask	read	reference
ast_userarg	longword	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSelectAsyncEvent**  
*(display, window\_id, event\_type, ast\_routine,  
ast\_userarg)*

**argument  
information**

---

```
XSelectAsyncEvent(display, window_id, event_type,  
                  ast_routine, ast_userarg)  
Display *display;  
Window window_id;  
unsigned long event_type;  
int (*ast_routine)();  
unsigned long ast_userarg;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***window\_id***

The identifier of the window for which you want to select asynchronous events.

### ***event\_type***

The type of event for which you want to select asynchronous events.

### ***ast\_routine***

The particular Asynchronous System Trap (AST) action routine to use when notifying the client that the specified event has occurred.

## Event Routines

### SELECT ASYNC EVENT

#### *ast\_userarg*

The user-specified argument to use when notifying the client that the specified event has occurred.

---

#### DESCRIPTION

Before calling `SELECT INPUT` to specify interest in a particular set of events for a window, clients can call `SELECT ASYNC EVENT` to specify an action routine and argument to be called when the specified event occurs.

Xlib uses the client's **`ast_routine`** and **`ast_userarg`** information to deliver an AST whenever it places the specified event on the event queue. The AST acts only as an event notification mechanism; the application uses the standard Xlib event routines to actually retrieve and process the event from the event queue.

Clients can call `SELECT ASYNC EVENT` multiple times to specify different routine and argument pairs for different events for a window. The last call always takes precedence. If called with **`ast_routine`** equal to zero, asynchronous notification is disabled, but the current selection for the specified event is unaffected.

Notification ASTs are queued in the same order as events are placed in the event queue by Xlib. Therefore, clients can assume that they receive notification ASTs in the same order that they find events in the queue.

`SELECT ASYNC EVENT` is similar to `SELECT ASYNC INPUT` except that `SELECT ASYNC EVENT` specifies one event type and `SELECT ASYNC INPUT` specifies an event mask.



## Event Routines

### SELECT ASYNC INPUT

---

## SELECT ASYNC INPUT

Specifies an action routine and arguments to be called when some subset of events occurs.

---

**VAX FORMAT**    **X\$SELECT\_ASYNC\_INPUT**  
(*display, window\_id, event\_mask, ast\_routine,*  
*ast\_userarg*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
event_mask	longword	uns longword	read	reference
ast_routine	procedure	proc entry mask	read	reference
ast_userarg	longword	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSelectAsyncInput**  
(*display, window\_id, event\_mask, ast\_routine,*  
*ast\_userarg*)

#### argument information

---

```
XSelectAsyncInput (display, window_id, event_mask,  
                  ast_routine, ast_userarg)  
Display *display;  
Window window_id;  
unsigned long event_mask;  
int (*ast_routine) ();  
unsigned long ast_userarg;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***window\_id***

The identifier of the window for which you want to select asynchronous input.

### ***event\_mask***

A bitmask that specifies the event types for which you want to remove an event. The mask is an inclusive OR of one or more of the event mask elements described in Table 4-2.

***ast\_routine***

The particular Asynchronous System Trap (AST) action routine to use when notifying the client that one of the specified events has occurred.

***ast\_userarg***

The user-specified parameter to use when notifying the client that one of the specified events has occurred.

---

**DESCRIPTION**

Before calling `SELECT INPUT` to specify interest in a particular set of events for a window, clients can call `SELECT ASYNC INPUT` to specify action routines and arguments to be called when some subset of those events occurs.

Xlib uses the client's ***ast\_routine*** and ***ast\_userarg*** information to deliver an AST whenever it places one of the specified events on its event queue. The ***ast\_routine*** and ***ast\_userarg*** arguments allow the client to specify the particular action routine and parameter pair to use when notifying the client that one of the specified events has occurred. The AST acts only as an event notification mechanism; the application uses the standard Xlib event routines to actually retrieve and process the event from the event queue.

Clients can call `SELECT ASYNC INPUT` multiple times to specify different routine and parameter pairs for different sets of events for a window. The last call always takes precedence. If called with ***ast\_routine*** equal to zero, asynchronous notification is disabled, but the current selection for the specified events is unaffected.

Notification ASTs are queued in the same order as events are placed in the event queue by Xlib. Therefore, clients can assume that they receive notification ASTs in the same order that they find events in the queue.

`SELECT ASYNC INPUT` is similar to `SELECT ASYNC EVENT` except that `SELECT ASYNC EVENT` specifies one event type and `SELECT ASYNC INPUT` specifies an event mask.

## Event Routines

### SELECT INPUT

---

## SELECT INPUT

Selects the events types to send to a window.

---

**VAX FORMAT**    **X\$SELECT\_INPUT**  
(*display, window\_id, event\_mask*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
event_mask	mask_longword	longword	read	reference

---

---

**MIT C FORMAT**    **XSelectInput**  
(*display, window\_id, event\_mask*)

**argument  
information**

```
XSelectInput (display, window_id, event_mask)
    Display *display;
    Window window_id;
    long event_mask;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***window\_id***  
The identifier of the window for which you want to select input events. Client applications that are interested in an event for a particular window pass that window identifier.

***event\_mask***  
A bitmask that specifies the event types for which you want the window to receive notification. This mask is the inclusive OR of one or more event mask elements described in Table 4-2.

---

**DESCRIPTION**    SELECT INPUT allows you to specify the input events for which a window receives notification. If a window is not interested in an event, the event propagates until an ancestor of the window expresses interest, or until the event is explicitly discarded. Initially, the server does not report any of these events.

## Event Routines

### SELECT INPUT

There are two ways to select the events that you want to report to your client application. The first way is to set the event mask members of the select window attributes data structure when you call `CREATE WINDOW` and `CHANGE WINDOW ATTRIBUTES`. The second way is to use `SELECT INPUT`.

Setting the *event\_mask* attribute of a window overrides any previous call for the same window, but not for other clients. Multiple clients can select events on the same window because their event masks are disjoint. When the server generates an event, it reports the event to all interested clients. Multiple clients can select the same events on the same window with the following restrictions:

- Only one client at a time can select Circulate Request, Configure Request, or Map Request events, which are associated with the Substructure Redirect mask.
- Only one client at a time can select a Resize Request event, which is associated with the Resize Redirect mask.
- Only one client at a time can select a Button Press event, which is associated with the Button Press mask.

The server reports the events to all interested clients.

---

## X ERRORS

VAX	MIT C	Description
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## Event Routines

### SEND EVENT

---

## SEND EVENT

Sends an event to a window without the window requesting the event.

---

**VAX FORMAT**     *status\_return = X\$SEND\_EVENT*  
                          (*display, window\_id, propagate, event\_mask, event*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	cond_value	uns longword	write	value
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
propagate	longword	longword	read	reference
event_mask	mask_longword	longword	read	reference
event	record	x\$event	read	reference

---

---

**MIT C FORMAT**     *status\_return = XSendEvent*  
                          (*display, window\_id, propagate, event\_mask, event*)

**argument  
information**

```
Status XSendEvent(display, window_id, propagate, event_mask,  
                  event)  
    Display *display;  
    Window window_id;  
    Bool propagate;  
    long event_mask;  
    XEvent *event;
```

---

**RETURNS**     *status\_return*  
Returns false if the conversion to the wire protocol failed. Returns true if the conversion to the wire protocol was successful.

---

**ARGUMENTS**     *display*  
The display information originally returned by OPEN DISPLAY.

*window\_id*  
The identifier of the window to which you want to send the event. This window is called the destination window. You can pass the window identifier or the constants Pointer Window or Input Focus, which are defined as follows:

VAX Binding	MIT C Binding	Description
XSC_POINTER_WINDOW	PointerWindow	The destination window is the window that contains the pointer.
X\$C_INPUT_FOCUS	InputFocus	The destination window is the window that has the input focus, regardless of whether the window contains the pointer.

### ***propagate***

A Boolean value that determines whether or not to propagate the event.

If **event\_mask** does not specify any event types, the event is sent to the client that created the destination window. If that client no longer exists, no event is sent.

When true, and no client has selected the destination window for any of the event types specified in the event mask, the destination is replaced with its closest ancestors for which some client has selected a matching event type. This condition is true only when intervening windows propagate that event type.

If no such window exists, or if the window is an ancestor of the focus window and input focus was originally specified as the destination, then the event is not sent to any clients. Otherwise, the event is reported to every client that selects any of the types specified in the event mask on the final destination.

When false, the event is sent to every client selecting on destination any of the event types specified in **event\_mask**.

### ***event\_mask***

A bitmask that specifies the events types for which you want the window to receive notification. This mask is the inclusive OR of one or more of the valid event mask elements described in Table 4-2.

### ***event***

A pointer to the event that is to be sent.

---

## DESCRIPTION

SEND EVENT identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs.

The event codes defined by the **event\_mask** argument must be one of the core events described in the event structure, or one of the events defined by a loaded extension, so that the server can byte swap the contents as necessary. The contents of the events are otherwise unaltered and unchecked by the server except to force the *send event* member of the forwarded event structure to true and to set the serial number in the event. Active grabs are ignored for this request.

The event data structure is shown in Section 4.1.

## Event Routines

### SEND EVENT

---

#### X ERRORS

VAX	MIT C	Description
X\$_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless you specify a specific range for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## SET AFTER FUNCTION

Specifies the synchronization handler to call before returning from an Xlib protocol routine.

**VAX FORMAT**     **X\$SET\_AFTER\_FUNCTION**  
                           (*display, func\_addr, prev\_func\_addr\_return*)

**argument information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
func_addr	procedure	proc entry mask	read	reference
prev_func_addr_return	procedure	proc entry mask	write	reference

**MIT C FORMAT**     *prev\_func\_return = XSetAfterFunction*  
                           (*display, func\_addr*)

**argument information**

```
int (*XSetAfterFunction(display, func_addr))()
    Display *display;
    int (*func_addr)();
```

**RETURNS**             *prev\_func\_return (MIT C only)*  
                           The previous after function is returned.

**ARGUMENTS**

***display***  
 The display information originally returned by OPEN DISPLAY.

***func\_addr***  
 The function to be called after an Xlib function that generates a protocol request completes its work. Only a display pointer is passed to the function.

***prev\_func\_addr\_return (VAX only)***  
 The previous synchronization routine is returned.



## Event Routines

### SET AFTER FUNCTION

---

**DESCRIPTION** SET AFTER FUNCTION specifies a synchronization routine to call before returning from an Xlib routine that generates a protocol request. All Xlib routines that generate protocol requests will then call this “after function” synchronization routine before returning. You use SET AFTER FUNCTION before a call to SYNCHRONIZE to specify a synchronization routine.

The X server does not automatically notify Xlib when the server has received and processed an output request. However, when debugging your client program, it is sometimes helpful to know that a routine has completed successfully, or generated an error, before the next output request is sent to the server. To do this, you can use SET AFTER FUNCTION to follow each Xlib output request with a call to a synchronization routine that generates a return. When your client program receives the return, you know that the previous routine completed successfully.

The synchronization routine can be any Xlib or user-written routine that generates a return. For example, you can specify a routine that generates program statistics. When your client program receives the values returned by the statistics routine, you can be sure that the previous routine completed successfully or caused an error.

The synchronization routine is called with one argument, a pointer to the display.

## SET ERROR HANDLER

Specifies a user-written routine to handle nonfatal errors.

### VAX FORMAT `X$SET_ERROR_HANDLER (handler)`

#### argument information

Argument	Usage	Data Type	Access	Mechanism
handler	procedure	proc entry mask	read	reference

### MIT C FORMAT `XSetErrorHandler (handler)`

#### argument information

```
XSetErrorHandler(handler)
    int (*handler)(Display*, XErrorEvent*);
```

### ARGUMENTS *handler*

A user-written routine that handles nonfatal errors. The **handler** argument is the address of the entry mask for the routine. The routine returns an integer value. SET ERROR HANDLER passes the following arguments to the routine:

- **display**—Pointer to the display structure associated with the display.
- **error\_event**—Pointer to the error event. The error must be of type **X\$ERROR\_EVENT** for the VAX binding and **XErrorEvent** for the MIT C binding.

### DESCRIPTION

SET ERROR HANDLER specifies the user-written error handler to be called whenever an error event is received. These errors are assumed to be nonfatal, and it is acceptable for the error handler to return control to the client program. However, the error handler should not call any routines (directly or indirectly) that generate protocol requests or look for input events.

## Event Routines

### SET ERROR HANDLER

The error handler is not called for the following conditions:

- On Bad Name errors from OPEN FONT, LOOKUP COLOR, and ALLOC NAMED COLOR protocol requests
- On Bad Font errors from a QUERY FONT protocol request
- On Bad Alloc or Bad Access errors from synchronous protocol requests

These errors are reflected back to the program through the routine's return value. Errors are listed for a routine only if the error results in a call to the error handler.

The error event structure defines the format of errors reported to the default or user-supplied error handlers.

The attributes of the error event structure are as follows:

Member	Description
display	The display from which the event was read.
serial	The total number of protocol requests, starting from one, sent across the transport since the connection was opened. This member reflects the number of the request immediately before the failing call was made.
error_code	The identifying error code of the failing call.
request_code	The operation code for the protocol request that failed.
minor_code	The minor operation code for the protocol request that failed.
resource_id	The resource identifier for which the error occurred.

The error event data structure is shown in Section 4.6.

See also the SET IO ERROR HANDLER routine.

---

## SET IO ERROR HANDLER

Specifies a user-written routine to handle fatal I/O errors.

---

### VAX FORMAT `X$SET_IO_ERROR_HANDLER (handler)`

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
handler	procedure	proc entry mask	read	reference

---

---

### MIT C FORMAT `XSetIOErrorHandler (handler)`

#### argument information

```
XSetIOErrorHandler(handler)  
int (*handler)(Display*);
```

---

### ARGUMENTS *handler*

A user-written routine that handles fatal errors. The handler argument is the address of the entry mask for the routine. SET IO ERROR HANDLER passes a pointer to the display structure to the routine.

---

### DESCRIPTION

SET IO ERROR HANDLER specifies a user-written error handler to be called by Xlib if any type of system-call error, such as losing the connection to the server, occurs. This is assumed to be a fatal condition; the error handler should not return. If the IO error handler does return, the client process exits.

Also see the SET ERROR HANDLER routine.

## Event Routines

### SYNC

---

## SYNC

Flushes the client's Xlib output buffer and waits for all requests to be received and processed by the server.

---

### VAX FORMAT **X\$SYNC** (*display, discard*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
discard	longword	longword	read	reference

---

---

### MIT C FORMAT **XSync** (*display, discard*)

#### argument information

```
XSync(display, discard)
Display *display;
Bool discard;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***discard***

If true, SYNC discards all events on the client event queue, including those events that were on the queue before SYNC was called. If discard is false, SYNC does not discard the events on the queue.

---

### DESCRIPTION

SYNC flushes the client's Xlib output buffer and then waits until all requests have been received and processed by the server. New events are placed on the Xlib event queue by the server. Any errors that are generated must be handled by the error handler; the client's error handling routine is called once for each error received.

Client applications seldom need to call SYNC.

See also the SYNCHRONIZE routine.



## Event Routines

### SYNCHRONIZE

---

#### DESCRIPTION

**SYNCHRONIZE** enables or disables synchronization and returns the after function. The X server does not automatically notify Xlib when the server has received and processed an output request. However, when debugging your client program, it is sometimes helpful to know that a routine has completed successfully, or generated an error, before the next output request is sent to the server.

**SYNCHRONIZE** flushes the client's Xlib output buffer after each Xlib routine then calls a synchronization routine that generates a return. When your client program receives the return, you know that the previous routine completed. If an output request generates an error, the server reports the error back to the client at the time it occurs.

**SYNCHRONIZE** uses the window and session manager routine **GET INPUT FOCUS** as the default synchronization routine. You can use **SET AFTER FUNCTION** to specify another synchronization routine.

Note that when you enable synchronization, all output for the display is synchronized. Graphics output may occur dramatically (30 or more times) slower when synchronization is enabled because of the increased overhead incurred by sending each output request individually and waiting for a return or error condition.

**SYNCHRONIZE** is similar to **SYNC**, except that **SYNCHRONIZE** flushes the output buffer after each Xlib routine and waits for the after function return value to be returned; **SYNC** flushes the output buffer only once and may also clear the event queue.

---

## WINDOW EVENT

Removes the next matching event from the queue for the specified window.

---

**VAX FORMAT**     **X\$WINDOW\_EVENT**  
                           (*display, window\_id, event\_mask, event\_return*)

argument  
information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
event_mask	mask_longword	uns longword	read	reference
event_return	record	x\$event	write	reference

---

**MIT C FORMAT**     **XWindowEvent**  
                           (*display, window\_id, event\_mask, event\_return*)

argument  
information

```
XWindowEvent(display, window_id, event_mask, event_return)
Display *display;
Window window_id;
long event_mask;
XEvent *event_return;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***window\_id***

The identifier of the window for which you want to remove the next matching event.

#### ***event\_mask***

A bitmask that specifies the event types for which you want to remove an event. This mask is the inclusive OR of one or more of the valid event mask elements described in Table 4-2.

#### ***event\_return***

A pointer to the event structure to which the matching event is returned. WINDOW EVENT copies the matched event's associated structure into this client-supplied structure. The event data structure is shown in Section 4.1.



## Event Routines

### WINDOW EVENT

---

#### DESCRIPTION

WINDOW EVENT searches the event queue for an event that matches both the specified window and event mask. If it finds a match, WINDOW EVENT removes the event from the queue and copies it into an event structure supplied in the **event\_return** argument. Other events in the queue are not discarded. If no such event has been queued, WINDOW EVENT flushes the output buffer and blocks until one is received.

Note that WINDOW EVENT differs from CHECK WINDOW EVENT in that CHECK WINDOW EVENT returns immediately while WINDOW EVENT blocks until a match is found.

The event data structure is shown in Section 4.1.

# 5

## Graphics Context Routines

Graphics contexts are data structures that specify several graphics attributes. For example, the graphics routines use the graphics context attributes to define how various operations are performed. For example, when a line is drawn, the graphics routine DRAW LINE uses a GC values data structure to define how wide the line is, how to draw the end points of the line, and which pattern to use to fill the line. For more information about graphics routines, see Chapter 6.

The graphics context routines enable you to perform the following operations:

- Create GC values data structures
- Manipulate the data structures
- Change members within the data structure

For concepts related to graphics context routines and information on how to use graphics context routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 5–1.

**Table 5–1 Graphics Context Routines**

Routine Name	Description
CHANGE GC	Change any attribute by changing the values in the graphics context data structure.
COPY GC	Copies the graphics context values from one graphics context data structure to another graphics context data structure.
CREATE GC	Creates a new graphics context for a specific window. Unless specified, the members of the graphics context are the default values.
FREE GC	Frees the storage associated with the graphics context and its identifier.
GCONTEXT FROM GC	Obtains the graphics context resource identifier for a specified graphics context.
QUERY BEST SIZE	Obtains the optimal size for improved server performance for the cursor, the tile pattern, or the stipple pattern.
QUERY BEST STIPPLE	Obtains the optimal size for improved server performance for a stipple pattern.

(continued on next page)

# Graphics Context Routines

**Table 5–1 (Cont.) Graphics Context Routines**

<b>Routine Name</b>	<b>Description</b>
QUERY BEST TILE	Obtains the optimal size for improved server performance for a tile pattern.
SET ARC MODE	Sets the arc mode to the value specified.
SET BACKGROUND	Sets the background color index to the value specified.
SET CLIP MASK	Changes the pixmap identifier of the clip mask for the specified graphics context.
SET CLIP ORIGIN	Changes the clip origin members in the specified graphics context.
SET CLIP RECTANGLES	Changes the clip origin members in the specified graphics context and specifies rectangular clip areas.
SET DASHES	Changes the values for the dash offset and the dash list.
SET FILL RULE	Sets the fill rule member in the graphics context to the value specified.
SET FILL STYLE	Sets the fill style member in the graphics context to the value specified.
SET FONT	Changes the identifier of the font in the specified graphics context.
SET FOREGROUND	Sets the foreground color index to the value specified.
SET FUNCTION	Sets the function value to the value specified.
SET GRAPHICS EXPOSURES	Changes the graphics exposure.
SET LINE ATTRIBUTES	Changes the line drawing members in the graphics context.
SET PLANE MASK	Sets the plane mask to the value specified.
SET STATE	Changes values for the foreground, background, plane mask, and function members of the graphics context.
SET STIPPLE	Changes the pixmap identifier of the stipple pattern in the specified graphics context.
SET SUBWINDOW MODE	Changes the value for the subwindow mode member in the graphics context.
SET TILE	Changes the pixmap identifier of the tile pattern in the specified graphics context.
SET TS ORIGIN	Changes the x- and y-coordinates of the tile or stipple origin in the graphics context.

## 5.1 The GC Values Data Structure

The GC values data structure stores all the values for each of 23 graphics context members.

## Graphics Context Routines

### 5.1 The GC Values Data Structure

The data structure for the VAX binding is shown in Figure 5–1, and information about members in the data structure is described in Table 5–2.

**Figure 5–1 GC Values Data Structure (VAX Binding)**

x\$I_gcvt_function	0
x\$I_gcvt_plane_mask	4
x\$I_gcvt_foreground	8
x\$I_gcvt_background	12
x\$I_gcvt_line_width	16
x\$I_gcvt_line_style	20
x\$I_gcvt_cap_style	24
x\$I_gcvt_join_style	28
x\$I_gcvt_fill_style	32
x\$I_gcvt_fill_rule	36
x\$I_gcvt_arc_mode	40
x\$I_gcvt_tile	44
x\$I_gcvt_stipple	48
x\$I_gcvt_ts_x_origin	52
x\$I_gcvt_ts_y_origin	56
x\$I_gcvt_font	60
x\$I_gcvt_subwindow_mode	64
x\$I_gcvt_graphics_exposures	68
x\$I_gcvt_clip_x_origin	72
x\$I_gcvt_clip_y_origin	76
x\$I_gcvt_clip_mask	80
x\$I_gcvt_dash_offset	84
x\$b_gcvt_dashes	

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents																																		
X\$L_GCVL_FUNCTION	<p>Defines how the server computes pixel values when the client updates a section of the screen. The following lists available functions:</p> <table border="1"><thead><tr><th>Constant Name</th><th>Description</th></tr></thead><tbody><tr><td>X\$C_GX_CLEAR</td><td>0</td></tr><tr><td>x\$C_GX_AND</td><td>src AND dst</td></tr><tr><td>X\$C_GX_AND_REVERSE</td><td>src AND NOT dst</td></tr><tr><td>X\$C_GX_COPY</td><td>src</td></tr><tr><td>X\$C_GX_AND_INVERTED</td><td>(NOT src) AND dst</td></tr><tr><td>X\$C_GX_NOOP</td><td>dst</td></tr><tr><td>X\$C_GX_XOR</td><td>src XOR dst</td></tr><tr><td>X\$C_GX_OR</td><td>src OR dst</td></tr><tr><td>X\$C_GX_NOR</td><td>(NOT src) AND NOT dst</td></tr><tr><td>X\$C_GX_EQUIV</td><td>(NOT src) XOR dst</td></tr><tr><td>X\$C_GX_INVERT</td><td>NOT dst</td></tr><tr><td>X\$C_GX_OR_REVERSE</td><td>src OR NOT dst</td></tr><tr><td>X\$C_GX_COPY_INVERTED</td><td>NOT src</td></tr><tr><td>X\$C_GX_OR_INVERTED</td><td>(NOT src) OR dst</td></tr><tr><td>X\$C_GX_NAND</td><td>(NOT src) OR NOT dst</td></tr><tr><td>X\$C_GX_SET</td><td>1</td></tr></tbody></table>	Constant Name	Description	X\$C_GX_CLEAR	0	x\$C_GX_AND	src AND dst	X\$C_GX_AND_REVERSE	src AND NOT dst	X\$C_GX_COPY	src	X\$C_GX_AND_INVERTED	(NOT src) AND dst	X\$C_GX_NOOP	dst	X\$C_GX_XOR	src XOR dst	X\$C_GX_OR	src OR dst	X\$C_GX_NOR	(NOT src) AND NOT dst	X\$C_GX_EQUIV	(NOT src) XOR dst	X\$C_GX_INVERT	NOT dst	X\$C_GX_OR_REVERSE	src OR NOT dst	X\$C_GX_COPY_INVERTED	NOT src	X\$C_GX_OR_INVERTED	(NOT src) OR dst	X\$C_GX_NAND	(NOT src) OR NOT dst	X\$C_GX_SET	1
Constant Name	Description																																		
X\$C_GX_CLEAR	0																																		
x\$C_GX_AND	src AND dst																																		
X\$C_GX_AND_REVERSE	src AND NOT dst																																		
X\$C_GX_COPY	src																																		
X\$C_GX_AND_INVERTED	(NOT src) AND dst																																		
X\$C_GX_NOOP	dst																																		
X\$C_GX_XOR	src XOR dst																																		
X\$C_GX_OR	src OR dst																																		
X\$C_GX_NOR	(NOT src) AND NOT dst																																		
X\$C_GX_EQUIV	(NOT src) XOR dst																																		
X\$C_GX_INVERT	NOT dst																																		
X\$C_GX_OR_REVERSE	src OR NOT dst																																		
X\$C_GX_COPY_INVERTED	NOT src																																		
X\$C_GX_OR_INVERTED	(NOT src) OR dst																																		
X\$C_GX_NAND	(NOT src) OR NOT dst																																		
X\$C_GX_SET	1																																		
	<p>The screen the client is updating is the destination (dst). The graphics context the client uses to update the screen is the source (src). X\$L_GCVL_FUNCTION specifies how the server computes new destination bits from the source (src) and the old bits of the destination (dst).</p> <p>The most common logical function is the default specified by the constant x\$c_gx_copy, which uses only relevant values in the specified GC values structure to update the screen.</p>																																		
X\$L_GCVL_PLANE_MASK	<p>Specifies the planes on which the server performs the bitwise computation of pixels, defined by X\$L_GCVL_FUNCTION.</p> <p>Because a monochrome display has only one plane, the plane mask value is given in the least significant bit of the longword. As planes are added to the display hardware, they are defined in the more significant bits of the mask. The display routine ALL_PLANES specifies that all planes of the display are referred to simultaneously.</p> <p>The server does not perform range checking on the plane mask. It truncates values to the appropriate number of bits.</p>																																		
X\$L_GCVL_FOREGROUND	<p>Specifies an index to a color map for foreground color.</p>																																		
X\$L_GCVL_BACKGROUND	<p>Specifies an index to a color map for background color.</p>																																		

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 (Cont.) Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents
X\$_L_GCVL_LINE_WIDTH	<p>Defines the width of a line in pixels.</p> <p>The server draws a line with a width of two or more pixels centered on the path described in the graphics request and contained within a bounding box. Unless otherwise specified by the join or cap style, the bounding box of a line with endpoints [ <math>x_1</math>, <math>y_1</math> ], [ <math>x_2</math>, <math>y_2</math> ] and width <math>w</math> is a rectangle with vertices at the following real coordinates:</p> $\begin{aligned} &[x_1-w*\sin/2, y_1+w*\cos/2], [x_1+w*\sin/2, y_1-w*\cos/2] \\ &[x_2-w*\sin/2, y_2+w*\cos/2], [x_2+w*\sin/2, y_2-w*\cos/2] \end{aligned}$ <p>In this example, <math>\sin</math> is the sine of the angle of the line. The symbol <math>\cos</math> is the cosine of the angle of the line. A pixel is part of the line and is drawn if the center of the pixel is fully inside the bounding box. If the center of the pixel is exactly on the bounding box, the pixel is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior is immediately below the bounding box (y increasing direction).</p> <p>Lines with zero line width are one pixel wide. The server draws them using an unspecified, device-dependent algorithm that imposes the following two constraints:</p> <ul style="list-style-type: none"> <li>• If the server draws the line unclipped from [ <math>x_1</math>, <math>y_1</math> ] to [ <math>x_2</math>, <math>y_2</math> ], and if the server draws a second line from [ <math>x_1 + dx</math>, <math>y_1 + dy</math> ] to [ <math>x_2 + dx</math>, <math>y_2 + dy</math> ], then point [ <math>x</math>, <math>y</math> ] is touched by drawing the first line if and only if the point [ <math>x + dx</math>, <math>y + dy</math> ] is touched by drawing the second line.</li> <li>• The effective set of points that compose a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and if the point would be touched by the line when drawn unclipped.</li> </ul> <p>A line more than one pixel wide drawn from [ <math>x_1</math>, <math>y_1</math> ] to [ <math>x_2</math>, <math>y_2</math> ] always draws the same pixels as a line of the same width drawn from [ <math>x_2</math>, <math>y_2</math> ] to [ <math>x_1</math>, <math>y_1</math> ], excluding cap and join styles.</p> <p>In general, drawing a line whose line width is zero is faster than drawing a line whose line width is one or more. However, because the drawing algorithms for thin lines is different than those for wide lines, thin lines may not look as good when mixed with wide lines. If clients want precise and uniform results across all displays, they should always use a line width of one or more. Note, however, that specifying a line width of greater than zero decreases performance substantially.</p>

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 (Cont.) Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents										
X\$L_GCVL_LINE_STYLE	<p>Defines which sections of the line the server draws. The following lists available line styles and the constants that specify them. Figure 5–2 illustrates the styles.</p> <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>x\$c_line_solid</td> <td>The full path of the line is drawn.</td> </tr> <tr> <td>x\$c_line_double_dash</td> <td>The full path of the line is drawn, but the even dashes are filled differently from the odd dashes, with cap butt style used where even and odd dashes meet.</td> </tr> <tr> <td>x\$c_line_off_on_dash</td> <td>Only the even dashes are drawn. The X\$L_CAP_STYLE member applies to all internal ends of dashes. Specifying the constant, x\$c_cap_not_last, is equivalent to specifying x\$c_cap_but.</td> </tr> </tbody> </table>	Constant Name	Description	x\$c_line_solid	The full path of the line is drawn.	x\$c_line_double_dash	The full path of the line is drawn, but the even dashes are filled differently from the odd dashes, with cap butt style used where even and odd dashes meet.	x\$c_line_off_on_dash	Only the even dashes are drawn. The X\$L_CAP_STYLE member applies to all internal ends of dashes. Specifying the constant, x\$c_cap_not_last, is equivalent to specifying x\$c_cap_but.		
Constant Name	Description										
x\$c_line_solid	The full path of the line is drawn.										
x\$c_line_double_dash	The full path of the line is drawn, but the even dashes are filled differently from the odd dashes, with cap butt style used where even and odd dashes meet.										
x\$c_line_off_on_dash	Only the even dashes are drawn. The X\$L_CAP_STYLE member applies to all internal ends of dashes. Specifying the constant, x\$c_cap_not_last, is equivalent to specifying x\$c_cap_but.										
X\$L_GCVL_CAP_STYLE	<p>Defines how the server draws the endpoints of a path. The following lists available cap styles and the constants that specify them. Figure 5–3 illustrates the available cap styles.</p> <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>x\$c_cap_not_last</td> <td>Equivalent to specifying x\$c_cap_but, except that the final endpoint is not drawn if the line width is zero or one</td> </tr> <tr> <td>x\$c_cap_but</td> <td>Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint</td> </tr> <tr> <td>x\$c_cap_round</td> <td>A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to specifying x\$c_cap_but for a line width of zero or one)</td> </tr> <tr> <td>x\$c_cap_projecting</td> <td>Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to specifying x\$c_cap_but for a line width of zero or one)</td> </tr> </tbody> </table>	Constant Name	Description	x\$c_cap_not_last	Equivalent to specifying x\$c_cap_but, except that the final endpoint is not drawn if the line width is zero or one	x\$c_cap_but	Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint	x\$c_cap_round	A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to specifying x\$c_cap_but for a line width of zero or one)	x\$c_cap_projecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to specifying x\$c_cap_but for a line width of zero or one)
Constant Name	Description										
x\$c_cap_not_last	Equivalent to specifying x\$c_cap_but, except that the final endpoint is not drawn if the line width is zero or one										
x\$c_cap_but	Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint										
x\$c_cap_round	A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to specifying x\$c_cap_but for a line width of zero or one)										
x\$c_cap_projecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to specifying x\$c_cap_but for a line width of zero or one)										

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 (Cont.) Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents	
	<p>If a line has coincident endpoints ( <math>x_1 = x_2, y_1 = y_2</math>), the cap style is applied to both endpoints with the following results:</p>	
	<b>Line Width</b>	<b>Description</b>
	<b>Constant Name</b>	
	x\$c_cap_not_last	Thin Device dependent, but the desired effect is that nothing is drawn
	x\$c_cap_but	Thin Device dependent, but the desired effect is that a single pixel is drawn
	x\$c_cap_but	Wide Nothing is drawn
	x\$c_cap_round	Thin Device dependent, but the desired effect is that a single pixel is drawn
	x\$c_cap_round	Wide The closed path is a circle, centered at the endpoint, with the diameter equal to the line width
	x\$c_cap_projecting	Thin Device dependent, but the desired effect is that a single pixel is drawn
	x\$c_cap_projecting	Wide The closed path is a square, aligned with the coordinate axes, centered at the endpoint with sides equal to the line width
X\$L_GCVL_JOIN_STYLE	<p>Defines how the server draws corners for wide lines. Available join styles and the constants that specify them are as follows:</p>	
	<b>Constant Name</b>	<b>Description</b>
	x\$c_join_mitre	The outer edges of the two lines extend to meet at an angle.
	x\$c_join_round	A circular arc with diameter equal to the line width, centered at the join point.
	x\$c_join_bevel	Cap butt endpoint style, with the triangular “notch” filled.
	<p>For a line with coincident endpoints ( <math>x_1 = x_2, y_1 = y_2</math>), when the join style is applied at one or both endpoints, the effect is as if the line were removed from the overall path. However, if the total path consists of (or is reduced to) a single point joined with itself, the effect is the same as if the X\$L_GCVL_CAP_STYLE were applied to both endpoints. Figure 5–4 illustrates the styles.</p>	

(continued on next page)



# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 (Cont.) Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents																				
X\$_GCVL_FILL_STYLE	<p>Specifies the contents of the source for line, text, and fill operations. The following lists available fill styles for text and fill requests (DRAW TEXT, DRAW TEXT 16, FILL RECTANGLE, FILL POLYGON, FILL ARC). It also lists available styles applicable to solid lines and even dashes resulting from line requests (LINE, SEGMENTS, RECTANGLE, ARC):</p> <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>x\$_c_fill_solid</td> <td>Foreground</td> </tr> <tr> <td>x\$_c_fill_tiled</td> <td>Tile</td> </tr> <tr> <td>x\$_c_fill_opaque_stippled</td> <td>A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one</td> </tr> <tr> <td>x\$_c_fill_stippled</td> <td>Foreground masked by stipple</td> </tr> </tbody> </table> <p>The following lists available styles applicable to odd dashes resulting from line requests:</p> <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>x\$_c_fill_solid</td> <td>Background</td> </tr> <tr> <td>x\$_c_fill_tiled</td> <td>Tile</td> </tr> <tr> <td>x\$_c_fill_opaque_stippled</td> <td>A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one</td> </tr> <tr> <td>x\$_c_fill_stippled</td> <td>Background masked by stipple</td> </tr> </tbody> </table>	Constant Name	Description	x\$_c_fill_solid	Foreground	x\$_c_fill_tiled	Tile	x\$_c_fill_opaque_stippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one	x\$_c_fill_stippled	Foreground masked by stipple	Constant Name	Description	x\$_c_fill_solid	Background	x\$_c_fill_tiled	Tile	x\$_c_fill_opaque_stippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one	x\$_c_fill_stippled	Background masked by stipple
Constant Name	Description																				
x\$_c_fill_solid	Foreground																				
x\$_c_fill_tiled	Tile																				
x\$_c_fill_opaque_stippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one																				
x\$_c_fill_stippled	Foreground masked by stipple																				
Constant Name	Description																				
x\$_c_fill_solid	Background																				
x\$_c_fill_tiled	Tile																				
x\$_c_fill_opaque_stippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one																				
x\$_c_fill_stippled	Background masked by stipple																				
X\$_GCVL_FILL_RULE	<p>Defines what pixels the server draws along a path when a polygon is filled. The two available choices are x\$_c_even_odd_rule and x\$_c_winding_rule. The x\$_c_even_odd_rule constant defines a point to be inside a polygon if an infinite ray with the point as origin crosses the path an odd number of times. If the point meets these conditions, the server draws a corresponding pixel.</p>																				

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 (Cont.) Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents
	<p>The <code>x\$c_winding_rule</code> constant defines a point to be inside the polygon if an infinite ray with the pixel as origin crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise directed path segment is one that crosses the ray from left to right as observed from the pixel. A counterclockwise segment is one that crosses the ray from right to left as observed from that point. When a directed line segment coincides with a ray, choose a different ray that is not coincident with a segment. If the point meets these conditions, the server draws a corresponding pixel.</p> <p>For both even odd rule and winding rule, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside the polygon if the center point of the pixel is inside, and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction). Figure 5–5 illustrates fill rules.</p>
<code>X\$_GCVL_ARC_MODE</code>	<p>Controls how the server fills an arc. The available choices are specified by the constants <code>x\$c_arc_pie_slice</code> and <code>x\$c_arc_chord</code>. Figure 5–6 illustrates the two modes.</p>
<code>X\$_GCVL_TILE</code>	<p>Specifies the pixmap the server uses for tiling operations. The pixmap must have the same root and depth as the graphics context or an error occurs. Clients can use any size pixmap for tiling, although some sizes are faster than others.</p> <p>Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile, the results are not defined.</p>
<code>X\$_GCVL_STIPPLE</code>	<p>Specifies the pixmap the server uses for stipple operations. The pixmap must have the same root as the graphics context and a depth of one, or an error occurs. For stipple operations where the fill style is specified as <code>x\$c_fill_stippled</code> but not <code>x\$c_fill_opaque_stipple</code>, the stipple pattern is tiled in a single plane and acts as an additional clip mask. Perform a bitwise AND operation with the clip mask. Clients can use any size pixmap for stipple operations, although some sizes are faster than others.</p>
<code>X\$_GCVL_TS_X_ORIGIN</code>	<p>Defines the origin for tiling and stipple operations. Origins are relative to the origin of whatever window or pixmap is specified in the graphics request.</p>
<code>X\$_GCVL_TS_Y_ORIGIN</code>	<p>Defines the origin for tiling and stipple operations. Origins are relative to the origin of whatever window or pixmap is specified in the graphics request.</p>
<code>X\$_GCVL_FONT</code>	<p>Specifies the font that the server uses for text operations.</p>

(continued on next page)

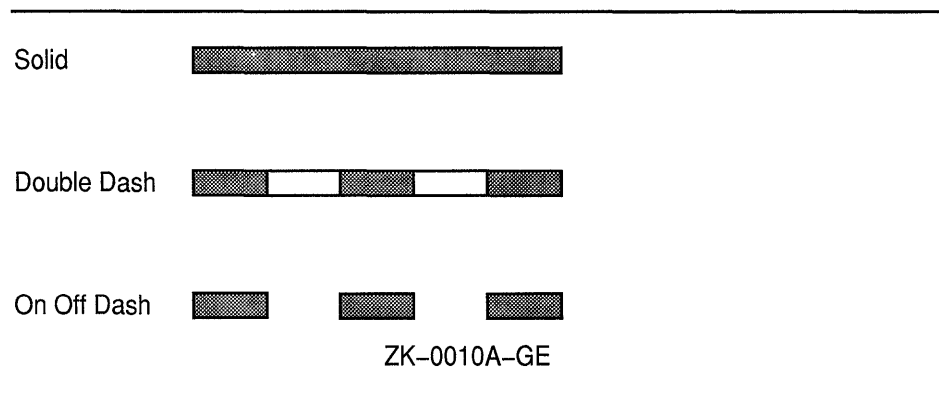
# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–2 (Cont.) Members of the GC Values Data Structure (VAX Binding)**

Member Name	Contents
X\$_GCVL_SUBWINDOW_MODE	Specifies whether inferior windows clip superior windows. The constant x\$c_clip_by_children specifies that all viewable input-output children clip both source and destination windows. The constant x\$c_include_inferiors specifies that inferiors clip neither source nor destination windows. This results in drawing through subwindow boundaries. The semantics of using the constant on a window with a depth of one and with mapped inferiors of differing depth is undefined by the core protocol.
X\$_GCVL_GRAPHIC_EXPOSURES	Specifies whether the server informs the client when the contents of a window region are lost.
X\$_GCVL_CLIP_X_ORIGIN	Defines the x-coordinate of the clip origin. The clip origin specifies the point within the clip region that is aligned with the drawable origin.
X\$_GCVL_CLIP_Y_ORIGIN	Defines the y-coordinate of the clip origin. The clip origin specifies the point within the clip region that is aligned with the drawable origin.
X\$_GCVL_CLIP_MASK	Identifies the pixmap the server uses to restrict write operations to the destination that is drawable. The pixmap must have a depth of one and have the same root as the graphics context. The clip mask clips only the destination that is drawable, not the source drawable. Where a value of one appears in the mask, the corresponding pixel in the destination drawable is drawn; where a value of zero occurs, no pixel is drawn. Any pixel within the destination drawable that is not represented within the clip mask pixmap is not drawn. When a client specifies the value of clip mask as x\$c_none, the server draws all pixels.
X\$_GCVL_DASH_OFFSET	Specifies the pixel within the dash length sequence, defined by X\$_GCVL_DASHES, to start drawing a dashed line. For example, a dash offset of zero starts a dashed line as the beginning of the dash line sequence. A dash offset of five starts the line at the fifth pixel of the line sequence.
X\$_GCVL_DASHES	Specifies the length, in number of pixels, of each dash. The value of this member must be nonzero or an error occurs.

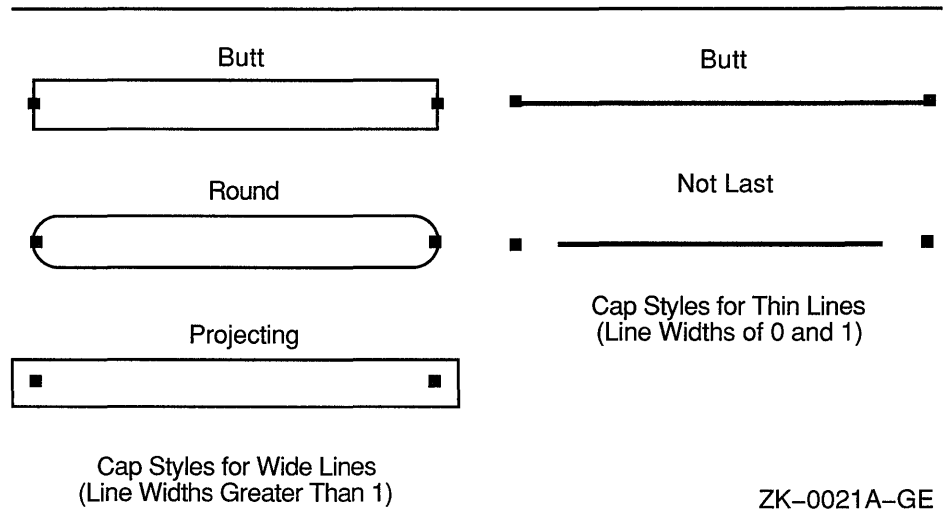
**Figure 5–2 Line Styles**



# Graphics Context Routines

## 5.1 The GC Values Data Structure

Figure 5-3 Cap Styles



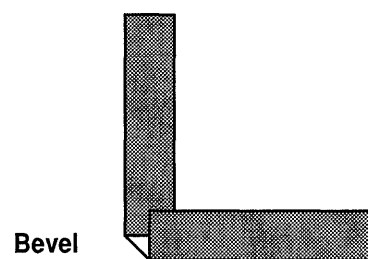
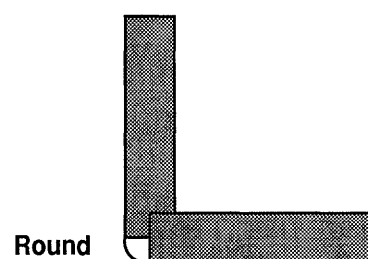
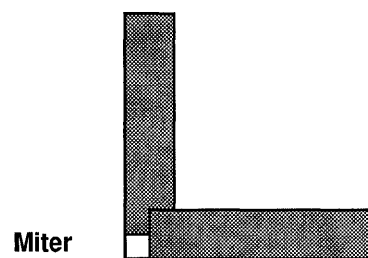
The default values for the GC values data structure are listed in Table 5-3. These values are used in the default graphics context.

# Graphics Context Routines

## 5.1 The GC Values Data Structure

Figure 5-4 Join Styles

---

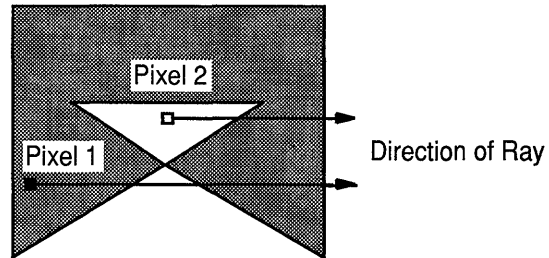


ZK-0013A-GE

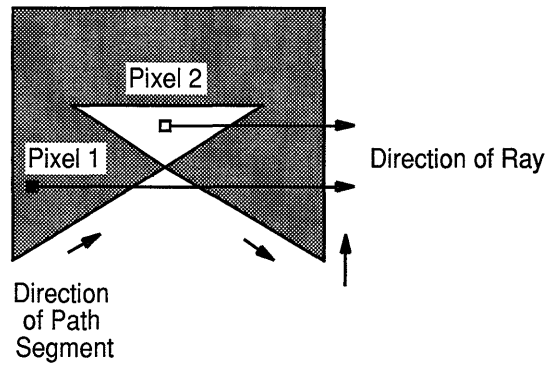
---

Figure 5-5 Fill Rules

Even Odd

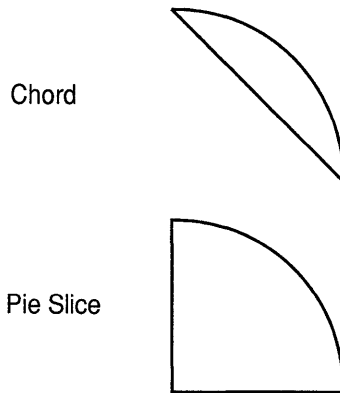


Winding



ZK-0071A-GE

Figure 5-6 Arc Fill Options



ZK-0008A-GE

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5-3 Default Values for the GC Values Data Structure**

Member	Default Value
X\$_L_GCVL_FUNCTION	X\$CGX_COPY
X\$_L_GCVL_PLANE_MASK	All ones
X\$_L_GCVL_FOREGROUND	0
X\$_L_GCVL_BACKGROUND	1
X\$_L_GCVL_LINE_WIDTH	0
X\$_L_GCVL_LINE_STYLE	x\$c_line_solid
X\$_L_GCVL_CAP_STYLE	x\$c_cap_but
X\$_L_GCVL_JOIN_STYLE	x\$c_join_mitre
X\$_L_GCVL_FILL_STYLE	x\$c_fill_solid
X\$_L_GCVL_FILL_RULE	x\$c_even_odd
X\$_L_GCVL_ARC_MODE	x\$c_arc_pie_slice
X\$_L_GCVL_TILE	Pixmap of unspecified size filled with foreground pixel
X\$_L_GCVL_STIPPLE	Pixmap of unspecified size filled with ones
X\$_L_GCVL_TS_X_ORIGIN	0
X\$_L_GCVL_TS_Y_ORIGIN	0
X\$_L_GCVL_FONT	Can vary
X\$_L_GCVL_SUBWINDOW_MODE	x\$C_clip_by_children
X\$_L_GCVL_GRAPHICS_EXPOSURES	True
X\$_L_GCVL_CLIP_X_ORIGIN	0
X\$_L_GCVL_CLIP_Y_ORIGIN	0
X\$_L_GCVL_CLIP_MASK	None
X\$_L_GCVL_DASH_OFFSET	0
X\$_L_GCVL_DASH_LIST	4 (the list [4,4])

The data structure for the MIT C binding is shown in Figure 5-7, and information about members in the data structure is described in Table 5-4.

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Figure 5–7 GC Values Data Structure (MIT C Binding)**

---

```
typedef struct {
    int function;
    unsigned long plane_mask;
    unsigned long foreground;
    unsigned long background;
    int line_width;
    int line_style;
    int cap_style;
    int join_style;
    int fill_style;
    int fill_rule;
    int arc_mode;
    Pixmap tile;
    Pixmap stipple;
    int ts_x_origin;
    int ts_y_origin;
    Font font;
    int subwindow_mode;
    Bool graphics_exposures;
    int clip_x_origin;
    int clip_y_origin;
    Pixmap clip_mask;
    int dash_offset;
    char dashes;
} XGCValues;
```

---

**Table 5–4 Members of the GC Values Data Structure (MIT C Binding)**

Member Name	Contents																								
function	<p>Defines how the server computes pixel values when the client updates a section of the screen. The following lists available functions:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Constant Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr><td>GXclear</td><td>0</td></tr> <tr><td>GXand</td><td>src AND dst</td></tr> <tr><td>GXandReverse</td><td>src AND NOT dst</td></tr> <tr><td>GXcopy</td><td>src</td></tr> <tr><td>GXandInverted</td><td>(NOT src) AND dst</td></tr> <tr><td>GXnoop</td><td>dst</td></tr> <tr><td>GXxor</td><td>src XOR dst</td></tr> <tr><td>GXor</td><td>src OR dst</td></tr> <tr><td>GXnor</td><td>(NOT src) AND NOT dst</td></tr> <tr><td>GXequiv</td><td>(NOT src) XOR dst</td></tr> <tr><td>GXinvert</td><td>NOT dst</td></tr> </tbody> </table>	Constant Name	Description	GXclear	0	GXand	src AND dst	GXandReverse	src AND NOT dst	GXcopy	src	GXandInverted	(NOT src) AND dst	GXnoop	dst	GXxor	src XOR dst	GXor	src OR dst	GXnor	(NOT src) AND NOT dst	GXequiv	(NOT src) XOR dst	GXinvert	NOT dst
Constant Name	Description																								
GXclear	0																								
GXand	src AND dst																								
GXandReverse	src AND NOT dst																								
GXcopy	src																								
GXandInverted	(NOT src) AND dst																								
GXnoop	dst																								
GXxor	src XOR dst																								
GXor	src OR dst																								
GXnor	(NOT src) AND NOT dst																								
GXequiv	(NOT src) XOR dst																								
GXinvert	NOT dst																								

---

(continued on next page)



# Graphics Context Routines

## 5.1 The GC Values Data Structure

Table 5-4 (Cont.) Members of the GC Values Data Structure (MIT C Binding)

Member Name	Contents												
	<table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>GXorReverse</td> <td>src OR NOT dst</td> </tr> <tr> <td>GXcopyInverted</td> <td>NOT src</td> </tr> <tr> <td>GXorInverted</td> <td>(NOT src) OR dst</td> </tr> <tr> <td>GXnand</td> <td>(NOT src) OR NOT dst</td> </tr> <tr> <td>GXset</td> <td>1</td> </tr> </tbody> </table>	Constant Name	Description	GXorReverse	src OR NOT dst	GXcopyInverted	NOT src	GXorInverted	(NOT src) OR dst	GXnand	(NOT src) OR NOT dst	GXset	1
Constant Name	Description												
GXorReverse	src OR NOT dst												
GXcopyInverted	NOT src												
GXorInverted	(NOT src) OR dst												
GXnand	(NOT src) OR NOT dst												
GXset	1												
	<p>The screen the client is updating is the destination (dst). The graphics context the client uses to update the screen is the source (src). The function member specifies how the server computes new destination bits from the source (src) and the old bits of the destination (dst).</p> <p>The most common logical function is the default specified by the constant GXcopy, which only uses relevant values in the specified GC VALUES structure to update the screen.</p>												
plane_mask	<p>Specifies the planes on which the server performs the bitwise computation of pixels, defined by <b>function</b>.</p> <p>Because a monochrome display has only one plane, the plane mask value is given in the least significant bit of the longword. As planes are added to the display hardware, they are defined in the more significant bits of the mask. The display routine ALL PLANES specifies that all planes of the display are referred to simultaneously.</p> <p>The server does not perform range checking on the plane mask. It truncates values to the appropriate number of bits.</p>												
foreground	Specifies an index to a color map for foreground color.												
background	Specifies an index to a color map for background color.												
line_width	<p>Defines the width of a line in pixels.</p> <p>The server draws a line with a width of two or more pixels centered on the path described in the graphics request and contained within a bounding box. Unless otherwise specified by the join or cap style, the bounding box of a line with endpoints [ <i>x1</i>, <i>y1</i>], [ <i>x2</i>, <i>y2</i>] and width <i>w</i> is a rectangle with vertices at the following real coordinates:</p> $[x1-w*sn/2, y1+w*cs/2], [x1+w*sn/2, y1-w*cs/2]$ $[x2-w*sn/2, y2+w*cs/2], [x2+w*sn/2, y2-w*cs/2]$ <p>In this example, <i>sn</i> is the sine of the angle of the line. The symbol <i>cs</i> is the cosine of the angle of the line. A pixel is part of the line and is drawn if the center of the pixel is fully inside the bounding box. If the center of the pixel is exactly on the bounding box, the pixel is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior is immediately below the bounding box (y increasing direction).</p>												

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–4 (Cont.) Members of the GC Values Data Structure (MIT C Binding)**

Member Name	Contents								
line_style	<p>Lines with zero line width are one pixel wide. The server draws them using an unspecified, device-dependent algorithm that imposes the following two constraints:</p> <ul style="list-style-type: none"> <li>• If the server draws the line unclipped from [ <math>x_1</math>, <math>y_1</math> ] to [ <math>x_2</math>, <math>y_2</math> ], and if the server draws a second line from [ <math>x_1 + dx</math>, <math>y_1 + dy</math> ] to [ <math>x_2 + dx</math>, <math>y_2 + dy</math> ], then point [ <math>x</math>, <math>y</math> ] is touched by drawing the first line if and only if the point [ <math>x + dx</math>, <math>y + dy</math> ] is touched by drawing the second line.</li> <li>• The effective set of points that compose a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and if the point would be touched by the line when drawn unclipped.</li> </ul> <p>A line more than one pixel wide drawn from [ <math>x_1</math>, <math>y_1</math> ] to [ <math>x_2</math>, <math>y_2</math> ] always draws the same pixels as a line of the same width drawn from [ <math>x_2</math>, <math>y_2</math> ] to [ <math>x_1</math>, <math>y_1</math> ], excluding cap and join styles.</p> <p>In general, drawing a line whose line width is zero is faster than drawing a line whose line width is one or more. However, because the drawing algorithms for thin lines is different than those for wide lines, thin lines may not look as good when mixed with wide lines. If clients want precise and uniform results across all displays, they should always use a line width of one or more. Note, however, that specifying a line width of greater than zero decreases performance substantially.</p> <p>Defines which sections of the line the server draws. The following lists available line styles and the constants that specify them:</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <thead> <tr> <th style="text-align: left;">Constant Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>LineSolid</td> <td>The full path of the line is drawn</td> </tr> <tr> <td>LineDoubleDash</td> <td>The full path of the line is drawn, but the even dashes are filled differently than the odd dashes, with cap butt style used where even and odd dashes meet</td> </tr> <tr> <td>LineOffOnDash</td> <td>Only the even dashes are drawn. The cap_style member applies to all internal ends of dashes. Specifying the constant CapNotLast is equivalent to specifying CapButt.</td> </tr> </tbody> </table>	Constant Name	Description	LineSolid	The full path of the line is drawn	LineDoubleDash	The full path of the line is drawn, but the even dashes are filled differently than the odd dashes, with cap butt style used where even and odd dashes meet	LineOffOnDash	Only the even dashes are drawn. The cap_style member applies to all internal ends of dashes. Specifying the constant CapNotLast is equivalent to specifying CapButt.
Constant Name	Description								
LineSolid	The full path of the line is drawn								
LineDoubleDash	The full path of the line is drawn, but the even dashes are filled differently than the odd dashes, with cap butt style used where even and odd dashes meet								
LineOffOnDash	Only the even dashes are drawn. The cap_style member applies to all internal ends of dashes. Specifying the constant CapNotLast is equivalent to specifying CapButt.								
	<p>Figure 5–2 illustrates the styles.</p>								

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–4 (Cont.) Members of the GC Values Data Structure (MIT C Binding)**

Member Name	Contents										
cap_style	<p>Defines how the server draws the endpoints of a path. The following lists available cap styles and the constants that specify them:</p> <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>CapNotLast</td> <td>Equivalent to CapButt, except that the final endpoint is not drawn if the line width is zero or one</td> </tr> <tr> <td>CapButt</td> <td>Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint</td> </tr> <tr> <td>CapRound</td> <td>A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to the value specified by CapButt for a line width of zero or one)</td> </tr> <tr> <td>CapProjecting</td> <td>Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to the value specified by the constant CapButt for a line width of zero or one)</td> </tr> </tbody> </table>	Constant Name	Description	CapNotLast	Equivalent to CapButt, except that the final endpoint is not drawn if the line width is zero or one	CapButt	Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint	CapRound	A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to the value specified by CapButt for a line width of zero or one)	CapProjecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to the value specified by the constant CapButt for a line width of zero or one)
Constant Name	Description										
CapNotLast	Equivalent to CapButt, except that the final endpoint is not drawn if the line width is zero or one										
CapButt	Square at the endpoint (perpendicular to the slope of the line) with no projection beyond the endpoint										
CapRound	A circular arc with the diameter equal to the line width, centered on the endpoint (equivalent to the value specified by CapButt for a line width of zero or one)										
CapProjecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the width of the line (equivalent to the value specified by the constant CapButt for a line width of zero or one)										

Figure 5–3 illustrates the butt, round, and projecting cap styles.

If a line has coincident endpoints ( $x_1 = x_2$ ,  $y_1 = y_2$ ), the cap style is applied to both endpoints with the following results:

Constant Name	Line Width	Description
CapNotLast	Thin	Device dependent, but the desired effect is that nothing is drawn
CapButt	Thin	Device dependent, but the desired effect is that a single pixel is drawn
CapButt	Wide	Nothing is drawn
CapRound	Thin	Device dependent, but the desired effect is that a single pixel is drawn
CapRound	Wide	The closed path is a circle, centered at the endpoint, with the diameter equal to the line width
CapProjecting	Thin	Device dependent, but the desired effect is that a single pixel is drawn
CapProjecting	Wide	The closed path is a square, aligned with the coordinate axes, centered at the endpoint with sides equal to the line width

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–4 (Cont.) Members of the GC Values Data Structure (MIT C Binding)**

Member Name	Contents																				
join_style	<p>Defines how the server draws corners for wide lines. Available join styles and the constants that specify them are as follows:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Constant Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>JoinMitre</td> <td>The outer edges of the two lines extend to meet at an angle.</td> </tr> <tr> <td>JoinRound</td> <td>A circular arc with diameter equal to the line width, centered at the join point.</td> </tr> <tr> <td>JoinBevel</td> <td>Cap butt endpoint style, with the triangular “notch” filled.</td> </tr> </tbody> </table> <p>Figure 5–4 illustrates the styles.</p> <p>For a line with coincident endpoints (<math>x_1 = x_2</math>, <math>y_1 = y_2</math>), when the join style is applied at one or both endpoints, the effect is as if the line were removed from the overall path. However, if the total path consists of (or is reduced to) a single point joined with itself, the effect is the same as if the cap style were applied to both endpoints.</p>	Constant Name	Description	JoinMitre	The outer edges of the two lines extend to meet at an angle.	JoinRound	A circular arc with diameter equal to the line width, centered at the join point.	JoinBevel	Cap butt endpoint style, with the triangular “notch” filled.												
Constant Name	Description																				
JoinMitre	The outer edges of the two lines extend to meet at an angle.																				
JoinRound	A circular arc with diameter equal to the line width, centered at the join point.																				
JoinBevel	Cap butt endpoint style, with the triangular “notch” filled.																				
fill_style	<p>Specifies the contents of the source for line, text, and fill operations. The following lists available fill styles for text and fill requests (DRAW TEXT, DRAW TEXT 16, FILL RECTANGLE, FILL POLYGON, FILL ARC). It also lists available styles applicable to solid lines and even dashes resulting from line requests (LINE, SEGMENTS, RECTANGLE, ARC):</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Constant Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>FillSolid</td> <td>Foreground</td> </tr> <tr> <td>FillTiled</td> <td>Tile</td> </tr> <tr> <td>FillOpaqueStippled</td> <td>A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one</td> </tr> <tr> <td>FillStippled</td> <td>Foreground masked by stipple</td> </tr> </tbody> </table> <p>The following lists available styles applicable to odd dashes resulting from line requests:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Constant Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>FillSolid</td> <td>Background</td> </tr> <tr> <td>FillTiled</td> <td>Tile</td> </tr> <tr> <td>FillOpaqueStippled</td> <td>A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one</td> </tr> <tr> <td>FillStippled</td> <td>Background masked by stipple</td> </tr> </tbody> </table>	Constant Name	Description	FillSolid	Foreground	FillTiled	Tile	FillOpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one	FillStippled	Foreground masked by stipple	Constant Name	Description	FillSolid	Background	FillTiled	Tile	FillOpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one	FillStippled	Background masked by stipple
Constant Name	Description																				
FillSolid	Foreground																				
FillTiled	Tile																				
FillOpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one																				
FillStippled	Foreground masked by stipple																				
Constant Name	Description																				
FillSolid	Background																				
FillTiled	Tile																				
FillOpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one																				
FillStippled	Background masked by stipple																				

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–4 (Cont.) Members of the GC Values Data Structure (MIT C Binding)**

Member Name	Contents
fill_rule	<p>Defines what pixels the server draws along a path when a polygon is filled. The two available choices are EvenOddRule and WindingRule. The EvenOddRule constant defines a point to be inside a polygon if an infinite ray with the point as origin crosses the path an odd number of times. If the point meets these conditions, the server draws a corresponding pixel.</p> <p>The WindingRule constant defines a point to be inside the polygon if an infinite ray with the pixel as origin crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise directed path segment is one that crosses the ray from left to right as observed from the pixel. A counterclockwise segment is one that crosses the ray from right to left as observed from that point. When a directed line segment coincides with a ray, choose a different ray that is not coincident with a segment. If the point meets these conditions, the server draws a corresponding pixel.</p> <p>For both even odd rule and winding rule, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside the polygon if the center point of the pixel is inside, and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction).</p> <p>Figure 5–5 illustrates fill rules.</p>
arc_mode	<p>Controls how the server fills an arc. The available choices are the values specified by the ArcPieSlice and ArcChord constants. Figure 5–6 illustrates the two modes.</p>
tile	<p>Specifies the pixmap the server uses for tiling operations. The pixmap must have the same root and depth as the graphics context or an error occurs. Clients can use any size pixmap for tiling, although some sizes are faster than others.</p> <p>Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is used later as the destination for a graphics request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile, the results are not defined.</p>
stipple	<p>Specifies the pixmap the server uses for stipple operations. The pixmap must have the same root as the graphics context and a depth of one, or an error occurs. For stipple operations where the fill style is specified as the FillStippled constant but not the FillOpaqueStipple constant, the stipple pattern is tiled in a single plane and acts as an additional clip mask. Perform a bitwise AND operation with the clip mask. Clients can use any size pixmap for stipple operations, although some sizes are faster than others.</p> <p>Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is used later as the destination for a graphics request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a stipple, the results are not defined.</p>
ts_x_origin	<p>Defines the origin for tiling and stipple operations. Origins are relative to the origin of whatever window or pixmap is specified in the graphics request.</p>

(continued on next page)

## Graphics Context Routines

### 5.1 The GC Values Data Structure

**Table 5–4 (Cont.) Members of the GC Values Data Structure (MIT C Binding)**

Member Name	Contents
ts_y_origin	Defines the origin for tiling and stipple operations. Origins are relative to the origin of whatever window or pixmap is specified in the graphics request.
font	Specifies the font that the server uses for text operations.
subwindow_mode	Specifies whether inferior windows clip superior windows. The constant <code>ClipByChildren</code> specifies that all viewable input-output children clip both source and destination windows. The constant <code>IncludeInferiors</code> specifies that inferiors clip neither source nor destination windows. This results in drawing through subwindow boundaries. The semantics of using the constant on a window with a depth of one and with mapped inferiors of differing depth is undefined by the core protocol.
graphic_exposures	Specifies whether the server informs the client when the contents of a window region are lost.
clip_x_origin	Defines the x-coordinate of the clip origin. The clip origin specifies the point within the clip region that is aligned with the drawable origin.
clip_y_origin	Defines the y-coordinate of the clip origin. The clip origin specifies the point within the clip region that is aligned with the drawable origin.
clip_mask	Identifies the pixmap the server uses to restrict write operations to the destination that is drawable. The pixmap must have a depth of one and have the same root as the graphics context. The clip mask clips only the destination that is drawable, not the source drawable. Where a value of one appears in the mask, the corresponding pixel in the destination drawable is drawn; where a value of zero occurs, no pixel is drawn. Any pixel within the destination drawable that is not represented within the clip mask pixmap is not drawn. When a client specifies the value of clip mask as <b>None</b> , the server draws all pixels.
dash_offset	Specifies the pixel within the dash length sequence, defined by the dashes member, to start drawing a dashed line. For example, a dash offset of zero starts a dashed line as the beginning of the dash line sequence. A dash offset of five starts the line at the fifth pixel of the line sequence. Figure 5–9 illustrated dashed offsets.
dashes	Specifies the length, in number of pixels, of each dash. The value of this member must be nonzero or an error occurs.

The default values for the GC values data structure are listed in Table 5–5. These values are used in the default graphics context.

**Table 5–5 Default Values for the GC Values Data Structure**

Member	Default Value
function	GXcopy
plane_mask	All ones
foreground	0
background	1
line_width	0
line_style	Solid

(continued on next page)

# Graphics Context Routines

## 5.1 The GC Values Data Structure

**Table 5–5 (Cont.) Default Values for the GC Values Data Structure**

Member	Default Value
cap_style	Butt
join_style	Miter
fill_style	Solid
fill_rule	EvenOdd
arc_mode	PieSlice
tile	Pixmap of unspecified size filled with foreground pixel
stipple	Pixmap of unspecified size filled with ones
ts_x_origin	0
ts_y_origin	0
font	Can vary
subwindow_mode	ClipByChildren
graphics_exposures	True
clip_x_origin	0
clip_y_origin	0
clip_mask	None
dash_offset	0
dash_list	4 (the list [4,4])

## 5.2 GC Mask

Table 5–6 lists the predefined values for the graphics context bit mask and their meaning.

**Table 5–6 GC Mask Bits**

VAX Predefined Bit Value	C Predefined Bit Value	Meaning when Set
X\$M_GC_FUNCTION	GCFunction	Change the function member (default is GXCopy)
X\$M_GC_PLANE_MASK	GCPlaneMask	Change the plane mask member (default is all ones)
X\$M_GC_FOREGROUND	GCForeground	Change the foreground member (default is 0)
X\$M_GC_BACKGROUND	GCBackground	Change the background member (default is 1)
X\$M_GC_LINE_WIDTH	GCLineWidth	Change the line width member (default is 0)

(continued on next page)

# Graphics Context Routines

## 5.2 GC Mask

Table 5–6 (Cont.) GC Mask Bits

VAX Predefined Bit Value	C Predefined Bit Value	Meaning when Set
X\$M_GC_LINE_STYLE	GCLineStyle	Change the line style member (default is LineSolid)
X\$M_GC_CAP_STYLE	GCCapStyle	Change the cap, endpoint, style member (default is CapButt)
X\$M_GC_JOIN_STYLE	GCJoinStyle	Change the join style member (default is JoinMiter)
X\$M_GC_FILL_STYLE	GCFillStyle	Change the fill style member (default is FillSolid)
X\$M_GC_FILL_RULE	GCFillRule	Change the fill rule member (default is EvenOddRule)
X\$M_GC_ARC_MODE	GCArcMode	Change the arc fill mode (default is ArcPieSlice)
X\$M_GC_TILE	GCTile	Change the tile pixmap identifier (default is a pixmap of unspecified size filled with foreground pixel)
X\$M_GC_STIPPLE	GCStipple	Change the stipple pixmap identifier (default is a pixmap of unspecified size filled with ones)
X\$M_GC_TILE_STIP_X_ORIGIN	GCTileStipXOrigin	Change the x-coordinate for a tile/stipple origin (default is 0)
X\$M_TILE_STIP_Y_ORIGIN	GCTileStipYOrigin	Change the y-coordinate for a tile/stipple origin (default is 0)
X\$M_GC_FONT	GCFont	Change the font identifier
X\$M_GC_SUBWINDOW_MODE	GCSubwindowMode	Change the subwindow mode member (default is ClipByChildren)
X\$M_GC_GRAPHICS_EXPOSURE	GCGraphicsExposures	Change the graphics exposures flag (default is True)
X\$M_GC_CLIP_X_ORIGIN	GCclipXOrigin	Change the x-coordinate for a clip origin (default is 0)

(continued on next page)



## Graphics Context Routines

### 5.2 GC Mask

Table 5–6 (Cont.) GC Mask Bits

VAX Predefined Bit Value	C Predefined Bit Value	Meaning when Set
X\$M_GC_CLIP_Y_ORIGIN	GCClipYOrigin	Change the y-coordinate for a clip origin (default is 0)
X\$M_GC_CLIP_MASK	GCClipMask	Change clip mask pixmap identifier (default is none)
X\$M_GC_DASH_OFFSET	GCDashOffset	Change the dash offset member (default is 0)
X\$M_GC_DASH_LIST	GCDashList	Change the dash list member (default is 4)

## 5.3 Graphics Context Routines

The following pages describe the Xlib graphics context routines.

---

## CHANGE GC

Changes any attribute by changing the values in the GC values data structure.

---

**VAX FORMAT**    **X\$CHANGE\_GC**  
(*display, gc\_id, gc\_mask, values\_struct*)

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
gc_mask	mask_longword	uns longword	read	reference
values_struct	record	x\$gc_values	read	reference

---

**MIT C FORMAT**    **XChangeGC**  
(*display, gc\_id, gc\_mask, values\_struct*)

---

**argument  
information**

```
XChangeGC(display, gc_id, gc_mask, values_struct)
    Display *display;
    GC gc_id;
    unsigned long gc_mask;
    XGCValues *values_struct;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***gc\_id***  
The identifier of the graphics context to be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***gc\_mask***  
A bit mask that specifies the members in the graphics context to be changed. A bitwise logical OR operation of the members sets the mask.

Table 5–6 lists the predefined values and the description for the ***gc\_mask*** argument.

***values\_struct***  
The GC values data structure that contains the new values. The new values specified in ***gc\_mask*** will be copied into the GC values data structure specified by ***gc\_id***.

# Graphics Context Routines

## CHANGE GC

---

### DESCRIPTION

CHANGE GC uses the mask to identify the values in the graphics context to be changed. You identify each member you want to change in that mask, then provide the new values for those members in the GC values data structure passed in **values\_struct**.

If the clip mask member is changed by this routine, any previous SET CLIP RECTANGLES routine request for the same graphics context is overridden by the new clip mask. If the dash offset or dash list members are changed by this routine, any previous SET DASHES request for the same graphics context is overridden.

For more information on using the GC values data structure and an illustration of this structure, refer to Section 5.1.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

## COPY GC

Copies the graphics context values from one graphics context data structure to another GC values data structure.

**VAX FORMAT**     **X\$COPY\_GC**  
                           (*display, src\_gc\_id, gc\_mask, dst\_gc\_id*)

**argument information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
src_gc_id	identifier	uns longword	read	reference
gc_mask	mask_longword	uns longword	read	reference
dst_gc_id	identifier	uns longword	read	reference

**MIT C FORMAT**     **XCopyGC**  
                           (*display, src\_gc\_id, gc\_mask, dst\_gc\_id*)

**argument information**

```
XCopyGC (display, src_gc_id, gc_mask, dst_gc_id)
Display *display;
GC src_gc_id, dst_gc_id;
unsigned long gc_mask;
```

**ARGUMENTS**

- display***  
The display information originally returned by OPEN DISPLAY.
- src\_gc\_id***  
The identifier of the source graphics context to be copied. The source GC values data structure contains the values that will be copied. This identifier was originally returned by DEFAULT GC or CREATE GC.
- gc\_mask***  
The bit mask that specifies the members that will be copied from the source graphics context to the destination graphics context. Use a bitwise OR operation of the predefined values to set the bit mask, or use the mask structure.

Table 5–6 lists the predefined values and descriptions for the **gc\_mask**.

# Graphics Context Routines

## COPY GC

### *dst\_gc\_id*

The identifier of the destination graphics context that the values from the source graphics context are copied to. Only those values specified in the **gc\_mask** are changed. This identifier was originally returned by DEFAULT GC or CREATE GC.

---

## DESCRIPTION

COPY GC copies specific values from one graphics context to another. The only values copied are those specified in the **gc\_mask**. The source graphics context and the destination graphics context must be of the same root and depth.

The graphics context is an identifier for an internal structure/representation and this routine must be used to copy it from one variable into others.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## CREATE GC

Creates a new graphics context for a specific window. Unless specified, the members of the graphics context are the default values.

---

**VAX FORMAT**     *gc\_id\_return = X\$CREATE\_GC*  
                          (*display, drawable\_id, gc\_mask, values\_struct*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
gc_id_return	identifier	uns longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_mask	mask_longword	uns longword	read	reference
values_struct	record	x\$gc_values	read	reference

---

---

**MIT C FORMAT**     *gc\_id\_return = XCreateGC*  
                          (*display, drawable\_id, gc\_mask, values\_struct*)

**argument  
information**

```
GC XCreateGC(display, drawable_id, gc_mask, values_struct)
    Display *display;
    Drawable drawable_id;
    unsigned long gc_mask;
    XGCValues *values_struct;
```

---

**RETURNS**             *gc\_id\_return*  
The identifier of the new graphics context is returned.

---

**ARGUMENTS**         *display*  
The display information originally returned by OPEN DISPLAY.

*drawable\_id*  
The identifier of the window or pixmap (drawable) to create the new graphics context for.

# Graphics Context Routines

## CREATE GC

### *gc\_mask*

The bit mask that specifies the values within the GC values data structure that are different from the default values. The values that are different from the default values are specified in **values\_struct**. Do a bitwise OR operation with the predefined values to set the mask.

Table 5–6 lists the predefined values and descriptions for the **gc\_mask**.

### *values\_struct*

The GC values data structure that specifies nondefault values for the new graphics context.

For more information about this data structure, see the Section 5.1.

---

## DESCRIPTION

CREATE GC creates a new graphic context for use with a specified pixmap or window (drawable). It returns an identifier for the new graphics context. Use the identifier to reference the graphics context in any subsequent routines.

If you need to use nondefault values, use this routine. To specify a nondefault value, set the appropriate bit in **gc\_mask**. Then, specify the value you want in the appropriate member in **values\_struct**. If you do not set the bit, the value is not changed.

Once created, you can change graphic context values with the CHANGE GC or individual SET routines. You can also copy the values from one graphics context to another one using the COPY GC routine.

You can use a default graphics context if the default values are valid for your program. Use the DEFAULT GC to return the identifier of the default graphics context. The default values are listed in Section 5.1.

When you are done with the graphics context, be sure to call FREE GC to free the storage associated with the graphics context.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).

## Graphics Context Routines

### CREATE GC

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.



# Graphics Context Routines

## FREE GC

---

### FREE GC

Frees a specified graphics context.

---

#### VAX FORMAT **X\$FREE\_GC** (*display, gc\_id*)

##### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference

---

---

#### MIT C FORMAT **XFreeGC** (*display, gc\_id*)

##### argument information

```
XFreeGC(display, gc_id)
Display *display;
GC gc_id;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***gc\_id***

The identifier of the graphics context to be freed. This must be an identifier that was originally returned by CREATE GC or COPY GC; it cannot be the default graphics context identifier.

---

### DESCRIPTION

FREE GC destroys a specified graphics context, and frees the storage associated with the graphics context as well.

You cannot free the default GC values data structure; you can free only those created with CREATE GC or COPY GC. Therefore, you cannot specify the identifier of the default graphics context in **gc\_id**.

---

### X ERRORS

---

VAX	C	Description
X\$_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

---

---

## GCONTEXT FROM GC

Obtains the graphics context resource identifier for a specified graphics context.

---

**VAX FORMAT**     *gc\_resource\_id\_return = X\$CREATE\_GC (gc\_id)*

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
<i>gc_resource_id_return</i>	identifier	uns longword	write	value
<i>gc_id_return</i>	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**     *gc\_resource\_id\_return = XCreateGC (gc\_id)*

---

**argument  
information**

```
GContext XGContextFromGC(gc_id)
GC gc_id;
```

---

**RETURNS**     *gc\_resource\_id\_return*  
The graphics context resource identifier for the specified graphics context.

---

---

**ARGUMENTS**     *gc\_id*  
The graphics context for which you want to obtain the resource identifier.

---

---

**DESCRIPTION**     GCONTEXT FROM GC obtains the graphics context resource identifier for a specified graphics context.

---

# Graphics Context Routines

## QUERY BEST SIZE

---

### QUERY BEST SIZE

Obtains the optimal size for improved server performance for the cursor, the tile pattern, or the stipple pattern.

---

**VAX FORMAT**     *status\_return = X\$QUERY\_BEST\_SIZE*  
*(display, class, drawable\_id, width, height,*  
*width\_return, height\_return)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
class	longword	longword	read	reference
drawable_id	identifier	uns longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference
width_return	uns longword	uns longword	write	reference
height_return	uns longword	uns longword	write	reference

---

---

**MIT C FORMAT**     *status\_return = XQueryBestSize*  
*(display, class, drawable\_id, width, height,*  
*width\_return, height\_return)*

**argument  
information**

```
Status XQueryBestSize(display, class, drawable_id, width, height,  
                      width_return, height_return)  
Display *display;  
int class;  
Drawable drawable_id;  
unsigned int width, height;  
unsigned int *width_return, *height_return;
```

---

### RETURNS

***status\_return***  
Specifies whether the routine completed successfully.

**C only**

This argument returns 1 if the routine completes successfully, and zero if it does not complete successfully.

# Graphics Context Routines

## QUERY BEST SIZE

### VAX only

This argument returns one of the following values.

Value	Description
X\$_ERRORREPLY	Error received from the server.
SS\$_NORMAL	Success.

## ARGUMENTS

### *display*

The display information originally returned by OPEN DISPLAY.

### *class*

The item to determine the best size for. The predefined values for **class** are as follows:

VAX	C	Description
X\$_C_TILE_SHAPE	TileShape	Determine the best size for the tile pattern.
X\$_C_STIPPLE_SHAPE	StippleShape	Determine the best size for the stipple pattern.
X\$_C_CURSOR_SHAPE	CursorShape	Determine the best size for the cursor.

Other values specified in this argument are not valid.

### *drawable\_id*

The identifier of the drawable (window or pixmap) associated with the item to determine the best size for. The **drawable\_id** argument cannot refer to an input-only window for a tile or a stipple, otherwise the routine cannot complete successfully.

### *width*

The width, in pixels, of the item to determine the best size for. The width and height determine the desired area of the tile, stipple, or cursor.

### *height*

The height, in pixels, of the item to determine the best size for. The height and width determine the desired area of the tile, stipple, or cursor.

### *width\_return*

A pointer to the optimal width, in pixels, of the item returned by the server. The width and height determine the best size closest to the area specified for the tile, stipple, or cursor that maximizes server performance.

### *height\_return*

A pointer to the optimal height, in pixels, of the item returned by the server. The width and height determine the best size that is closest to the area specified for the tile, stipple, or cursor that maximizes server performance.

# Graphics Context Routines

## QUERY BEST SIZE

---

### DESCRIPTION

QUERY BEST SIZE obtains the optimal size for a tile, stipple, or cursor pattern that maximizes server performance. For a tile and stipple, the best size is the size that can be tiled or stippled in the least amount of time. For a cursor, the best size is the largest cursor that can be fully displayed.

You specify the desired size of the tile, stipple, or cursor in the **width** and **height** arguments. The server returns the best size that is closest to the size you specified for the item in the **width\_return** and **height\_return** arguments.

You can also use the QUERY BEST TILE, QUERY BEST STIPPLE and QUERY BEST CURSOR routines to determine the best sizes for these items.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

## QUERY BEST STIPPLE

Obtains the optimal size for improved server performance for a stipple pattern.

**VAX FORMAT**     *status\_return = X\$QUERY\_BEST\_STIPPLE*  
                           (*display, drawable\_id, width, height, width\_return,*  
                           *height\_return*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference
width_return	uns longword	uns longword	write	reference
height_return	uns longword	uns longword	write	reference

**MIT C FORMAT**     *status\_return = XQueryBestStipple*  
                           (*display, drawable\_id, width, height, width\_return,*  
                           *height\_return*)

**argument  
information**

```
Status XQueryBestStipple(display, drawable_id, width, height,
                          width_return, height_return)
Display *display;
Drawable drawable_id;
unsigned int width, height;
unsigned int *width_return, *height_return;
```

**RETURNS**

***status\_return***  
 Specifies whether the routine completed successfully.

**C only**

This argument returns 1 if the routine completes successfully, and zero if it does not complete successfully.

## Graphics Context Routines

### QUERY BEST STIPPLE

#### VAX only

This argument returns one of the following values.

Value	Description
X\$_ERRORREPLY	Error received from the server.
SS\$_NORMAL	Success.

---

## ARGUMENTS

### *display*

The display information originally returned by OPEN DISPLAY.

### *drawable\_id*

The identifier of the drawable (window or pixmap) associated with the stipple pattern. The drawable refers to the screen and possibly the window class and depth that the stipple pattern will be used on. The **drawable\_id** argument cannot refer to an input-only window, otherwise the routine cannot complete successfully.

### *width*

The width, in pixels, of the stipple pattern to determine the best size for. The width and height determine the desired area of the stipple.

### *height*

The height, in pixels, of the stipple pattern to determine the best size for. The height and width determine the desired area of the stipple.

### *width\_return*

A pointer to the optimal width, in pixels, of the stipple returned by the server. The width and height determine the best size that is closest to the size specified for the stipple that maximizes server performance.

### *height\_return*

A pointer to the optimal height, in pixels, of the stipple returned by the server. The width and height determine the best size that is closest to the size specified for the stipple that maximizes server performance.

---

## DESCRIPTION

QUERY BEST STIPPLE obtains the optimal size for a stipple pattern that maximizes server performance. The best size is the size that can be stippled in the least amount of time.

You specify the desired size of the stipple in the **width** and **height** arguments. The server returns the best size that is closest to the size specified in the **width\_return** and **height\_return** arguments.

You can also use the QUERY BEST SIZE routine to determine the best size for a stipple pattern.

## Graphics Context Routines

### QUERY BEST STIPPLE

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---



# Graphics Context Routines

## QUERY BEST TILE

---

### QUERY BEST TILE

Obtains the optimal size for improved server performance for a tile pattern.

---

**VAX FORMAT**     *status\_return = X\$QUERY\_BEST\_TILE*  
*(display, drawable\_id, width, height, width\_return,*  
*height\_return)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
width	uns longword	uns longword	read	reference
height	uns longword	uns longword	read	reference
width_return	uns longword	uns longword	write	reference
height_return	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**     *status\_return = XQueryBestTile*  
*(display, drawable\_id, width, height, width\_return,*  
*height\_return)*

**argument  
information**

---

```
Status XQueryBestTile(display, drawable_id, width, height,  
                      width_return, height_return)  
Display *display;  
Drawable drawable_id;  
unsigned int width, height;  
unsigned int *width_return, *height_return;
```

---

**RETURNS**

***status\_return***  
Specifies whether the routine completed successfully.

**VAX only**

This argument returns one of the following values:

# Graphics Context Routines

## QUERY BEST TILE

---

Value	Description
X\$_ERRORREPLY	Error received from the server.
SS\$_NORMAL	Success.

---

### C only

This argument returns 1 if the routine completes successfully, and zero if it does not complete successfully.

---

## ARGUMENTS

### *display*

The display information originally returned by OPEN DISPLAY.

### *drawable\_id*

The identifier of the drawable (window or pixmap) associated with the tile pattern. The drawable refers to the screen and possibly the window class and depth that the tile pattern will be used on. The **drawable\_id** cannot refer to an input-only window, otherwise the routine cannot complete successfully.

### *width*

The width, in pixels, of the tile pattern to determine the best size for. The width and height determine the desired area of the tile.

### *height*

The height, in pixels, of the tile pattern to determine the best size for. The height and width determine the desired area of the tile.

### *width\_return*

A pointer to the optimal width, in pixels, of the tile returned by the server. The width and height determine the best size that is closest to the size specified for the tile that maximizes server performance.

### *height\_return*

A pointer to the optimal height, in pixels, of the tile returned by the server. The width and height determine the best size that is closest to the size specified for the tile that maximizes server performance.

---

## DESCRIPTION

QUERY BEST TILE obtains the optimal size for a tile pattern that maximizes server performance. The best size is the size that can be tiled in the least amount of time.

You specify the desired size of the tile in the **width** and **height** arguments. The server returns the best size, which is closest to the size specified, in the **width\_return** and **height\_return** arguments.

You can also use the QUERY BEST SIZE routine to determine the best size for a tile pattern.

## Graphics Context Routines

### QUERY BEST TILE

---

#### X ERRORS

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## SET ARC MODE

Sets the arc mode to the value specified.

**VAX FORMAT**    **X\$SET\_ARC\_MODE**  
(*display, gc\_id, arc\_mode*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
arc_mode	longword	longword	read	reference

**MIT C FORMAT**    **XSetArcMode**  
(*display, gc\_id, arc\_mode*)

**argument  
information**

```
XSetArcMode(display, gc_id, arc_mode)
Display *display;
GC gc_id;
int arc_mode;
```

### ARGUMENTS

***display***  
The display information originally returned by OPEN DISPLAY.

***gc\_id***  
The identifier of the graphics context where the arc mode member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***arc\_mode***  
Specifies how an arc will be filled in a subsequent graphics request. The predefined values for **arc\_mode** are as follows:

VAX	C	Description
X\$C_ARC_CHORD	ArcChord	Only the chord area of the arc is filled.
X\$C_ARC_PIE_SLICE	ArcPieSlice	The triangular area defined by the arc center and the arc endpoints is filled.

Other values specified in this argument are not valid.

# Graphics Context Routines

## SET ARC MODE

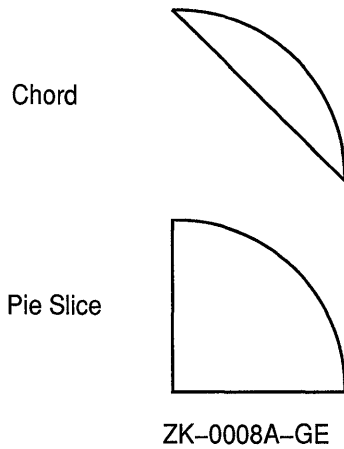
---

### DESCRIPTION

SET ARC MODE changes the value for the arc mode member in the graphics context data structure. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

The arc mode specifies which portion of an arc to draw: either a pie slice or a chord. Refer to Figure 5–8 for an illustration of the arc fill options.

**Figure 5–8 Arc Fill Options**



---

You can also use CHANGE GC to change the arc mode.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

## SET BACKGROUND

Sets the background color index to the value specified.

**VAX FORMAT**     **X\$SET\_BACKGROUND**  
                           (*display, gc\_id, background*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
background	uns longword	uns longword	read	reference

**MIT C FORMAT**     **XSetBackground**  
                           (*display, gc\_id, background*)

**argument  
information**

```
XSetBackground(display, gc_id, background)
Display *display;
GC gc_id;
unsigned long background;
```

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the background member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***background***

The new value of the background member. The background value is a color index to a color map containing the color definition for the background.

### DESCRIPTION

SET BACKGROUND changes the value for the background member in the GC values data structure. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

The background member value is the color index to a color map containing the color definition for the background. The color index you specify in **background** must have been returned by one of the color routines (ALLOC COLOR CELLS, ALLOC COLOR PLANES, ALLOC COLOR, or ALLOC NAMED COLOR) or by one of the display information routines

## Graphics Context Routines

### SET BACKGROUND

(BLACK PIXEL or WHITE PIXEL). For more information on working with colors, see Chapter 12.

You can also use SET STATE or CHANGE GC to change the background. When you use SET STATE, you must also change the foreground, function, and plane mask members.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

## SET CLIP MASK

Sets the clip mask of a graphics context to the specified pixmap.

**VAX FORMAT**     **X\$SET\_CLIP\_MASK**  
                          (*display, gc\_id, pixmap\_id*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
pixmap_id	identifier	uns longword	read	reference

**MIT C FORMAT**     **XSetClipMask**  
                          (*display, gc\_id, pixmap\_id*)

**argument  
information**

```
XSetClipMask(display, gc_id, pixmap_id)
Display *display;
GC gc_id;
Pixmap pixmap_id;
```

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the clip mask will be changed. This identifier was originally returned by CREATE PIXMAP.

***pixmap\_id***

The identifier of the pixmap containing the new clip mask.

### DESCRIPTION

SET CLIP MASK sets the clip mask of a graphics context to the specified pixmap. Use a clip mask when you want to clip a nonrectangular area. The area clipped out is represented by zeros; the area maintained is represented by ones.

If you specify None, all pixels are drawn.

After you identify a pixmap as a clip mask, you should not write into that pixmap. Otherwise, the results are undefined.



## Graphics Context Routines

### SET CLIP MASK

When you want to clip a rectangular area, use `SET CLIP RECTANGLES`. You can also change the clip mask with `CHANGE GC`.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

---

## SET CLIP ORIGIN

Changes the clip origin members in the specified graphics context.

---

**VAX FORMAT**     **X\$SET\_CLIP\_ORIGIN**  
(*display, gc\_id, clip\_x\_coord, clip\_y\_coord*)

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
clip_x_coord	longword	longword	read	reference
clip_y_coord	longword	longword	read	reference

---

**MIT C FORMAT**     **XSetClipOrigin**  
(*display, gc\_id, clip\_x\_coord, clip\_y\_coord*)

---

**argument  
information**

```
XSetClipOrigin(display, gc_id, clip_x_coord, clip_y_coord)
    Display *display;
    GC gc_id;
    int clip_x_coord, clip_y_coord;
```

---

**ARGUMENTS**     ***display***  
The display information originally returned by OPEN DISPLAY.

***gc\_id***  
The identifier of the graphics context where clip origin coordinates will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***clip\_x\_coord***  
The new x-coordinate of the clip origin. The default value is 0.

***clip\_y\_coord***  
The new y-coordinate of the clip origin. The default value is 0.

## Graphics Context Routines

### SET CLIP ORIGIN

---

#### DESCRIPTION

SET CLIP ORIGIN changes the x- and y-coordinate values for the clip origin. The clip origin is aligned with the origin of the window or pixmap (drawable) involved in the clip operation. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

You can also change the clip origin members with CHANGE GC.

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

## SET CLIP RECTANGLES

Changes the clip origin members in the specified graphics context and specifies rectangular clip areas.

**VAX FORMAT**    **X\$SET\_CLIP\_RECTANGLES**  
*(display, gc\_id, clip\_x\_coord, clip\_y\_coord, rectangles, num\_rectangles, ordering)*

**argument information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
clip_x_coord	longword	longword	read	reference
clip_y_coord	longword	longword	read	reference
rectangles	record	x\$rectangle	read	reference
num_rectangles	longword	longword	read	reference
ordering	longword	longword	read	reference

**MIT C FORMAT**    **XSetClipRectangles**  
*(display, gc\_id, clip\_x\_coord, clip\_y\_coord, rectangles, num\_rectangles, ordering)*

**argument information**

```
XSetClipRectangles (display, gc_id, clip_x_coord, clip_y_coord,
                    rectangles, num_rectangles, ordering)
Display *display;
GC gc_id;
int clip_x_coord, clip_y_coord;
XRectangle rectangles[];
int num_rectangles;
int ordering;
```

**ARGUMENTS**

- display***  
The display information originally returned by OPEN DISPLAY.
- gc\_id***  
The identifier of the graphics context where the clip origin will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

## Graphics Context Routines

### SET CLIP RECTANGLES

#### ***clip\_x\_coord***

The new x-coordinate of the clip origin. The default value is 0. The x- and y-coordinates define the clip origin.

#### ***clip\_y\_coord***

The new y-coordinate of the clip origin. The default value is 0. The x- and y-coordinates define the clip origin.

#### ***rectangles***

A pointer to an array of rectangle data structures where each element defines a clip area. Within each rectangle data structure, you specify the rectangle size (width and height) and position of the upper left corner (x- and y-coordinates). The x- and y-coordinates are interpreted relative to the clip origin. The array length is specified by **num\_rectangles**.

#### ***num\_rectangles***

The number of rectangles to be specified in the graphics context. This value defines the length of the array in **rectangles**.

#### ***ordering***

Specifies the ordering relationship of the clip rectangles.

The predefined values for **ordering** are as follows:

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_UN_SORTED	UnSorted	Rectangles are in arbitrary order.
X\$C_Y_SORTED	YSorted	Rectangles are nondecreasing in the y-coordinate of their origin.
X\$C_Y_X_SORTED	YXSorted	Rectangles are nondecreasing in the y-coordinate of their origin. For rectangles with an equal y-coordinate, they are nondecreasing in the x-coordinate of their origin.
X\$C_Y_X_BANDED	YXBanded	Rectangles are nondecreasing in the y-coordinate of their origin for every possible y scan line, all rectangles that include that scan line have an identical y-coordinate.

Other values specified in this argument are not valid.

Server performance improves if the rectangles are ordered (UnSorted would result in the lowest performance, then YSorted, then YXSorted, with YXBanded having the best performance). However, if the rectangles are not in the same order as specified in this argument, the graphics results will be undefined.

## Graphics Context Routines

### SET CLIP RECTANGLES

---

#### DESCRIPTION

SET CLIP RECTANGLES changes the clip origin in the specified graphics context. It also specifies the size and position of rectangular clip areas.

The output is clipped to remain contained within the rectangles. The clip origin is aligned with the origin of the destination drawable that is specified in a graphics request. The coordinates are interpreted relative to the clip origin. The rectangles should not intersect, or graphics results will be undefined.

If you know the correct order of the rectangles, you can specify the order in **ordering** to improve server performance. If you specify an incorrect order, the graphics results are undefined.

If you want to effectively disallow any output, the list of rectangles in **rectangles** can be empty.

If you want to clip a nonrectangular area, use a clip mask. Use the SET CLIP MASK routine to specify the pixmap in the graphics context containing the clip mask.

You cannot use CHANGE GC to specify clip rectangles, although you can use CHANGE GC to specify the clip origin.

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

# Graphics Context Routines

## SET DASHES

---

### SET DASHES

Changes the values for the dash offset and the dash list.

---

#### VAX FORMAT

#### **X\$SET\_DASHES**

*(display, gc\_id, dash\_offset, dash\_list, dash\_list\_len)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
dash_offset	longword	longword	read	reference
dash_list	byte	byte	read	reference
dash_list_len	longword	longword	read	reference

---

---

#### MIT C FORMAT

#### **XSetDashes**

*(display, gc\_id, dash\_offset, dash\_list, dash\_list\_len)*

#### argument information

```
XSetDashes(display, gc_id, dash_offset, dash_list, dash_list_len)
Display *display;
GC gc_id;
int dash_offset;
char dash_list[];
int dash_list_len;
```

---

#### ARGUMENTS

##### ***display***

The display information originally returned by OPEN DISPLAY.

##### ***gc\_id***

The identifier of the graphics context where the dash members will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

##### ***dash\_offset***

The pixel within the dash length sequence, defined in **dash\_list**, to start drawing a dashed line. A **dash\_offset** of zero would start at the beginning of the dash length sequence. A **dash\_offset** of 5 would start at the fifth pixel. Refer to Figure 5-9.

# Graphics Context Routines

## SET DASHES

### ***dash\_list***

A pointer to an array that lists the length, in number of pixels, for each dash. The initial and alternating elements in the array (elements 0, 2, 4, and so forth) are the even dashes. The other elements in the array (in other words, elements 3, 5, and so forth) reference the odd dashes.

The values of all elements must be nonzero. The number of elements (in other words, the length of the array) is defined by **n**. The default value is 4.

This argument cannot be empty, otherwise the routine will not complete successfully.

### ***dash\_list\_len***

The number of dash lengths specified. This value defines the length of the array in **dash\_list**.

---

## DESCRIPTION

SET DASHES changes the dash members in the specified graphics context.

The dash list specifies the length, in pixels, of each element in the dash line. For example, the dash list array might have the following values:

Array Element	Value
0	5
1	10
2	3
3	5
4	10
5	3

The first dash in the line (even dash 0) is 5 pixels long, the second dash in the line is 10 pixels long (odd dash 1), and so forth. Refer to Figure 5-9.

**Figure 5-9 Dash Offset and Dash List**

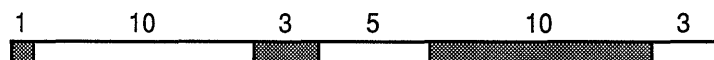
---

Dash List: 5,10,3,5,10,3

Dash Offset = 0



Dash Offset = 4



ZK-0009A-GE

---

The actual dash line is likely to have more dashes than are represented in the dash list. In this case, the list is simply reread so that the next dash



# Graphics Context Routines

## SET DASHES

has the length specified in array element 0, and so forth. The dash list array will continue to be read until the dash line is complete.

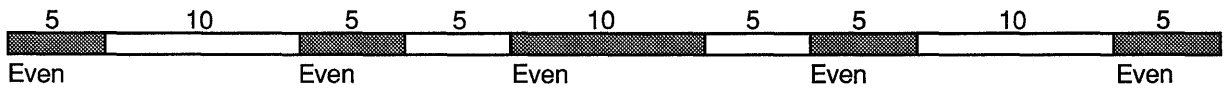
You can specify an odd number of dash lengths in the dash list. For example, the dash list array might have the following values:

Array Element	Value
0	5
1	10
2	5

If this dash line has nine dashes, the lengths are as shown in Figure 5–10.

**Figure 5–10 Odd Dash List**

Dash List 5, 10, 5



ZK-0017A-GE

The dash offset specifies where to start reading, within the dash list, the lengths of the dashes. For example, if the dash offset is 3, the first length in the dash list is the third from the start. In the preceding dash list example, a dash offset of 3 means that the length of the first dash is 2.

A dashed line is continuous through a specified path. When dashed lines are joined, they are joined according to the join style specified in the graphics context. When a dashed line is ended (in other words, a cap\_style from the graphics context is applied to an endpoint), the dash line is reset to the dash offset.

## X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

## SET FILL RULE

Sets the fill rule in the graphics context to the value specified.

**VAX FORMAT**    **X\$SET\_FILL\_RULE**  
                  (*display, gc\_id, fill\_rule*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
fill_rule	longword	longword	read	reference

**MIT C FORMAT**    **XSetFillRule**  
                  (*display, gc\_id, fill\_rule*)

**argument  
information**

```
XSetFillRule(display, gc_id, fill_rule)
    Display *display;
    GC gc_id;
    int fill_rule;
```

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the fill rule member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***fill\_rule***

The fill rule that you want to set for the specified graphics context. The fill rule specifies which pixels are considered to be inside of a polygon for a FILL POLYGON routine. Those inside the polygon are then displayed creating the fill. There are two predefined values:

## Graphics Context Routines

### SET FILL RULE

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_EVEN_ODD_RULE	EvenOddRule	A pixel is considered to be inside the polygon when a ray drawn from the pixel intersects the polygon lines an odd number of times. If the ray drawn from the pixel intersects an even number of times, the pixel is not drawn.
X\$C_WINDING_RULE	WindingRule	A pixel is considered to be inside when a ray with the point as origin crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise line is one that crosses the ray from left to right as observed from the origin. A counterclockwise line is one which crosses the ray from right to left as observed from the origin.

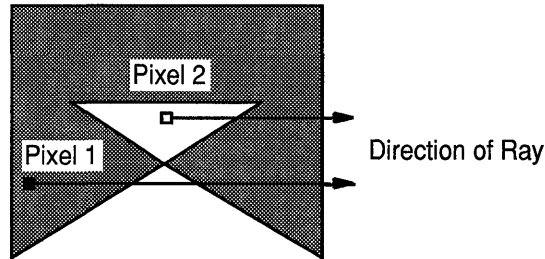
Refer to Figure 5–11 for an illustration of the Even Odd and Winding options.

# Graphics Context Routines

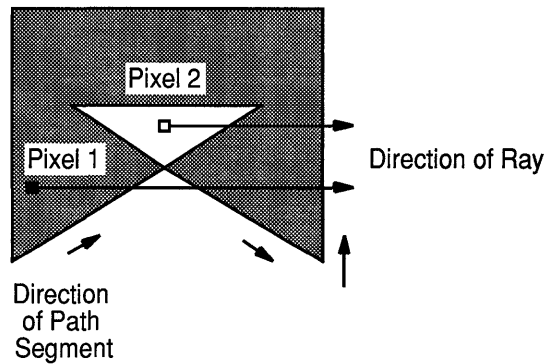
## SET FILL RULE

Figure 5–11 Fill Rules

### Even Odd



### Winding



ZK-0071A-GE

## DESCRIPTION

SET FILL RULE changes the value for the fill rule in the graphics context data structure.

You can also use CHANGE GC to change the fill rule.

## X ERRORS

VAX	C	Description
X\$c_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$c_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

# Graphics Context Routines

## SET FILL STYLE

---

## SET FILL STYLE

Sets the fill style in the graphics context to the value specified.

---

**VAX FORMAT**     **X\$SET\_FILL\_STYLE**  
*(display, gc\_id, fill\_style)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
fill_style	longword	longword	read	reference

---

---

**MIT C FORMAT**     **XSetFillStyle**  
*(display, gc\_id, fill\_style)*

**argument  
information**

```
XSetFillStyle(display, gc_id, fill_style)
Display *display;
GC gc_id;
int fill_style;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the fill style member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***fill\_style***

The fill style for the space within the bounding box of a line, the even dash of an On Off Dash or Double Dash line, for an odd dash of a Double Dash line, for text lines, and for all fill requests. The predefined values for **fill\_style** are as follows:

## Graphics Context Routines

### SET FILL STYLE

VAX	C	Description
X\$C_FILL_SOLID	FillSolid	Use the color index specified by the graphics context foreground member for each pixel in the fill area. For the odd dash, use the color index specified by the graphics context background member for each pixel in the odd dash fill area.
X\$C_FILL_TILED	FillTiled	Use the tile pattern as specified by the graphics context tile member.
X\$C_FILL_OPAQUE_STIPPLED	FillOpaqueStippled	Use the stipple pattern as specified by the graphics context stipple member. Where the value of 1 appears in the stipple pattern, use the foreground color index for the corresponding pixel; where the value of zero appears, use the background color index for the corresponding pixel.
X\$C_FILL_STIPPLED	FillStippled	Use the stipple pattern as specified by the graphics context. Where the value of 1 appears in the stipple pattern, you can write to the corresponding pixel; where the value of zero appears, you cannot write to the corresponding pixel. Use the background color index for the corresponding pixel.

Other values specified in this argument are not valid.

---

**DESCRIPTION** SET FILL STYLE changes the fill style in the graphics context data structure.

The fill style specifies how to fill in bounding boxes in lines or figures. You can also use CHANGE GC to change the fill style.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

# Graphics Context Routines

## SET FONT

---

### SET FONT

Changes the identifier of the font in the specified graphics context.

---

**VAX FORMAT**    **X\$SET\_FONT**  
*(display, gc\_id, font\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
font_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetFont**  
*(display, gc\_id, font\_id)*

**argument  
information**

```
XSetFont(display, gc_id, font_id)
Display *display;
GC gc_id;
Font font_id;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the font identifier will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***font\_id***

The identifier of the font that will be set in the graphics context. The font identifier was originally returned by GET FONT.

---

### DESCRIPTION

SET FONT changes the font identifier in the graphics context. Before you change the font identifier, you must have already loaded the font and generated a font identifier using GET FONT and LOAD FONT.

You can also use CHANGE GC to change the font identifier.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

---



# Graphics Context Routines

## SET FOREGROUND

---

# SET FOREGROUND

Sets the foreground color index to the value specified.

---

**VAX FORMAT**     **X\$SET\_FOREGROUND**  
*(display, gc\_id, foreground)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
foreground	uns longword	uns longword	read	reference

---

---

**MIT C FORMAT**     **XSetForeground**  
*(display, gc\_id, foreground)*

**argument  
information**

```
XSetForeground(display, gc_id, foreground)
Display *display;
GC gc_id;
unsigned long foreground;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the foreground member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***foreground***

The new value of the foreground member. The foreground value is a color index to a color map containing the color definition for the foreground color. The default value is zero.

---

**DESCRIPTION**

SET FOREGROUND changes the value for the foreground in the GC values data structure.

The foreground member value is the color index to a color map containing the color definition for the foreground. The color index you specify in **foreground** must have been returned by one of the color routines (ALLOC COLOR CELLS, ALLOC COLOR PLANES, ALLOC COLOR, or ALLOC NAMED COLOR) or by one of the display information routines (BLACK

## Graphics Context Routines

### SET FOREGROUND

PIXEL or WHITE PIXEL). For more information about working with colors, see Chapter 12.

You can also use SET STATE or CHANGE GC to change the foreground member. When you use SET STATE, you must also change the background, function, and plane mask members.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

# Graphics Context Routines

## SET FUNCTION

---

# SET FUNCTION

Sets the function value to the value specified.

---

### VAX FORMAT **X\$SET\_FUNCTION** (*display, gc\_id, function*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
function	longword	longword	read	reference

---

---

### MIT C FORMAT **XSetFunction** (*display, gc\_id, function*)

#### argument information

```
XSetFunction(display, gc_id, function)
Display *display;
GC gc_id;
int function;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***gc\_id***

The identifier of the graphics context where the function member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

#### ***function***

The new value for the function member of the graphics context. The function member describes how the new destination bits are to be computed from the source bits and the old destination bits. Table 5-7 lists the valid values for function. The default value is 3 for GX Copy.

# Graphics Context Routines

## SET FUNCTION

**Table 5–7 Graphics Context Codes for Function Member**

Hex Value	VAX Function Name	C Function Name	Operation
0	X\$C_GX_CLEAR	GXclear	0
1	X\$C_GX_AND	GXand	src AND dst
2	X\$C_GX_AND_REVERSE	GXandReverse	src AND NOT dst
3	X\$C_GX_COPY	GXcopy	src
4	X\$C_GX_AND_INVERTED	GXandInverted	(NOT src) AND dst
5	X\$C_GX_NOOP	GXnoop	dst
6	X\$C_GX_XOR	GXxor	src XOR dst
7	X\$C_GX_OR	GXor	src OR dst
8	X\$C_GX_NOR	GXnor	(NOT src) AND NOT dst
9	X\$C_GX_EQUIV	GXequiv	(NOT src) XOR dst
A	X\$C_GX_INVERT	GXinvert	NOT dst
B	X\$C_GX_OR_REVERSE	GXorReverse	src OR NOT dst
C	X\$C_GX_COPY_INVERTED	GXcopyInverted	NOT src
D	X\$C_GX_OR_INVERTED	GXorInverted	(NOT src) OR dst
E	X\$C_GX_NAND	GXnand	(NOT src) OR NOT dst
F	X\$C_GX_SET	GXset	1

### DESCRIPTION

SET FUNCTION changes the value for the function member in the GC values data structure. The function describes how the new destination bits are computed from the source bits and the old destination bits.

You can also change the function using SET STATE or CHANGE GC. When you use SET STATE, you also change the foreground, background, and plane mask.

### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

## Graphics Context Routines

### SET GRAPHICS EXPOSURES

---

## SET GRAPHICS EXPOSURES

Changes the graphics exposures.

---

**VAX FORMAT**    **X\$SET\_GRAPHICS\_EXPOSURES**  
*(display, gc\_id, graphics\_exposures)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
graphics_exposures	boolean	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetGraphicsExposures**  
*(display, gc\_id, graphics\_exposures)*

**argument  
information**

```
XSetGraphicsExposures(display, gc_id, graphics_exposures)
    Display *display;
    GC gc_id;
    Boolean graphics_exposures;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the graphics exposures member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***graphics\_exposures***

A graphics exposures member that specifies whether to generate graphics exposures events when using the COPY AREA or COPY PLANE routines. When true, the events are generated. When false, the events are not generated.

---

**DESCRIPTION**

SET GRAPHICS EXPOSURES changes the value for the graphics exposures member in the graphics context.

You can also use CHANGE GC to change graphics exposures.

## Graphics Context Routines

### SET GRAPHICS EXPOSURES

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

# Graphics Context Routines

## SET LINE ATTRIBUTES

---

# SET LINE ATTRIBUTES

Changes the line drawing members in the graphics context.

---

**VAX FORMAT**    **X\$SET\_LINE\_ATTRIBUTES**  
(*display, gc\_id, line\_width, line\_style, cap\_style, join\_style*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
line_width	longword	longword	read	reference
line_style	longword	longword	read	reference
cap_style	longword	longword	read	reference
join_style	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XSetLineAttributes**  
(*display, gc\_id, line\_width, line\_style, cap\_style, join\_style*)

**argument  
information**

---

```
XSetLineAttributes(display, gc_id, line_width, line_style,  
                  cap_style, join_style)  
  
Display *display;  
GC gc_id;  
unsigned int line_width;  
int line_style;  
int cap_style;  
int join_style;
```

---

**ARGUMENTS**

***display***  
The display information originally returned by OPEN DISPLAY.

***gc\_id***  
The identifier of the graphics context where the line attributes will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

## Graphics Context Routines

### SET LINE ATTRIBUTES

#### ***line\_width***

Specifies the line width member. The line width defines how wide lines are drawn, in terms of number of pixels. The default value is zero, specifying the special case of a line one pixel wide. The maximum value for this member is limited by the size of the window you are working in. Refer to the *VMS DECwindows Xlib Programming Volume* for more information on line widths.

#### ***line\_style***

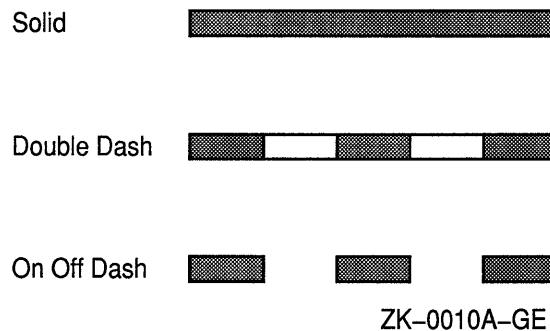
Specifies the line style member. Line style defines the pattern of a line. The predefined values for ***line\_style*** are as follows:

VAX	C	Description
X\$C_LINE_SOLID	LineSolid	A solid line
X\$C_LINE_DOUBLE_DASH	LineDoubleDash	A dashed line where the even dashes are different from the odd dashes
X\$C_LINE_OFF_DASH	LineOnOffDash	A dashed line where the even dashes are drawn, but the odd dashes are empty

Other values specified in this argument are not valid.

The default value specifies the Solid pattern. Refer to Figure 5–12 for an illustration of these styles.

**Figure 5–12 Line Styles**



#### ***cap\_style***

Specifies the cap style member, which defines how the endpoints of a path are drawn. The predefined values for ***cap\_style*** are as follows:

VAX	C	Description
X\$C_CAP_NOT_LAST	CapNotLast	A square endpoint with no projection. This style is used for line widths of zero or 1 only.
X\$C_CAP_BUTT	CapButt	A square endpoint, where the square is perpendicular to the slope of the line, with no projection beyond the endpoint.



# Graphics Context Routines

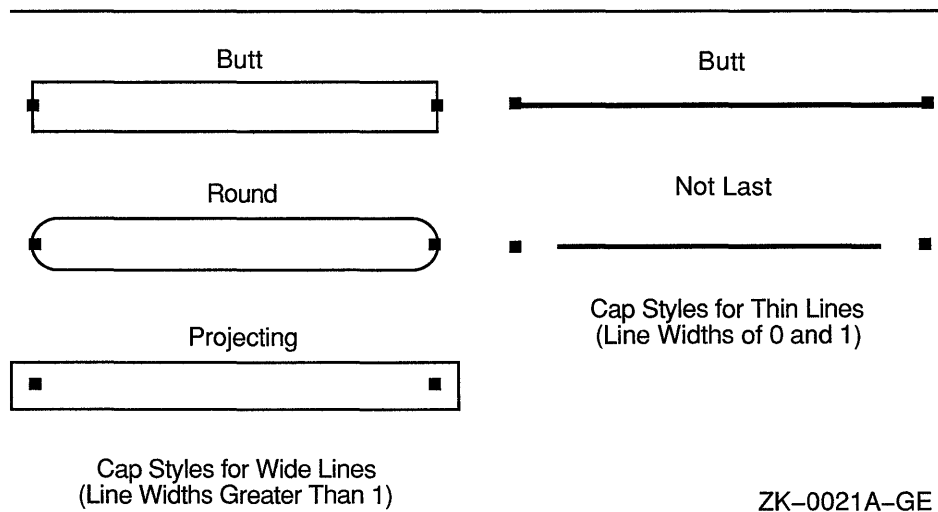
## SET LINE ATTRIBUTES

VAX	C	Description
X\$C_CAP_ROUND	CapRound	A circular arc endpoint. The diameter of the arc is equal to the line width. The arc is centered on the endpoint. This style is not used for line widths of zero or 1.
X\$C_CAP_PROJECTING	CapProjecting	A square endpoint with the path extended beyond the endpoint. The distance extended is equal to one-half the line width. This style is not used for line widths of zero or 1.

Other values specified in this argument are not valid.

The default value is the Butt pattern. Refer to Figure 5–13 for an illustration of these styles.

**Figure 5–13 Cap Styles**



### ***join\_style***

The *join\_style*. The join style specifies how corners are drawn for wide lines. The predefined values for **join\_style** are as follows:

VAX	C	Description
X\$C_JOIN_MITER	JoinMiter	The outer edges of the lines are extended to meet at an angle.
X\$C_JOIN_ROUND	JoinRound	A circular arc corner. The diameter of the arc is equal to the line width. The arc is centered on the point where the two lines join.
X\$C_JOIN_BEVEL	JoinBevel	A square corner with the triangular notch of the corner filled in.

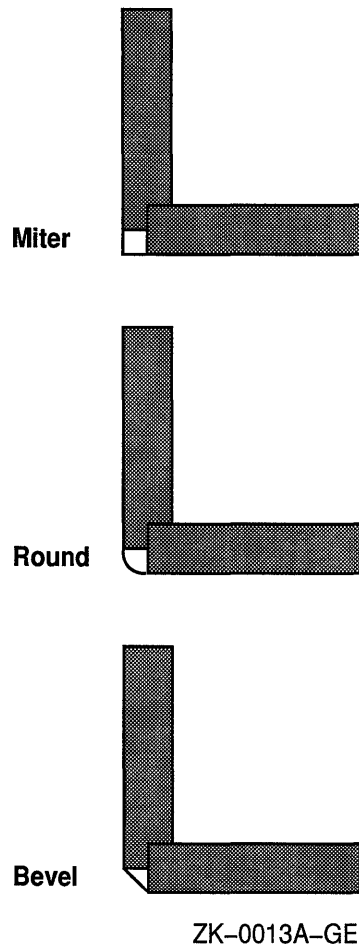
Other values specified in this argument are not valid.

## Graphics Context Routines

### SET LINE ATTRIBUTES

The default value is the Miter pattern. Refer to Figure 5-14 for an illustration of these styles.

**Figure 5-14 Join Styles**



#### **DESCRIPTION**

SET LINE ATTRIBUTES changes the values for the following line drawing members in the specified graphics context:

- Line width
- Line style
- Cap style (how endpoints are drawn)
- Join Style (how corners are drawn)

The patterns for the line style, cap style, and join style are illustrated in the argument descriptions. The join style is used only when lines are joined within a single graphics request.

You can also change the line attributes with CHANGE GC.

## Graphics Context Routines

### SET LINE ATTRIBUTES

---

#### X ERRORS

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

---

## SET PLANE MASK

Sets the plane mask to the value specified.

---

**VAX FORMAT**    **X\$SET\_PLANE\_MASK**  
(*display, gc\_id, plane\_mask*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
plane_mask	mask_longword	uns longword	read	reference

---



---

**MIT C FORMAT**    **XSetPlaneMask**  
(*display, gc\_id, plane\_mask*)

**argument  
information**

```
XSetPlaneMask(display, gc_id, plane_mask)
Display *display;
GC gc_id;
unsigned long plane_mask;
```

---

### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the plane mask member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***plane\_mask***

A bit mask that specifies which planes will be modified.

The plane mask specifies which planes of the display will be modified. For monochrome displays, there is only one plane. Within the plane mask, the least significant bit represents this plane. For displays with additional planes, the bits that represent those planes occupy more significant bits.

When a plane is to be modified, its corresponding bit in **plane\_mask** is set. The default value is all ones, specifying that all planes can be modified.

# Graphics Context Routines

## SET PLANE MASK

---

### DESCRIPTION

SET PLANE MASK changes the value for the plane mask member in the GC values data structure. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

The plane mask member identifies which planes of a display can be modified. There is one bit in the plane mask per plane. The least significant bit represents the first plane. As planes are added, they are represented by more significant bits.

You can also change the plane mask with CHANGE GC or SET STATE. When you use SET STATE, you must also change the foreground, background, and function members.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

## SET STATE

Changes values for the foreground, background, plane mask, and function members of a graphics context.

### VAX FORMAT

### **X\$SET\_STATE**

*(display, gc\_id, foreground, background, func, plane\_mask)*

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
foreground	longword	longword	read	reference
background	longword	longword	read	reference
func	longword	longword	read	reference
plane_mask	mask_longword	uns longword	read	reference

### MIT C FORMAT

### **XSetState**

*(display, gc\_id, foreground, background, func, plane\_mask)*

#### argument information

```
XSetState(display, gc_id, foreground, background, func,
          plane_mask)
Display *display;
GC gc_id;
unsigned long foreground, background;
int func;
unsigned long plane_mask;
```

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***gc\_id***

The identifier of the graphics context where the new values will be set. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

# Graphics Context Routines

## SET STATE

### *foreground*

The new value for the foreground member of the graphics context. The foreground value is a color index to a color map containing the color definition for the foreground. If the value specified is out of range, it is truncated to the appropriate number of bits.

### *background*

The new value for the background member of the graphics context. The background value is a color index to a color map containing the color definition for the background. If the value specified is out of range, it is truncated to the appropriate number of bits.

### *func*

The new value for the function member of the graphics context.

The function member describes how the new destination bits are computed from the source bits and the old destination bits. Table 5–8 lists the valid values for **func**. The default value is 3 for GX Copy.

**Table 5–8 Graphics Context Codes for Function Member**

Hex Value	VAX Function Name	C Function Name	Operation
0	X\$C_GX_CLEAR	GXclear	0
1	X\$C_GX_AND	GXand	src AND dst
2	X\$C_GX_AND_REVERSE	GXandReverse	src AND NOT dst
3	X\$C_GX_COPY	GXcopy	src
4	X\$C_GX_AND_INVERTED	GXandInverted	(NOT src) AND dst
5	X\$C_GX_NOOP	GXnoop	dst
6	X\$C_GX_XOR	GXxor	src XOR dst
7	X\$C_GX_OR	GXor	src OR dst
8	X\$C_GX_NOR	GXnor	(NOT src) AND NOT dst
9	X\$C_GX_EQUIV	GXequiv	(NOT src) XOR dst
A	X\$C_GX_INVERT	GXinvert	NOT dst
B	X\$C_GX_OR_REVERSE	GXorReverse	src OR NOT dst
C	X\$C_GX_COPY_INVERTED	GXcopyInverted	NOT src
D	X\$C_GX_OR_INVERTED	GXorInverted	(NOT src) OR dst
E	X\$C_GX_NAND	GXnand	(NOT src) OR NOT dst
F	X\$C_GX_SET	GXset	1

### *plane\_mask*

A bit mask that specifies which planes will be modified.

The plane mask specifies which planes of the display will be modified. For monochrome displays, there is only one plane. Within the plane mask, the least significant bit represents this plane. For displays with additional planes, the bits that represent those planes occupy more significant bits.

# Graphics Context Routines

## SET STATE

When a plane is to be modified, its corresponding bit in **plane\_mask** is set. The default value is all ones, specifying that all planes can be modified.

---

### DESCRIPTION

SET STATE changes values in the GC values data structure for four members:

- Foreground
- Background
- Function
- Planes

Use the graphics context identifier to refer to the specific graphics context that you want to change.

Use this routine when you want to change all these values in the same graphics context. If you want to change individual values, use the routines SET FOREGROUND to change the foreground; SET BACKGROUND to change the background; SET FUNCTION to change the function value; or SET PLANE MASK to change planes.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---



# Graphics Context Routines

## SET STIPPLE

---

### SET STIPPLE

Changes the pixmap identifier of the stipple pattern in the specified graphics context.

---

**VAX FORMAT**     **X\$SET\_STIPPLE**  
*(display, gc\_id, stipple\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
stipple_id	identifier	longword	read	reference

---

---

**MIT C FORMAT**     **XSetStipple**  
*(display, gc\_id, stipple\_id)*

**argument  
information**

```
XSetStipple(display, gc_id, stipple_id)
Display *display;
GC gc_id;
Pixmap stipple_id;
```

---

**ARGUMENTS**     ***display***  
The display information originally returned by OPEN DISPLAY.

***gc\_id***  
The identifier of the graphics context where the pixmap identifier of the stipple pattern will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***stipple\_id***  
The identifier of the pixmap containing the stipple pattern. The pixmap identifier was originally returned by CREATE PIXMAP.

---

**DESCRIPTION**     SET STIPPLE changes the pixmap identifier in the specified graphics context to the identifier of a pixmap containing the desired stipple pattern. The default value is a pixmap of unspecified size filled with values of 1.

## Graphics Context Routines

### SET STIPPLE

The stipple pixmap can be any size, although some sizes might be faster to use than others. Use the `QUERY BEST STIPPLE` routine to determine the best size. The stipple pixmap must have the same depth as the graphics context.

You can also use `CHANGE GC` to specify the pixmap identifier for a stipple pattern.

---

## X ERRORS

VAX	C	Description
<code>X\$C_BAD_ALLOC</code>	<code>BadAlloc</code>	The server did not allocate the requested resource for any cause.
<code>X\$C_BAD_GC</code>	<code>BadGC</code>	A value that you specified for a graphics context argument does not name a defined graphics context.
<code>X\$C_BAD_MATCH</code>	<code>BadMatch</code>	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
<code>X\$C_BAD_PIXMAP</code>	<code>BadPixmap</code>	A value that you specified for a pixmap argument does not name a defined pixmap.

## Graphics Context Routines

### SET SUBWINDOW MODE

---

## SET SUBWINDOW MODE

Changes the value for the subwindow mode in the graphics context.

---

**VAX FORMAT**    **X\$SET\_SUBWINDOW\_MODE**  
*(display, gc\_id, subwindow\_mode)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
subwindow_mode	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XSetSubwindowMode**  
*(display, gc\_id, subwindow\_mode)*

**argument  
information**

```
XSetSubwindowMode(display, gc_id, subwindow_mode)
Display *display;
GC gc_id;
int subwindow_mode;
```

---

## ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the subwindow mode member will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***subwindow\_mode***

Specifies whether the source and destination windows are clipped by subwindows. The predefined values are as follows:

## Graphics Context Routines

### SET SUBWINDOW MODE

VAX	C	Description
X\$C_CLIP_BY_CHILDREN	ClipByChildren	The source and destination windows are clipped.
X\$C_INCLUDE_INFERIORS	IncludeInferiors	Inferiors clip neither source nor destination windows. This results in drawing through subwindows boundaries.

Other values specified in this argument are not valid.

---

**DESCRIPTION**     SET SUBWINDOW MODE changes the value for the subwindow mode in the graphics context.

You can also use CHANGE GC to change the subwindow mode.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

# Graphics Context Routines

## SET TILE

---

### SET TILE

Changes the pixmap identifier of the tile pattern in the specified graphics context.

---

**VAX FORMAT**    **X\$SET\_TILE**  
*(display, gc\_id, tile\_id)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
tile_id	identifier	uns longword	read	reference

---

---

**MIT C FORMAT**    **XSetTile**  
*(display, gc\_id, tile\_id)*

**argument  
information**

```
XSetTile(display, gc_id, tile_id)
Display *display;
GC gc_id;
Pixmap tile_id;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***gc\_id***

The identifier of the graphics context where the tile pixmap identifier will be changed. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

***tile\_id***

The identifier of the pixmap containing the tile pattern. The pixmap identifier was originally returned by CREATE PIXMAP.

---

**DESCRIPTION**

SET TILE changes the pixmap identifier in the specified graphics context to the identifier of a pixmap containing the desired tile pattern. The default value is a pixmap of unspecified size filled with the foreground pixel.

## Graphics Context Routines

### SET TILE

The tile pixmap can be any size, although some sizes might be faster to use than others. Use the `QUERY BEST TILE` to determine the best size. The tile pixmap must have the same root and depth as the graphics context.

The graphics context identifier was originally returned by `DEFAULT GC` or `CREATE GC`.

You can also use `CHANGE GC` to specify a tile pixmap identifier.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.

## Graphics Context Routines

### SET TS ORIGIN

---

## SET TS ORIGIN

Changes the x- and y-coordinates of the tile or stipple origin in the graphics context.

---

**VAX FORMAT**    **X\$SET\_TS\_ORIGIN**  
(*display, gc\_id, ts\_x\_coord, ts\_y\_coord*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
ts_x_coord	longword	longword	read	reference
ts_y_coord	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XSetTSTOrigin**  
(*display, gc\_id, ts\_x\_coord, ts\_y\_coord*)

**argument  
information**

```
XSetTSTOrigin(display, gc_id, ts_x_coord, ts_y_coord)
Display *display;
GC gc_id;
int ts_x_coord, ts_y_coord;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***gc\_id***

The identifier of the graphics context where the tile or stipple origin is located. The graphics context identifier was originally returned by DEFAULT GC or CREATE GC.

### ***ts\_x\_coord***

The x-coordinate of the tile or stipple origin to be changed. The x- and y-coordinates define where the tile or stipple pattern is aligned with the origin of the drawable.

### ***ts\_y\_coord***

The y-coordinate of the tile or stipple origin to be changed. The x- and y-coordinates define where the tile or stipple pattern is aligned with the origin of the drawable.

---

**DESCRIPTION**

SET TS ORIGIN changes the coordinates of the tile or stipple origin in the graphics context. The identifier of the graphics context was originally returned by DEFAULT GC or CREATE GC.

The tile or stipple origin defines the point within the tile or stipple pixmap that is aligned with the drawable that will be tiled or stippled.

When graphics requests call for tiling or stippling, the parent's origin is interpreted to whatever destination drawable is specified in the graphics context.

You can also change the coordinate values with CHANGE GC.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.





# 6

## Graphics Routines

Use the graphics routines to complete the following graphics operations:

- Clearing areas
- Copying areas
- Drawing points
- Drawing lines
- Drawing rectangles
- Drawing arcs
- Filling rectangles, polygons, and arcs
- Creating and manipulating images

For concepts related to graphics routines and information on how to use graphics routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 6–1.

**Table 6–1 Graphics Routines**

Routine Name	Description
ADD PIXEL	Increments each pixel in a pixmap by a constant value.
CLEAR AREA	Clears a rectangular area in a window.
CLEAR WINDOW	Clears an entire window.
COPY AREA	Copies a specified rectangular area from one window or pixmap (drawable) to another drawable.
COPY PLANE	Copies a rectangular area from one plane of a window or pixmap (drawable) to another drawable.
CREATE IMAGE	Specifies the size of the image and allocates sufficient memory for the image data structure.
DRAW ARC	Draws one arc in the specified window or pixmap.
DRAW ARCS	Draws more than one arc in the specified window or pixmap.
DRAW LINE	Draws one line between two points in the specified window or pixmap.
DRAW LINES	Draws more than one connected line in the specified drawable.
DRAW POINT	Draws a point in the specified window or pixmap.
DRAW POINTS	Draws more than one point in the specified drawable.

(continued on next page)

# Graphics Routines

**Table 6–1 (Cont.) Graphics Routines**

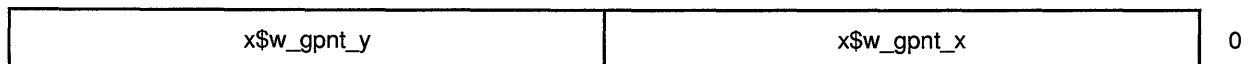
<b>Routine Name</b>	<b>Description</b>
DRAW RECTANGLE	Draws the outline of one rectangle in the specified drawable.
DRAW RECTANGLES	Draws the outline of more than one rectangle in the specified drawable.
DRAW SEGMENTS	Draws more than one line in the specified drawable. The lines are not connected.
FILL ARC	Fills in either a pie slice or chord area of an arc in the specified drawable.
FILL ARCS	Fills in either a pie slice or a chord area of more than one arc in the specified window or pixmap.
FILL POLYGON	Fills a polygon area in a specified drawable.
FILL RECTANGLE	Fills the area defined by a rectangular outline in the specified drawable.
FILL RECTANGLES	Fills the areas defined by rectangular outlines in the specified drawable.
GET IMAGE	Returns the contents of a rectangle to the specified drawable.
GET PIXEL	Obtains the value of one pixel stored in an image.
PUT IMAGE	Combines the image in memory with the image in a specified rectangle on the specified drawable.
PUT PIXEL	Changes one pixel value within the pixmap containing an image.
SUBIMAGE	Creates a new image from an existing image.

The graphics routines use several predefined data structures for points, unconnected line segments, rectangles, and arcs.

## 6.1 Point Data Structure

The point data structure defines a list of coordinates for use with the DRAW POINTS and DRAW LINES routines. In the case of DRAW POINTS, it affects those pixels of the specified coordinates, while in the case of DRAW LINES, it uses the specified coordinates as endpoints of consecutive, adjoining lines. The data structure for the VAX binding is shown in Figure 6–1, and information about members in the data structure is described in Table 6–2.

**Figure 6–1 Point Data Structure (VAX Binding)**



# Graphics Routines

## 6.1 Point Data Structure

**Table 6–2 Members of the Point Data Structure (VAX Binding)**

Member Name	Contents
X\$W_GPNT_X	Defines the x value of the coordinate of a point
X\$W_GPNT_Y	Defines the y value of the coordinate of a point

The data structure for the MIT C binding is shown in Figure 6–2, and information about members in the data structure is described in Table 6–3.

**Figure 6–2 Point Data Structure (MIT C Binding)**

```
typedef struct {
    short x,y;
}XPoint;
```

**Table 6–3 Members of the Point Data Structure (MIT C Binding)**

Member Name	Contents
x	Defines the x value of the coordinate of a point
y	Defines the y value of the coordinate of a point

## 6.2 Segment Data Structure

The segment data structure defines the x- and y-coordinates for two points. A line segment is drawn between one pair of points. The data structure for the VAX binding is shown in Figure 6–3.

**Figure 6–3 Segment Data Structure (VAX Binding)**

x\$w_gseg_y1	x\$w_gseg_x1	0
x\$w_gseg_y2	x\$w_gseg_x2	4

# Graphics Routines

## 6.2 Segment Data Structure

The MIT C binding segment data structure is shown in Figure 6–4.

**Figure 6–4 Segment Data Structure (MIT C Binding)**

```
typedef struct {  
    short x1,y1,x2,y2;  
} XSegment;
```

## 6.3 Rectangle Data Structure

The rectangle data structure defines the upper left corner and the width and height of a rectangle. The data structure for the VAX binding is shown in Figure 6–5, and information about members in the data structure is described in Table 6–4.

**Figure 6–5 Rectangle Data Structure (VAX Binding)**

x\$w_grec_y	x\$w_grec_x	0
x\$w_grec_height	x\$w_grec_width	4

**Table 6–4 Members of the Rectangle Data Structure (VAX Binding)**

Member Name	Contents
X\$W_GREC_X	Defines the x value of the rectangle origin
X\$W_GREC_Y	Defines the y value of the rectangle origin
X\$W_GREC_WIDTH	Defines the width of the rectangle
X\$W_GREC_HEIGHT	Defines the height of the rectangle

The data structure for the MIT C binding is shown in Figure 6–6, and information about members in the data structure is described in Table 6–5.

**Figure 6–6 Rectangle Data Structure (MIT C Binding)**

```
typedef struct {
    short x,y;
    unsigned short width, height;
} XRectangle;
```

**Table 6–5 Members of the Rectangle Data Structure (MIT C Binding)**

Member Name	Contents
x	Defines the x value of the rectangle origin
y	Defines the y value of the rectangle origin
width	Defines the width of the rectangle
height	Defines the height of the rectangle

## 6.4 Arc Data Structure

The arc data structure defines the starting point and size of an arc. The data structure for the VAX binding is shown in Figure 6–7, and information about members in the data structure is described in Table 6–6.

**Figure 6–7 Arc Data Structure (VAX Binding)**

x\$w_garc_y	x\$w_garc_x	0
x\$w_garc_height	x\$w_garc_width	4
x\$w_garc_angle2	x\$w_garc_angle1	8

**Table 6–6 Members of the Arc Data Structure (VAX Binding)**

Member Name	Contents
X\$W_GARC_X	Defines the x-coordinate value of the rectangle in which the server draws the arc
X\$W_GARC_Y	Defines the y-coordinate value of the rectangle in which the server draws the arc
X\$W_GARC_WIDTH	Defines the x axis diameter of the arc

(continued on next page)

# Graphics Routines

## 6.4 Arc Data Structure

**Table 6–6 (Cont.) Members of the Arc Data Structure (VAX Binding)**

Member Name	Contents
X\$W_GARC_HEIGHT	Defines the y axis diameter of the arc
X\$W_GARC_ANGLE1	Defines the starting point of the arc relative to the three o'clock position from the center of the rectangle
X\$W_GARC_ANGLE2	Defines the extent of the arc relative to the starting point

The data structure for the MIT C binding is shown in Figure 6–8, and information about members in the data structure is described in Table 6–7.

**Figure 6–8 Arc Data Structure (MIT C Binding)**

```
typedef struct {  
    short x,y;  
    unsigned short width, height;  
    short angle1, angle2;  
} XArc;
```

**Table 6–7 Members of the Arc Data Structure (MIT C Binding)**

Member Name	Contents
x	Defines the x-coordinate of the rectangle in which the server draws the arc
y	Defines the y-coordinate of the rectangle in which the server draws the arc
width	Defines the x axis diameter of the arc
height	Defines the y axis diameter of the arc
angle1	Defines the starting point of the arc relative to the three o'clock position from the center of the rectangle
angle2	Defines the extent of the arc relative to the starting point

---

## 6.5 Image Data Structure

Each image is defined and referenced using the image data structure. The data structure for the VAX binding is shown in Figure 6–9, and information about members in the data structure is described in Table 6–8.

## Graphics Routines

### 6.5 Image Data Structure

**Figure 6–9 Image Data Structure (VAX Binding)**

---

x\$I_imag_width	0
x\$I_imag_height	4
x\$I_imag_xoffset	8
x\$I_imag_format	12
x\$a_imag_data	16
x\$I_imag_byte_order	20
x\$I_imag_bitmap_unit	24
x\$I_imag_bitmap_bit_order	28
x\$I_imag_bitmap_pad	32
x\$I_imag_depth	36
x\$I_imag_bytes_per_line	40
x\$I_imag_bits_per_pixel	44
x\$I_imag_red_mask	48
x\$I_imag_green_mask	52
x\$I_imag_blue_mask	56
x\$a_imag_obdata	60
x\$a_imag_create_image	64
x\$a_imag_destroy_image	68
x\$a_imag_get_pixel	72
x\$a_imag_put_pixel	76
x\$a_imag_sub_image	80
x\$a_imag_add_pixel	84

---



# Graphics Routines

## 6.5 Image Data Structure

**Table 6–8 Members of the Image Data Structure (VAX Binding)**

Member Name	Contents								
X\$L_IMAG_WIDTH	Specifies the width of the image								
X\$L_IMAG_HEIGHT	Specifies the height of the image								
X\$L_IMAG_OFFSET	Specifies the number of pixels offset in the x direction. Specifying an offset permits the server to ignore the beginning of scan lines and rapidly display images when ZPixmap format is used.								
X\$L_IMAG_FORMAT	Specifies whether the data is stored in XYPixmap or ZPixmap format. The following flags facilitate specifying data format:								
	<table border="1"> <thead> <tr> <th>Flag Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>x\$c_xy_bitmap</td> <td>A single bitmap representing one plane</td> </tr> <tr> <td>x\$c_xy_pixmap</td> <td>A set of bitmaps representing individual planes</td> </tr> <tr> <td>x\$c_z_pixmap</td> <td>Data organized as a list of pixel values viewed as a horizontal row</td> </tr> </tbody> </table>	Flag Name	Description	x\$c_xy_bitmap	A single bitmap representing one plane	x\$c_xy_pixmap	A set of bitmaps representing individual planes	x\$c_z_pixmap	Data organized as a list of pixel values viewed as a horizontal row
Flag Name	Description								
x\$c_xy_bitmap	A single bitmap representing one plane								
x\$c_xy_pixmap	A set of bitmaps representing individual planes								
x\$c_z_pixmap	Data organized as a list of pixel values viewed as a horizontal row								
X\$A_IMAG_DATA	The address of the image data								
X\$L_IMAG_BYTE_ORDER	Indicates whether the least significant or the most significant byte is first								
X\$L_IMAG_BITMAP_UNIT	Specifies whether the bitmap is organized in units of 8, 16, or 32 bits								
X\$L_IMAG_BITMAP_BIT_ORDER	Specifies whether the bitmap order is least or most significant								
X\$L_IMAG_BITMAP_PAD	Specifies whether padding in XY format or Z format should be done in units of 8, 16, or 32 bits								
X\$L_IMAG_DEPTH	The depth of the image								
X\$L_IMAG_BYTES_PER_LINE	The bytes per line to be used as an accelerator								
X\$L_IMAG_BITS_PER_PIXEL	Indicates for Z format the number of bits per pixel								
X\$L_IMAG_RED_MASK	Specifies the red values for ZFormat								
X\$L_IMAG_GREEN_MASK	Specifies the green values for ZFormat								
X\$L_IMAG_BLUE_MASK	Specifies the blue values for ZFormat								
X\$A_IMAG_OBDATA	A structure that contains object routines								
X\$A_IMAG_CREATE_IMAGE	A CREATE IMAGE routine								
X\$A_IMAG_DESTROY_IMAGE	A DESTROY IMAGE routine								
X\$A_IMAG_GET_PIXEL	A GET PIXEL routine								
X\$A_IMAG_PUT_PIXEL	A PUT IMAGE routine								
X\$A_IMAG_SUB_IMAGE	A SUB IMAGE routine								
X\$A_IMAG_ADD_PIXEL	An ADD PIXEL routine								

The data structure for the MIT C binding is shown in Figure 6–10, and information about members in the data structure is described in Table 6–9.

# Graphics Routines

## 6.5 Image Data Structure

**Figure 6–10 Image Data Structure (MIT C Binding)**

---

```

typedef struct _XImage{
    int width,height;
    int xoffset;
    int format;
    char *data;
    int byte_order;
    int bitmap_unit;
    int bitmap_bit_order;
    int bitmap_pad;
    int depth;
    int bytes_per_line;
    int bits_per_pixel;
    unsigned long red_mask;
    unsigned long green_mask;
    unsigned long blue_mask;
    char *obdata;
    struct funcs {
        struct _XImage *(*create_image) ();
        int (*destroy_image) ();
        unsigned long (*get_pixel) ();
        int (*put_pixel) ();
        struct _XImage* (*sub_image) ();
        int (*add_pixel) ();
    }f;
}XImage;

```

---

**Table 6–9 Members of the Image Data Structure (MIT C Binding)**

Member Name	Contents								
width	Specifies the width of the image in pixels.								
height	Specifies the height of the image in pixels.								
offset	Specifies the number of bits offset in the x direction. Specifying an offset permits the server to ignore the beginning of scan lines and rapidly display images when ZPixmap format is used.								
format	Specifies whether the data is stored in XYPixmap or ZPixmap format. The following flags facilitate specifying data format: <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr> <th style="text-align: left;">Constant Name</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>XYBitmap</td> <td>A single bitmap representing one plane</td> </tr> <tr> <td>XYPixmap</td> <td>A set of bitmaps representing individual planes</td> </tr> <tr> <td>ZPixmap</td> <td>Data organized as a list of pixel values viewed as a horizontal row</td> </tr> </tbody> </table>	Constant Name	Description	XYBitmap	A single bitmap representing one plane	XYPixmap	A set of bitmaps representing individual planes	ZPixmap	Data organized as a list of pixel values viewed as a horizontal row
Constant Name	Description								
XYBitmap	A single bitmap representing one plane								
XYPixmap	A set of bitmaps representing individual planes								
ZPixmap	Data organized as a list of pixel values viewed as a horizontal row								
data	The address of the image data.								

---

(continued on next page)

# Graphics Routines

## 6.5 Image Data Structure

**Table 6–9 (Cont.) Members of the Image Data Structure (MIT C Binding)**

Member Name	Contents						
byte_order	Indicates whether the least significant or the most significant byte is first. The following flags facilitate specifying byte order:  <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>LSBFirst</td> <td>Least significant byte first</td> </tr> <tr> <td>MSBFirst</td> <td>Most significant byte first</td> </tr> </tbody> </table>	Constant Name	Description	LSBFirst	Least significant byte first	MSBFirst	Most significant byte first
Constant Name	Description						
LSBFirst	Least significant byte first						
MSBFirst	Most significant byte first						
bitmap_unit	Specifies whether the bitmap is organized in units of 8, 16, or 32 bits (not used with ZPixmap).						
bitmap_bit_order	Specifies whether the least significant or most significant bit is first. The following flags facilitate specifying bitmap order:  <table border="1"> <thead> <tr> <th>Constant Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>LSBFirst</td> <td>Least significant byte first</td> </tr> <tr> <td>MSBFirst</td> <td>Most significant byte first</td> </tr> </tbody> </table>	Constant Name	Description	LSBFirst	Least significant byte first	MSBFirst	Most significant byte first
Constant Name	Description						
LSBFirst	Least significant byte first						
MSBFirst	Most significant byte first						
bitmap_pad	Specifies whether scan line padding in XY format or Z format should be done in units of 8, 16, or 32 bits.						
depth	Specifies the depth of the image.						
bytes_per_line	Specifies the bytes per line to be used as an accelerator.						
bits_per_pixel	Indicates for ZFormat the number of bits per pixel.						
red_mask	Specifies red values for ZFormat.						
green_mask	Specifies green values for ZFormat.						
blue_mask	Specifies blue values for ZFormat.						
obdata	The address of a structure that contains object routines.						
create_image	The address of a CREATE IMAGE routine.						
destroy_image	The address of a DESTROY IMAGE routine.						
get_pixel	The address of a GET PIXEL routine.						
put_pixel	The address of a PUT PIXEL routine.						
sub_image	The address of a SUB IMAGE routine.						
add_pixel	The address of an ADD PIXEL routine.						

## 6.6 Graphics Routines

The following pages describe the Xlib graphics routines.

---

## ADD PIXEL

Increments each pixel in an image by a constant value.

---

**VAX FORMAT**    **X\$ADD\_PIXEL**    (*ximage, value*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
ximage	record	x\$image	read	reference
value	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XAddPixel**    (*ximage, value*)

**argument  
information**

```
int XAddPixel(ximage, value)
    XImage *ximage;
    int value;
```

---

### ARGUMENTS

***ximage***

The image data structure which contains the image to be changed. For more information on the image data structure, see Section 6.5.

***value***

The constant value to add to each pixel.

---

### DESCRIPTION

ADD PIXEL adds a constant value to each pixel in an image.

## Graphics Routines

### CLEAR AREA

---

## CLEAR AREA

Clears a rectangular area in a window.

---

### VAX FORMAT

#### **X\$CLEAR\_AREA**

*(display, window\_id, x\_coord, y\_coord, width, height, exposures)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
exposures	Boolean	uns longword	read	reference

---

---

### MIT C FORMAT

#### **XCclearArea**

*(display, window\_id, x\_coord, y\_coord, width, height, exposures)*

#### argument information

```
XCclearArea(display, window_id, x_coord, y_coord, width, height,
             exposures)
Display *display;
Window window_id;
int x_coord, y_coord;
unsigned int width, height;
Bool exposures;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***window\_id***

The identifier of the window where an area will be cleared. If the window specified is an input-only window, an error will occur.

#### ***x\_coord***

The x-coordinate of the rectangle to be cleared. This coordinate is relative to the origin of the window. The x- and y-coordinates define the upper left corner of the rectangle.

***y\_coord***

The y-coordinate of the rectangle to be cleared. This coordinate is relative to the origin of the window. The x- and y-coordinates define the upper left corner of the rectangle.

***width***

The width, in pixels, of the rectangle to be cleared. The width and height determine the area of the rectangle to be cleared.

If the value of **width** is zero, it is assigned a default value equivalent to the current width of the window minus the value of the x-coordinate.

***height***

The height, in pixels, of the rectangle to be cleared. The width and height determine the area of the rectangle to be cleared.

If the value of **height** is zero, it is assigned a default value equivalent to the current height of the window minus the value of the y-coordinate.

***exposures***

The flag that specifies whether or not one or more exposure events are generated. When true, one or more exposure events are generated for regions of the rectangle that are either visible or are being retained in a backing store. When false, no exposure events are generated.

---

**DESCRIPTION**

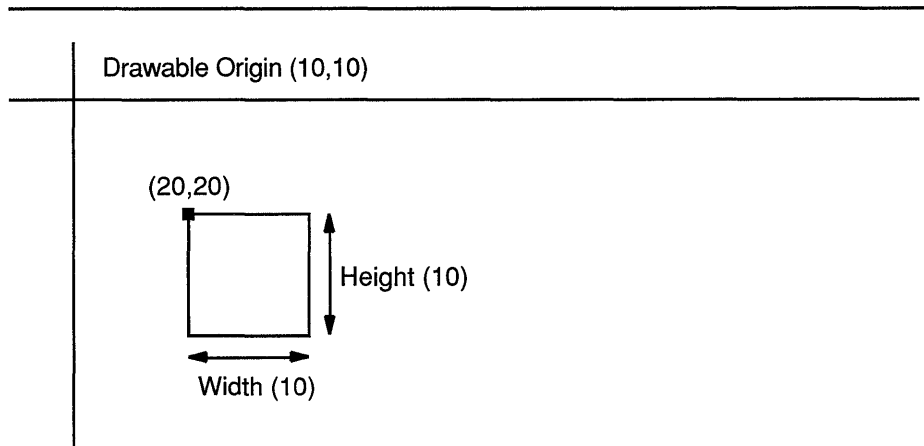
CLEAR AREA clears a rectangular area in a specified window.

The rectangle is defined by the arguments **x\_coord**, **y\_coord**, **width**, and **height** and by the window origin coordinates. The x- and y-coordinates are relative to the window origin coordinates and define the upper left corner of the rectangle. The width and height specify the area. For example, if you specify the width and height to be 10 and 10, the rectangle to be cleared would be as shown in Figure 6-11.

# Graphics Routines

## CLEAR AREA

**Figure 6–11 Rectangular Area Cleared**



ZK-0015A-GE

The area cleared is replaced by a background. If the background has a specific tile pattern, the rectangular area is tiled with a plane mask of all ones and a function of GXcopy. If the background is none, the contents of the cleared rectangular area do not change.

To clear the entire window, use CLEAR WINDOW.

## X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	<p>Possible causes are as follows:</p> <ul style="list-style-type: none"> <li>In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>An input-only window is used as a drawable.</li> <li>One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>An input-only window lacks this attribute.</li> </ul>
X\$C_BAD_VALUE	BadValue	<p>Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.</p>
X\$C_BAD_WINDOW	BadWindow	<p>A value that you specified for a window argument does not name a defined window.</p>

---

## CLEAR WINDOW

Clears an entire window.

---

**VAX FORMAT**    **X\$CLEAR\_WINDOW**    (*display, window\_id*)

---

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
window_id	identifier	uns longword	read	reference

---



---

**MIT C FORMAT**    **XClearWindow**    (*display, window\_id*)

---

**argument  
information**

```
XClearWindow(display, window_id)
Display *display;
Window window_id;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***window\_id***

The identifier of the window to be cleared.

---

**DESCRIPTION**

CLEAR WINDOW clears the entire window area. The identifier of the window was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW.

The area cleared is replaced by the window's background. If the background has a defined tile, the window is tiled with a plane mask of all ones and a function of GXcopy. If the background is none, the contents of the cleared window does not change.

To clear an area within a window, use CLEAR AREA.



# Graphics Routines

## CLEAR WINDOW

---

### X ERRORS

VAX	C	Description
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

## COPY AREA

Copies a specified rectangular area from one window or pixmap (drawable) to another drawable.

**VAX FORMAT**     **X\$COPY\_AREA**  
*(display, src\_drawable\_id, dst\_drawable\_id, gc\_id, src\_x\_coord, src\_y\_coord, width, height, dst\_x\_coord, dst\_y\_coord)*

**argument information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
src_drawable_id	identifier	uns longword	read	reference
dst_drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
src_x_coord	longword	longword	read	reference
src_y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
dst_x_coord	longword	longword	read	reference
dst_y_coord	longword	longword	read	reference

**MIT C FORMAT**     **XCopyArea**  
*(display, src\_drawable\_id, dst\_drawable\_id, gc\_id, src\_x\_coord, src\_y\_coord, width, height, dst\_x\_coord, dst\_y\_coord)*

**argument information**

```
XCopyArea(display, src_drawable_id, dst_drawable_id, gc_id,
          src_x_coord, src_y_coord, width, height, dst_x_coord,
          dst_y_coord)
Display *display;
Drawable src_drawable_id, dst_drawable_id;
GC gc_id;
int src_x_coord, src_y_coord;
unsigned int width, height;
int dst_x_coord, dst_y_coord;
```

## Graphics Routines

### COPY AREA

---

#### ARGUMENTS

***display***

The display information originally returned by OPEN DISPLAY.

***src\_drawable\_id***

The identifier of the source window or pixmap (drawable) containing the rectangular area to be copied. The drawable identifier can be either a window or pixmap identifier. The source drawable must be of the same root and depth as the destination drawable.

***dst\_drawable\_id***

The identifier of the destination window or pixmap (drawable) where the rectangular area will be copied to. The drawable identifier can be either a window or pixmap identifier. The destination drawable must be of the same root and depth as the source drawable.

***gc\_id***

The identifier of the graphics context associated with the drawables.

***src\_x\_coord***

The x-coordinate of the rectangle to be copied from the source drawable. The x-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***src\_y\_coord***

The y-coordinate of the rectangle to be copied from the source drawable. The y-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***width***

The width, in pixels, of the rectangle. The width and height define the area of the rectangle to be copied. The rectangle copied to the destination drawable also has the same dimensions.

***height***

The height, in pixels, of the rectangle. The width and height define the area of the rectangle to be copied. The rectangle copied to the destination drawable also has the same dimensions.

***dst\_x\_coord***

The x-coordinate of the destination drawable where the rectangle will be copied to. The x-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***dst\_y\_coord***

The y-coordinate of the destination drawable where the rectangle will be copied to. The y-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

---

#### DESCRIPTION

COPY AREA copies a rectangular area from one window or pixmap (drawable) to another drawable. The drawables are specified by their identifiers. If the drawables are pixmaps, the identifiers were originally returned by CREATE\_PIXMAP. If the drawables are windows, the identifiers were originally returned by any CREATE\_WINDOW request.

The rectangular area to be copied is specified as follows:

- The starting point is specified by x- and y-coordinates in **src\_x\_coord** and **src\_y\_coord**. These coordinates define the upper left corner of the rectangle, in relation to the origin of the drawable.
- The size of the rectangle is specified by **width** and **height**.

The rectangular area copied is specified as follows:

- The place where the copying starts is specified by x- and y-coordinates in **dst\_x\_coord** and **dst\_y\_coord**. These coordinates define the upper left corner of the rectangle, in relation to the origin of the drawable.
- The size of the rectangle is defined by **width** and **height**.

COPY AREA uses the graphics context to define how copying is done. The following members are used:

- Function
- Plane Mask
- Subwindow Mode
- Graphics Exposures
- Clip X Origin
- Clip Y Origin
- Clip Mask

COPY AREA combines the specified rectangle from the source with the specified rectangle of the destination. If the regions of the source rectangle are obscured and have not been retained by the server, or if regions outside the boundaries of the source drawable are specified, the corresponding regions of the destination are tiled with the plane mask member of all ones and function GXcopy with that background, unless the destination drawable is a window with a background of none.

When the graphics exposure member in the graphics context is true, graphics exposure events for the corresponding destination regions are generated. If the graphics exposure member is true, but no regions are exposed, then a no expose event is generated.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

## Graphics Routines

### COPY AREA

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_MATCH	BadMatch	<p>Possible causes are as follows:</p> <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## COPY PLANE

Copies a rectangular area from one plane of a window or pixmap (drawable) to another drawable.

---

### VAX FORMAT

### X\$COPY\_PLANE

*(display, src\_drawable\_id, dst\_drawable\_id, gc\_id, src\_x\_coord, src\_y\_coord, width, height, dst\_x\_coord, dst\_y\_coord, plane)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
src_drawable_id	identifier	uns longword	read	reference
dst_drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
src_x_coord	longword	longword	read	reference
src_y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
dst_x_coord	longword	longword	read	reference
dst_y_coord	longword	longword	read	reference
plane	mask_longword	uns longword	read	reference

---



---

### MIT C FORMAT

### XCopyPlane

*(display, src\_drawable\_id, dst\_drawable\_id, gc\_id, src\_x\_coord, src\_y\_coord, width, height, dst\_x\_coord, dst\_y\_coord, plane)*

#### argument information

```
XCopyPlane(display, src_drawable_id, dst_drawable_id, gc_id,
            src_x_coord, src_y_coord, width, height, dst_x_coord,
            dst_y_coord, plane)
Display *display;
Drawable src_drawable_id, dst_drawable_id;
GC gc_id;
int src_x_coord, src_y_coord;
unsigned int width, height;
int dst_x_coord, dst_y_coord;
unsigned long plane;
```

## Graphics Routines

### COPY PLANE

---

#### ARGUMENTS

##### ***display***

The display information originally returned by OPEN DISPLAY.

##### ***src\_drawable\_id***

The identifier of the window or pixmap (drawable) containing the bit plane to be copied. The drawable identifier can be either a window or pixmap identifier. The source drawable must be of the same root, but not necessarily of the same depth, as the destination drawable.

##### ***dst\_drawable\_id***

The identifier of the window or pixmap (drawable) where the bit plane will be copied to. The drawable identifier can be either a window or pixmap identifier. The destination drawable must be of the same root, but not necessarily of the same depth, as the source drawable.

##### ***gc\_id***

The identifier of the graphics context associated with the drawables.

##### ***src\_x\_coord***

The x-coordinate of the rectangle to be copied from the plane of the source drawable. The x-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

##### ***src\_y\_coord***

The y-coordinate of the rectangle to be copied from the plane of the source drawable. The y-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

##### ***width***

The width, in pixels, of the rectangle. The width and height define the area of the rectangle to be copied. The rectangle copied to the destination drawable also has the same dimensions.

##### ***height***

The height, in pixels, of the rectangle. The width and height define the area of the rectangle to be copied. The rectangle copied to the destination drawable also has the same dimensions.

##### ***dst\_x\_coord***

The x-coordinate within a plane of the destination drawable where the rectangle will be copied to. The x-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

##### ***dst\_y\_coord***

The y-coordinate within a plane of the destination drawable where the rectangle will be copied to. The y-coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

##### ***plane***

A bit mask that specifies which plane to copy from. Exactly one bit is set to 1. The least significant bit represents the first plane; increasingly significant bits represent subsequent planes (in order).

---

## DESCRIPTION

`COPY PLANE` copies a rectangular area from a plane of a window or pixmap (drawable) to another drawable. The drawables are specified by their identifiers. If the drawables are pixmaps, the identifiers were originally returned by `CREATE_PIXMAP`. If the drawables are windows, the identifiers were originally returned by `CREATE_SIMPLE_WINDOW` or `CREATE_WINDOW`.

The rectangular area to be copied is specified as follows:

- The starting point is specified by x- and y-coordinates in **`src_x_coord`** and **`src_y_coord`**. These coordinates define the upper left corner of the rectangle, in relation to the origin of the drawable.
- The size of the rectangle is defined by **`width`** and **`height`**.

The rectangular area copied is specified as follows:

- The place where the copying starts is specified by x- and y-coordinates in **`dst_x_coord`** and **`dst_y_coord`**. These coordinates define the upper left corner of the rectangle, in relation to the origin of the drawable.
- The size of the rectangle is defined by **`width`** and **`height`**.

`COPY PLANE` uses the graphics context to define how copying is done. The following members are used:

- Function
- Plane Mask
- Foreground
- Background
- Subwindow Mode
- Graphics Exposures
- Clip X Origin
- Clip Y Origin
- Clip Mask

For more information about the graphics context data structure and each of its members, see Section 5.1.

`COPY PLANE` combines the rectangular area of the plane from the source drawable with the foreground/background pixels in the graphics context to form a pixmap of the same depth as the destination drawable. The rectangular area of the plane is treated similarly to an opaque stippled fill. All ones are replaced by the foreground color, and all zeros are replaced by the background color. The equivalent of a call to `COPY AREA` is performed with all the same exposure semantics.



# Graphics Routines

## COPY PLANE

---

### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

---

## CREATE IMAGE

Specifies the size of the image and allocates sufficient memory for the image data structure.

---

**VAX FORMAT**     *status\_return = X\$CREATE\_IMAGE*  
                           (*display, visual\_id, depth, image\_format, offset,*  
                           *data, width, height, bitmap\_pad, bytes\_per\_line,*  
                           *ximage\_return*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
visual_id	identifier	uns longword	read	reference
depth	longword	longword	read	reference
image_format	longword	longword	read	reference
offset	longword	longword	read	reference
data	char string	char string	read	descriptor
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
bitmap_pad	longword	longword	read	reference
bytes_per_line	longword	longword	read	reference
ximage_return	record	x\$image	write	reference

---

**MIT C FORMAT**     *ximage\_return = XCreateImage*  
                           (*display, visual\_id, depth, image\_format, offset, data,*  
                           *width, height, bitmap\_pad, bytes\_per\_line*)

# Graphics Routines

## CREATE IMAGE

### argument information

---

```
XImage *XCreateImage(display, visual_id, depth, image_format,
                    offset, data, width, height, bitmap_pad,
                    bytes_per_line)
Display *display;
Visual *visual_id;
unsigned int depth;
int image_format;
int offset;
char *data;
unsigned int width;
unsigned int height;
int bitmap_pad;
int bytes_per_line;
```

---

### RETURNS

#### ***status\_return (VAX only)***

Specifies whether or not the return completed successfully.

#### ***ximage\_return (MIT C only)***

A pointer to the newly created image.

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***visual\_id***

A pointer to a visual structure. For information about this visual structure, see the description of the DEFAULT VISUAL routine in Chapter 2.

#### ***depth***

The depth of the image that will be created and referenced by the image data structure.

#### ***image\_format***

The format of the image that will be created and referenced by the image data structure. The predefined values for **format** are as follows:

---

VAX	C	
X\$C_BITMAP	Bitmap	Bitmap image
X\$C_XY_PIXMAP	XPixmap	XY pixmap format image
X\$C_Z_PIXMAP	ZPixmap	Z pixmap format image

---

Other values specified in this argument are not valid.

#### ***offset***

The number of pixels beyond the first address of a scan line where an image actually begins. The **offset** argument is useful when the image is not on an addressable boundary.

#### ***data***

A pointer to the data that defines the image.

***width***

The width, in pixels, of the image. The width and height determine the area of the image.

***height***

The height, in pixels, of the image. The width and height determine the area of the image.

***bitmap\_pad***

The space allocated in memory for each scan line. This value specifies the number of bits to multiply by **bytes\_per\_line**. The value of the **bitmap\_pad** argument must be 8, 16, or 32.

***bytes\_per\_line***

The number of bytes in a scan line. When zero, it is assumed that the scan lines are contiguous in memory and the number of bytes per line will then be calculated for you.

***ximage\_return***

A pointer to the newly created image.

---

**DESCRIPTION**

CREATE IMAGE allocates memory for the image data structure. It initializes the image data structure with the values you specify in the arguments. A pointer to the image data structure is returned. Use this pointer in subsequent routines to reference the image data structure.

CREATE IMAGE does not allocate space for the image itself.

The red, green, and blue mask values stored in the image data structure are relevant only for z pixmap format images. If you are working with the z pixmap format, these values are derived from the visual type specified in **visual\_id**.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_ALLOC	BadAlloc	The server did not allocate the requested resource for any cause.
X\$C_BAD_COLOR	BadColor	A value that you specified for a color map argument does not name a defined color map.
X\$C_BAD_CURSOR	BadCursor	A value that you specified for a cursor argument does not name a defined cursor.

## Graphics Routines

### CREATE IMAGE

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_PIXMAP	BadPixmap	A value that you specified for a pixmap argument does not name a defined pixmap.
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
X\$C_BAD_WINDOW	BadWindow	A value that you specified for a window argument does not name a defined window.

---

## DESTROY IMAGE

Deallocates the memory associated with an image structure.

---

**VAX FORMAT**     *status\_return = X\$DESTROY\_IMAGE (ximage)*

argument  
information

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
ximage	record	x\$image	read	reference

---



---

**MIT C FORMAT**     *status\_return = XDestroyImage (ximage)*

argument  
information

```
int *XDestroyImage(ximage)
    XImage *ximage;
```

---

**RETURNS**     *status\_return*  
A return value that specifies whether or not the routine completed successfully.

---

**ARGUMENTS**     *ximage*  
A pointer to the image data structure that you want to destroy.

---

**DESCRIPTION**     DESTROY IMAGE deallocates the memory associated with an image data structure.  
  
When the image specified by **ximage** was created by CREATE IMAGE, GET IMAGE, or SUB IMAGE, the destroy procedure that DESTROY IMAGE calls frees both the image structure and the data pointed to by the image structure.

## Graphics Routines

### DRAW ARC

---

## DRAW ARC

Draws one arc in the specified window or pixmap.

---

### VAX FORMAT

#### **X\$DRAW\_ARC**

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height, angle1, angle2)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
angle1	longword	longword	read	reference
angle2	longword	longword	read	reference

---

---

### MIT C FORMAT

#### **XDrawArc**

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height, angle1, angle2)*

#### argument information

```
XDrawArc(display, drawable_id, gc_id, x_coord, y_coord, width,
         height, angle1, angle2)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
unsigned int width, height;
int angle1, angle2;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) to draw the arc in. The drawable identifier can be either a window or pixmap identifier. If the drawable is a window, the identifier was originally returned by CREATE

SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

***x\_coord***

The x-coordinate of the rectangle used to define the arc. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***y\_coord***

The y-coordinate of the rectangle used to define the arc. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***width***

The width, in pixels, of the rectangle used to define the arc. The width and height are the major and minor axes of the arc. If one of the axes is specified as zero, a horizontal or vertical line is drawn.

***height***

The height, in pixels, of the rectangle used to define the arc. The width and height are the major and minor axes of the arc. If one of the axes is specified as zero, a horizontal or vertical line is drawn.

***angle1***

The angle to specify the beginning of the arc relative to the three o'clock position from the center. This value is in degrees, scaled by 64 with positive indicating counterclockwise motion and negative indicating clockwise motion.

***angle2***

The angle to specify the path and extent of the arc relative to the start of the arc. This value is in degrees, scaled by 64 with positive indicating counterclockwise motion and negative indicating clockwise motion.

---

**DESCRIPTION**

DRAW ARC draws one arc in the specified drawable. To specify the dimensions of an arc, follow this procedure:

- Specify a rectangle that represents the size of the ideal center path of an ellipse from which the arc will be cut. The **x\_coord** and **y\_coord** arguments specify the upper left corner of the rectangle. The **width** and **height** arguments specify the size of the rectangle.
- Specify where the arc should start in relation to the three o'clock position within the ellipse (in **angle1**).
- Specify the extent or angular length of the arc, relative to the start of the arc (in **angle2**).

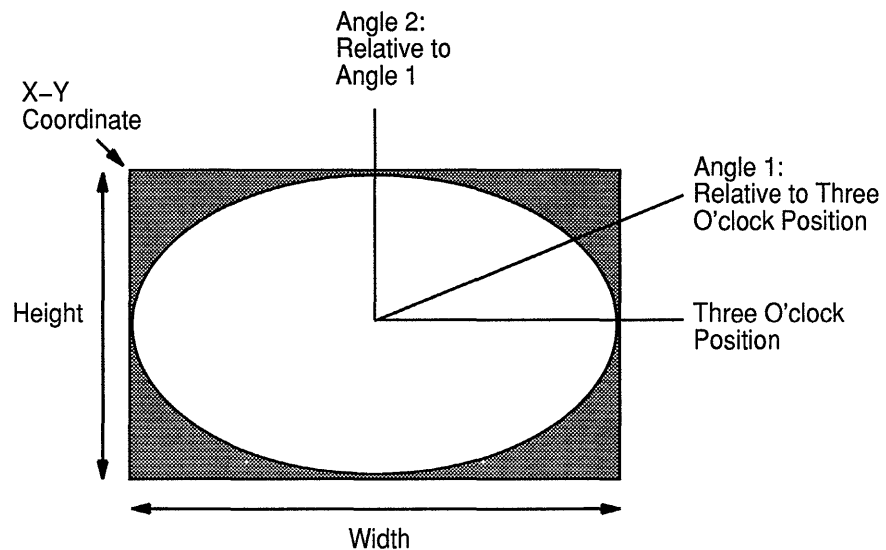


## Graphics Routines

### DRAW ARC

Figure 6-12 illustrates the angles that specify an ellipse.

**Figure 6-12 Specifying an Arc**



ZK-0018A-GE

DRAW ARC uses the following graphics context members:

- Function
- Plane Mask
- Subwindow Mode
- Graphics Exposures
- Clip X Origin
- Clip Y Origin
- Clip Mask

DRAW ARC also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin
- Dash offset
- Dash list

For more information about the graphics context data structure and each of its members, see Section 5.1.

When you want to draw more than one arc, use DRAW ARCS. Using DRAW ARC repeatedly is less efficient than using DRAW ARCS.

For information about filling an arc, see the descriptions of the FILL ARC and FILL ARCS routines.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## Graphics Routines

### DRAW ARCS

---

## DRAW ARCS

Draws more than one arc in the specified window or pixmap.

---

**VAX FORMAT**    **X\$DRAW\_ARCS**  
(*display, drawable\_id, gc\_id, arcs, num\_arcs*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
arcs	array	x\$arc	read	reference
num_arcs	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XDrawArcs**  
(*display, drawable\_id, gc\_id, arcs, num\_arcs*)

**argument  
information**

```
XDrawArcs(display, drawable_id, gc_id, arcs, num_arcs)
Display *display;
Drawable drawable_id;
GC gc_id;
XArc *arcs;
int num_arcs;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) to draw the arcs in. The drawable identifier can be either a window or pixmap identifier.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

#### ***arcs***

A pointer to an array of arc data structures, where each element defines one arc to be drawn. The length of the array is specified by **num\_arcs**.

### *num\_arcs*

The number of arcs to be drawn. This value specifies the length of the array **arcs**.

---

## DESCRIPTION

DRAW ARCS draws more than one arc in the specified drawable. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

The arcs are drawn in the order they are listed in the array **arcs**. For any given arc drawn with nonzero line widths, no pixel is drawn more than once. If arcs intersect, pixels are drawn multiple times. If the last point in one arc coincides with the first point in the following arc, the two arcs will join according to the join style. Angles are computed based on a coordinate system (before skewing either axis to form an ellipse).

Specify the position and size of one arc in one arc data structure.

For information about the arc data structure, see Section 6.4.

DRAW ARCS uses the following members of the graphics context:

- Function
- Plane Mask
- Join Style
- Subwindow Mode
- Cap Style
- Graphics Exposures
- Clip X Origin
- Clip Y Origin
- Clip Mask

DRAW ARCS also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin
- Dash offset
- Dash list

For more information about the graphics context data structure and its members, see Section 5.1.

## Graphics Routines

### DRAW ARCS

For more information about specifying the dimensions of an arc, or for information on drawing just one arc, refer to the DRAW ARC routine.

For information about filling an arc, see the descriptions of the FILL ARC and FILL ARCS routines.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## DRAW LINE

Draws one line between two points in the specified window or pixmap.

---

### VAX FORMAT

#### X\$DRAW\_LINE

*(display, drawable\_id, gc\_id, x1\_coord, y1\_coord, x2\_coord, y2\_coord)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x1_coord	longword	longword	read	reference
y1_coord	longword	longword	read	reference
x2_coord	longword	longword	read	reference
y2_coord	longword	longword	read	reference

---



---

### MIT C FORMAT

#### XDrawLine

*(display, drawable\_id, gc\_id, x1\_coord, y1\_coord, x2\_coord, y2\_coord)*

#### argument information

```
XDrawLine(display, drawable_id, gc_id, x1_coord, y1_coord,
          x2_coord, y2_coord)
Display *display;
Drawable drawable_id;
GC gc_id;
int x1_coord, y1_coord, x2_coord, y2_coord;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) to draw the line in. The drawable identifier can be either a window or pixmap identifier.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. The identifier of the graphics context was originally returned by DEFAULT GC or CREATE GC.

## Graphics Routines

### DRAW LINE

#### ***x1\_coord***

The x-coordinate of the first point. This coordinate is relative to the origin of the drawable. The x1- and y1-coordinates define the first point of the line.

#### ***y1\_coord***

The y-coordinate of the first point. This coordinate is relative to the origin of the drawable. The x1- and y1-coordinates define the first point of the line.

#### ***x2\_coord***

The x-coordinate of the second point. This coordinate is relative to the origin of the drawable. The x2- and y2-coordinates define the second point of the line.

#### ***y2\_coord***

The y-coordinate of the second point. This coordinate is relative to the origin of the drawable. The x2- and y2-coordinates define the second point of the line.

---

### DESCRIPTION

DRAW LINE draws one line between the two specified points on a drawable. If the drawable is a window, the identifier was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

DRAW LINE uses the following graphics context members to draw the line in the drawable:

- Function
- Plane Mask
- Subwindow Mode
- Graphics Exposures
- Clip X Origin
- Clip Y Origin
- Clip Mask

DRAW LINE also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin
- Dash offset
- Dash list

## Graphics Routines

### DRAW LINE

For more information about the graphics context data structure and its members, see Section 5.1.

When you want to draw more than one line, use DRAW LINES to draw connected lines or DRAW SEGMENTS to draw unconnected lines. Using DRAW LINE repeatedly is less efficient than using DRAW LINES or DRAW SEGMENTS.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---



## Graphics Routines

### DRAW LINES

---

## DRAW LINES

Draws more than one connected line in the specified drawable.

---

### VAX FORMAT

#### **X\$DRAW\_LINES**

*(display, drawable\_id, gc\_id, points, num\_points, line\_mode)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
points	array	x\$point	read	reference
num_points	longword	longword	read	reference
line_mode	longword	longword	read	reference

---

---

### MIT C FORMAT

#### **XDrawLines**

*(display, drawable\_id, gc\_id, points, num\_points, line\_mode)*

#### argument information

```
XDrawLines(display, drawable_id, gc_id, points, num_points,
           line_mode)
Display *display;
Drawable drawable_id;
GC gc_id;
XPoint *points;
int num_points;
int line_mode;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) to draw the lines in. The drawable identifier can be either a window or pixmap identifier.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

***points***

A pointer to an array of point data structures where each element defines a single point. The length of the array is specified by **num\_points**.

***num\_points***

The number of points to be drawn. This value defines the length of the array **points**.

***line\_mode***

The coordinate mode of the points. The coordinates of the points can be relative to the drawable origin or to the previous point. The predefined values for **mode** are as follows:

VAX	C	Description
X\$C_COORD_MODE	CoordModeOrigin	The point coordinates are relative to the drawable origin.
X\$C_COORD_MODE_PREVIOUS	CoordModePrevious	The first point is relative to the drawable origin and each subsequent point is relative to the point preceding it.

Other values specified in this argument are not valid.

---

**DESCRIPTION**

DRAW LINES draws more than one line in the specified drawable. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

DRAW LINES draws lines from one point in the array **points** to the next point in the array. The lines are drawn in the same order as the points are listed in the array. For any given line with a nonzero width, no pixel is drawn more than once. If zero width (this) lines intersect, pixels are drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once. If the first and last points coincide, the first and last lines will join according to the join style specified in the graphics context.

DRAW LINES uses the following members of the graphics context:

- Function
- Plane Mask
- Subwindow Mode
- Graphics Exposures
- Clip X Origin
- Clip Y Origin
- Clip Mask

DRAW LINES also uses the following mode-dependent members:

- Foreground
- Background

## Graphics Routines

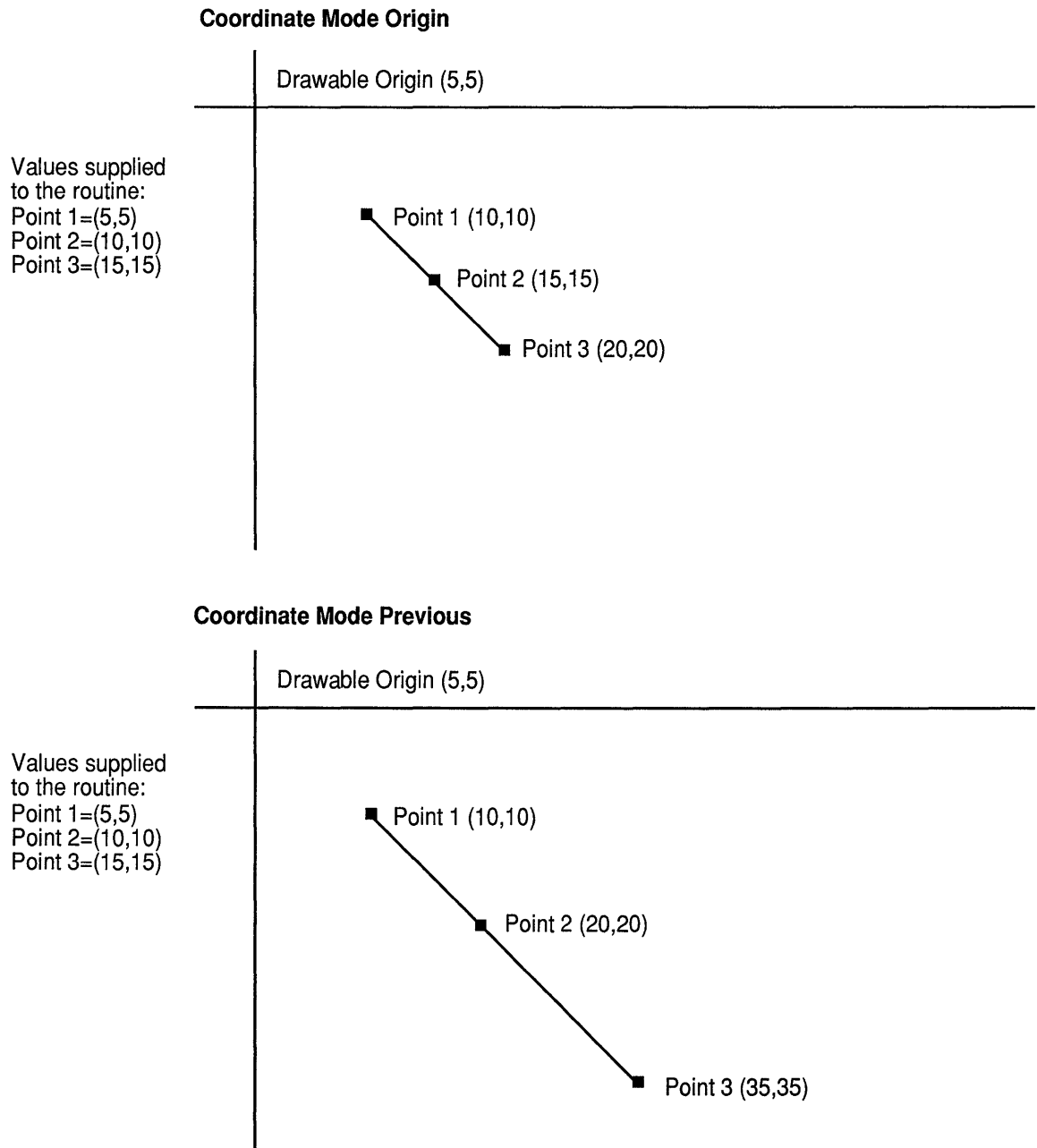
### DRAW LINES

- Tile
- Stipple
- Ts x origin
- Ts y origin
- Dash offset
- Dash list

For more information about the graphics context data structure and its members, see Section 5.1.

The points can be drawn relative to the origin or relative to the previous point. Refer to Figure 6–13 for an illustration of how the mode changes the position of the points.

**Figure 6-13 Lines Drawn in Different Line Modes**



ZK-0020A-GE

When you want to draw a series of unconnected lines, use DRAW SEGMENTS. When you want to draw just one line, use DRAW LINE.

# Graphics Routines

## DRAW LINES

---

### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

---

## DRAW POINT

Draws a point in the specified window or pixmap.

---

### VAX FORMAT

### X\$DRAW\_POINT

*(display, drawable\_id, gc\_id, x\_coord, y\_coord)*

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference

---

### MIT C FORMAT

### XDrawPoint

*(display, drawable\_id, gc\_id, x\_coord, y\_coord)*

#### argument information

```
XDrawPoint(display, drawable_id, gc_id, x_coord, y_coord)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) where the point will be drawn. The drawable identifier can be either a window or pixmap identifier.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by CREATE GC.

#### ***x\_coord***

The x-coordinate of the drawable where the point will be drawn.

#### ***y\_coord***

The y-coordinate of the drawable where the point will be drawn.

# Graphics Routines

## DRAW POINT

---

### DESCRIPTION

DRAW POINT draws one point in the specified drawable. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP. The identifier of the graphics context was originally returned by CREATE GC.

DRAW POINT uses the following graphics context members to draw the point in the drawable:

- Function
- Plane Mask
- Foreground
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

For more information about the graphics context data structure and its members, see Section 5.1.

DRAW POINT is not affected by the tile or stipple pattern in the graphics context.

When you want to draw more than one point, use DRAW POINTS. Using DRAW POINT repeatedly is less efficient than using DRAW POINTS.

---

### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## DRAW POINTS

Draws more than one point in the specified drawable.

---

### VAX FORMAT

#### X\$DRAW\_POINTS

*(display, drawable\_id, gc\_id, points, num\_points, point\_mode)*

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
points	array	x\$point	read	reference
num_points	longword	longword	read	reference
point_mode	longword	longword	read	reference

---

### MIT C FORMAT

#### XDrawPoints

*(display, drawable\_id, gc\_id, points, num\_points, point\_mode)*

#### argument information

```
XDrawPoints(display, drawable_id, gc_id, points, num_points,
            point_mode)
Display *display;
Drawable drawable_id;
GC gc_id;
XPoint *points;
int num_points;
int point_mode;
```

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.

#### *drawable\_id*

The identifier of the window or pixmap (drawable) where the points will be drawn. The drawable identifier can be either a window or pixmap identifier. If the drawable is a window, the identifier was originally returned by any CREATE WINDOW request. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.



## Graphics Routines

### DRAW POINTS

#### *gc\_id*

The identifier of the graphics context associated with the drawable. This identifier was originally returned by `DEFAULT GC` or `CREATE GC`.

#### *points*

A pointer to an array of point data structures where each element defines a single point. The length of the array is specified by `num_points`.

#### *num\_points*

The number of points to be drawn. This value defines the length of the array `points`.

#### *point\_mode*

The coordinate mode of the points. The coordinates of the points can be relative to the drawable origin or relative to the previous point. The predefined values for `point_mode` are as follows:

VAX	C	Description
<code>X\$C_COORD_MODE_ORIGIN</code>	<code>CoordModeOrigin</code>	The point coordinates are relative to the drawable origin.
<code>X\$C_COORD_MODE_PREVIOUS</code>	<code>CoordModePrevious</code>	The first point is relative to the drawable origin and each subsequent point is relative to the point preceding it.

Other values specified in this argument are not valid.

---

## DESCRIPTION

`DRAW POINTS` draws more than one point on the specified drawable.

You specify the x- and y-coordinates in a point data structure for each point you want to draw. The point data structures are stored in the array `points`.

For more information about the point data structure, see Section 6.1.

`DRAW POINTS` uses the following members of the graphics context:

- Function
- Plane Mask
- Foreground
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

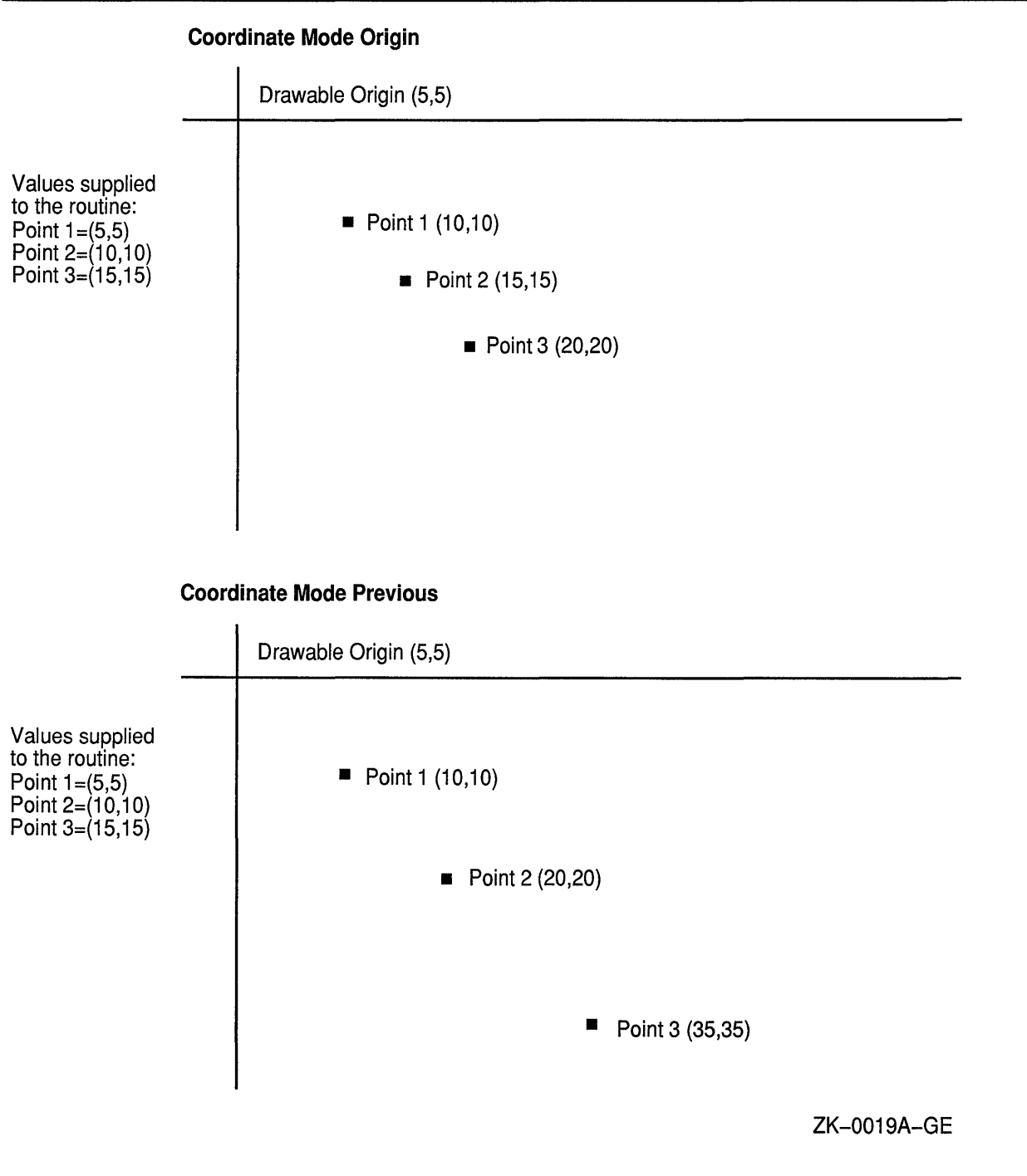
For more information about the graphics context data structure and its members, see Section 5.1.

# Graphics Routines

## DRAW POINTS

The points can be drawn relative to the origin or relative to the previous point. Refer to Figure 6-14 for an illustration of how the mode changes the position of the points.

**Figure 6-14 Points Drawn in Different Coordinate Modes**



ZK-0019A-GE

When you want to draw just one point, use DRAW POINT.

# Graphics Routines

## DRAW POINTS

---

### X ERRORS

VAX	C	Description
X\$_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

---

## DRAW RECTANGLE

Draws the outline of one rectangle in the specified drawable.

---

**VAX FORMAT**    **X\$DRAW\_RECTANGLE**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height)*

**argument information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference

---



---

**MIT C FORMAT**    **XDrawRectangle**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height)*

**argument information**

---

```
XDrawRectangle(display, drawable_id, gc_id, x_coord, y_coord,
               width, height)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
unsigned int width, height;
```

---

**ARGUMENTS**

***display***  
The display information originally returned by OPEN DISPLAY.

***drawable\_id***  
The identifier of the window or pixmap (drawable) to draw the rectangle in. The drawable identifier can be either a window or a pixmap identifier. If the drawable is a window, the identifier was originally returned by any CREATE WINDOW request. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

## Graphics Routines

### DRAW RECTANGLE

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

#### ***x\_coord***

The x-coordinate of the rectangle. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

#### ***y\_coord***

The y-coordinate of the rectangle. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

#### ***width***

The width, in pixels, of the rectangle to be drawn. The width and height define the outline of the rectangle.

#### ***height***

The height, in pixels, of the rectangle to be drawn. The width and height define the outline of the rectangle.

---

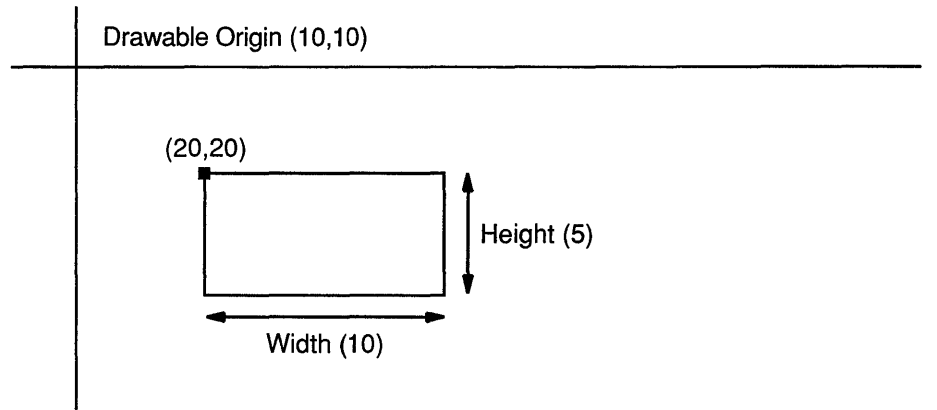
### DESCRIPTION

DRAW RECTANGLE draws a single rectangular outline in the specified drawable.

The x- and y-coordinates (***x\_coord*** and ***y\_coord***) specify the position. The values you specify are relative to the origin of the drawable. The point defined by the coordinates is the upper left corner of the rectangle. The width and height specify the size.

For example, the drawable has an origin at position (10,10). You specify the x-coordinate of the drawable origin as 10 and the y-coordinate of the drawable origin as 10. The upper left corner of the rectangle is at position (20,20). You specify the width and height of the rectangle as 10 and 5, respectively. The resulting rectangular outline is shown in Figure 6-15.

**Figure 6-15 Outline of a Rectangle**



ZK-0014A-GE

**DRAW RECTANGLE** uses the following members of the graphics context:

- Function
- Plane Mask
- Line Width
- Line Style
- Join Style
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

**DRAW RECTANGLE** also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin
- Dash offset
- Dash list

For more information about the graphics context data structure and its members, see Section 5.1.

When you want to draw several rectangles, use **DRAW RECTANGLES**.

## Graphics Routines

### DRAW RECTANGLE

For information about filling a rectangular outline, see the descriptions of the `FILL RECTANGLE` and `FILL RECTANGLES` routines.

---

## X ERRORS

<b>VAX</b>	<b>C</b>	<b>Description</b>
<code>X\$C_BAD_DRAWABLE</code>	<code>BadDrawable</code>	A value that you specified for a drawable argument does not name a defined window or pixmap.
<code>X\$C_BAD_GC</code>	<code>BadGC</code>	A value that you specified for a graphics context argument does not name a defined graphics context.
<code>X\$C_BAD_MATCH</code>	<code>BadMatch</code>	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## DRAW RECTANGLES

Draws the outline of more than one rectangle in the specified drawable.

---

**VAX FORMAT**    **X\$DRAW\_RECTANGLES**  
*(display, drawable\_id, gc\_id, rectangles,  
num\_rectangles)*

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
rectangles	array	x\$rectangle	read	reference
num_rectangles	longword	longword	read	reference

---



---

**MIT C FORMAT**    **XDrawRectangles**  
*(display, drawable\_id, gc\_id, rectangles,  
num\_rectangles)*

**argument  
information**

```
XDrawRectangles(display, drawable_id, gc_id, rectangles,
                num_rectangles)
Display *display;
Drawable drawable_id;
GC gc_id;
XRectangle rectangles[];
int num_rectangles;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***drawable\_id***

The identifier of the window or pixmap (drawable) to draw the rectangles in. The drawable identifier can be either a window or pixmap identifier. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.



## Graphics Routines

### DRAW RECTANGLES

#### ***rectangles***

A pointer to an array of rectangle data structures where each element defines one rectangular outline. The rectangle data structure has members to specify the position and size of the rectangular outline. The length of the array is specified in **num\_rectangles**.

#### ***num\_rectangles***

The number of rectangles to be drawn. This value specifies the length of the array in **rectangles**.

---

### DESCRIPTION

DRAW RECTANGLES draws more than one rectangular outline in the specified drawable. The rectangles are drawn in the order they are listed in the array **rectangles**. For any given rectangle with a nonzero line width, no pixel is drawn more than once. If rectangles intersect, pixels are drawn multiple times.

Specify the position and size of one rectangle in one rectangle data structure. The x- and y-coordinates (in the *x\_coord* and *y\_coord* members) specify the position. The values you specify are relative to the origin of the drawable. The point defined by the coordinates is the upper left corner of the rectangle. The width and height (in the *width* and *height* members) specify the size.

For information about the rectangle data structure, see Section 6.3.

DRAW RECTANGLES uses the following members of the graphics context:

- Function
- Plane Mask
- Line Width
- Line Style
- Join Style
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

DRAW RECTANGLES also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple

- Ts x origin
- Ts y origin
- Dash offset
- Dash list

For more information about the graphics context data structure and its members, see Section 5.1.

When you want to draw just one rectangle, use DRAW RECTANGLE.

For information about filling a rectangular outline, see the descriptions of the FILL RECTANGLE and FILL RECTANGLES routines.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## Graphics Routines

### DRAW SEGMENTS

---

## DRAW SEGMENTS

Draws more than one line in the specified drawable. The lines are not connected unless specified.

---

**VAX FORMAT**    **X\$DRAW\_SEGMENTS**  
(*display, drawable\_id, gc\_id, segments,*  
*num\_segments*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
segments	array	x\$segment	read	reference
num_segments	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XDrawSegments**  
(*display, drawable\_id, gc\_id, segments,*  
*num\_segments*)

**argument  
information**

```
XDrawSegments(display, drawable_id, gc_id, segments,  
              num_segments)  
Display *display;  
Drawable drawable_id;  
GC gc_id;  
XSegment *segments;  
int num_segments;
```

---

## ARGUMENTS

### ***display***

The display information originally returned by OPEN DISPLAY.

### ***drawable\_id***

The identifier of the window or pixmap (drawable) to draw the lines in. The drawable identifier can be either a window or a pixmap identifier. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

***segments***

A pointer to an array of segment data structures where each entry defines two points. The line is drawn between the two points. The coordinates defining the points are relative to the origin of the drawable. The length of the array is specified by **num\_segments**.

***num\_segments***

The number of lines to be drawn. This value defines the length of the array **segments**.

---

**DESCRIPTION**

DRAW SEGMENTS draws more than one line in the specified drawable.

DRAW SEGMENTS uses the points defined in the segment data structures to draw each line. The lines are drawn in the order they are stored in the array. The lines are not connected by a join when you specify a common point. Segments sharing common endpoints will not be joined with the graphics context's join style. You specify the two points for each line within the segment data structure. Refer to Section 6.2 for an illustration of this data structure. Use the X1 and Y1 members to define one point; use the X2 and Y2 members to define the second point.

For information about the segment data structure, see Section 6.2.

For any given line, no pixel is drawn more than once. If lines intersect, pixels are drawn multiple times. The lines are drawn separately, without regard to the join style member specified by the graphics context.

DRAW SEGMENTS uses the following members of the graphics context:

- Function
- Plane Mask
- Line Width
- Line Style
- Cap Style
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

DRAW SEGMENTS also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple

## Graphics Routines

### DRAW SEGMENTS

- Ts x origin
- Ts y origin
- Dash offset
- Dash list

For more information about the graphics context data structure and its members, see Section 5.1.

When you want to draw a series of connected lines, use `DRAW LINES`. When you want to draw just one line, use `DRAW LINE`.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## FILL ARC

Fills in either a pie slice or chord area of an arc in the specified drawable.

---

### VAX FORMAT

#### X\$FILL\_ARC

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height, angle1, angle2)*

---

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
angle1	longword	longword	read	reference
angle2	longword	longword	read	reference

---

### MIT C FORMAT

#### XFillArc

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height, angle1, angle2)*

---

#### argument information

```
XFillArc(display, drawable_id, gc_id, x_coord, y_coord, width,
         height, angle1, angle2)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
unsigned int width, height;
int angle1, angle2;
```

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.

#### *drawable\_id*

The identifier of the window or pixmap (drawable) where the arc is. The drawable identifier can be either a window or pixmap identifier. If the drawable is a window, the identifier was originally returned by CREATE

## Graphics Routines

### FILL ARC

WINDOW or CREATE SIMPLE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

#### ***x\_coord***

The x-coordinate of the rectangle used to define the arc. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

#### ***y\_coord***

The y-coordinate of the rectangle used to define the arc. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

#### ***width***

The width, in pixels, of the rectangle used to define the arc. The width and height are the major and minor axes of the arc.

#### ***height***

The height, in pixels, of the rectangle used to define the arc. The width and height are the major and minor axes of the arc.

#### ***angle1***

The angle to specify the start of the arc relative to the three o'clock position from the center. This value is in degrees, scaled by 64 with positive indicating counterclockwise motion and negative indicating clockwise motion.

#### ***angle2***

The angle to specify the path and extent of the arc relative to the start of the arc. This value is in degrees, scaled by 64 with positive indicating counterclockwise motion and negative indicating clockwise motion.

---

## DESCRIPTION

FILL ARC fills the area defined by one arc in the specified drawable.

The pattern used and the area filled in the arc is specified by the fill style and arc mode members of the associated graphics context. The arc mode specifies whether the pie slice or chord area is filled.

For more information about the graphics context data structure and its members, see Section 5.1.

FILL ARC uses the following graphics context members:

- Function
- Plane Mask
- Fill Style
- Arc Mode
- Subwindow Mode
- Clip X Origin

- Clip Y Origin
- Clip Mask

FILL ARC also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

When you want to fill in more than one arc, use FILL ARCS. Using FILL ARCS is more efficient than using FILL ARC repeatedly.

For information about drawing and specifying the dimensions of an arc, see the description of the DRAW ARC routine.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	<p>Possible causes are as follows:</p> <ul style="list-style-type: none"> <li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>• An input-only window is used as a drawable.</li> <li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>• An input-only window lacks this attribute.</li> </ul>

---



# Graphics Routines

## FILL ARCS

---

### FILL ARCS

Fills in either a pie slice or a chord area of more than one arc in the specified window or pixmap.

---

#### VAX FORMAT

#### **X\$FILL\_ARCS**

*(display, drawable\_id, gc\_id, arcs, num\_arcs)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
arcs	array	x\$arc	read	reference
num_arcs	longword	longword	read	reference

---

#### MIT C FORMAT

#### **XFillArcs**

*(display, drawable\_id, gc\_id, arcs, num\_arcs)*

#### argument information

```
XFillArcs(display, drawable_id, gc_id, arcs, num_arcs)
    Display *display;
    Drawable drawable_id;
    GC gc_id;
    XArc *arcs;
    int num_arcs;
```

---

#### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) where the arcs are. The drawable identifier can be either a window or pixmap identifier. If the drawable is a window, the identifier was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

***arcs***

A pointer to an array of arc data structures, where each element defines one arc to be filled in. The length of the array is specified by **num\_arcs**.

***num\_arcs***

The number of arcs to be filled in. This value specifies the length of the array **arcs**.

---

**DESCRIPTION**

FILL\_ARCS fills areas defined by **arcs** in the specified drawable.

The identifier of the graphics context associated with the drawable was originally returned by DEFAULT\_GC or CREATE\_GC.

The arcs are filled in according to their order in the array. For any one arc, no pixel is drawn more than once. If filled areas intersect, pixels are drawn multiple times.

The pattern used and the area filled in the arc is specified by the fill style and arc mode members of the associated graphics context. The arc mode specifies whether the pie slice or chord area is filled.

For more information about the graphics context data structure and its members, see Section 5.1.

FILL\_ARCS uses the following graphics context members:

- Function
- Plane Mask
- Fill Style
- Arc Mode
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

FILL\_ARCS also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

For information about drawing and specifying the dimensions of an arc, see the description of the DRAW\_ARC routine.

When you want to fill one arc, use FILL\_ARC.

# Graphics Routines

## FILL ARCS

---

### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## FILL POLYGON

Fills a polygon within a specified drawable.

---

### VAX FORMAT

### X\$FILL\_POLYGON

*(display, drawable\_id, gc\_id, points, num\_points, shape, mode)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
points	array	x\$point	read	reference
num_points	longword	longword	read	reference
shape	longword	longword	read	reference
mode	longword	longword	read	reference

---



---

### MIT C FORMAT

### XFillPolygon

*(display, drawable\_id, gc\_id, points, num\_points, shape, mode)*

#### argument information

```
XFillPolygon(display, drawable_id, gc_id, points, num_points,
             shape, mode)
Display *display;
Drawable drawable_id;
GC gc_id;
XPoint *points;
int num_points;
int shape;
int mode;
```

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.

#### *drawable\_id*

The identifier of the window or pixmap (drawable) where the rectangle will be filled. The drawable identifier can be either a window or pixmap identifier. If the drawable is a window, the identifier was originally returned by CREATE WINDOW or CREATE SIMPLE WINDOW. If the

# Graphics Routines

## FILL POLYGON

drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

### *gc\_id*

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

### *points*

A pointer to an array of points that define the polygon shape. Each element in the array is a point data structure. Within the point data structure, the x- and y-coordinates of a point are specified. A path is drawn from one point in the array to the next. This path defines the polygon. The length of the array is specified in **num\_points**.

### *num\_points*

The number of points in the polygon. This defines the length of the array in **points**.

### *shape*

The shape of the polygon. The predefined values for **shape** are as follows:

VAX	C	Description
X\$C_POLYCOMPLEX	Complex	The polygon outline can intersect itself
X\$C_CONVEX	Convex	The polygon is wholly convex
X\$C_NONCONVEX	Nonconvex	The polygon outline does not intersect itself

Other values specified in this argument are not valid.

If the shape of the polygon is known and correctly specified with this argument, server performance can be improved. If the shape is specified incorrectly, the result of the operation will be undefined.

### *mode*

The coordinate mode of the points. The coordinates of the points can be relative to the drawable origin or relative to the previous point. The predefined values for **mode** are as follows:

VAX	C	Description
X\$C_COORD_MODE_ORIGIN	CoordModeOrigin	The point coordinates of each point are relative to the drawable origin.
X\$C_COORD_MODE_PREVIOUS	CoordModePrevious	The first point is relative to the drawable origin and each subsequent point is relative to the point preceding it.

Other values specified in this argument are not valid.

---

**DESCRIPTION**

FILL POLYGON fills in the area of a multi-sided shape in the specified window or pixmap (drawable). The fill pattern is specified by the fill style member in the associated graphics context.

Specify the polygon outline by defining point coordinates in the **points** array. Lines are drawn between the points. If the last point in the list does not coincide with the first point to close the polygon, it is closed by the routine. No pixel is drawn more than once.

The first point in the array is always relative to the origin of the drawable. Subsequent points can be drawn relative to the origin or relative to the previous point. Refer to Figure 6–16 for an illustration of how the mode changes the position of the points.

The polygon is filled according to the fill pattern and fill rule specified in the graphics context.

The **points** array uses a predefined point data structure. This data structure has members for the x- and y-coordinates that define one point.

For information about the point data structure, see Section 6.1.

FILL POLYGON uses these graphics context members:

- Function
- Plane Mask
- Fill Style
- Fill Rule
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

FILL POLYGON also uses the following mode-dependent members:

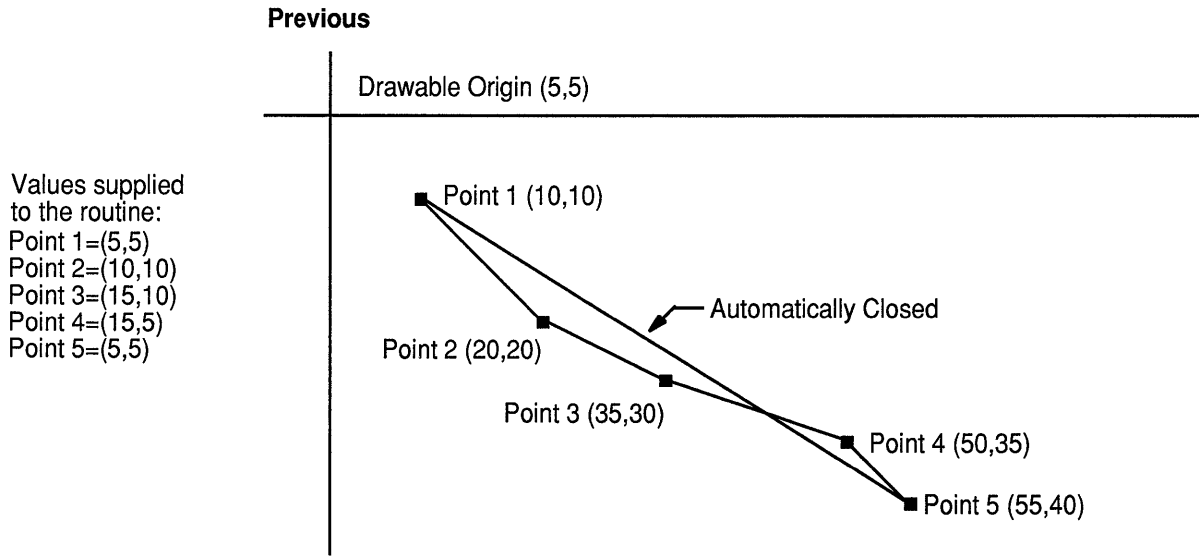
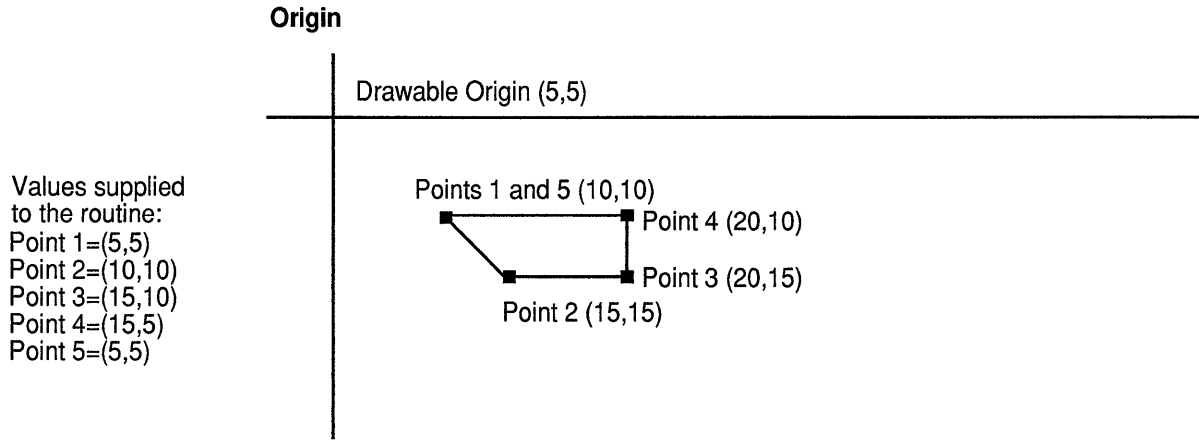
- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

For more information about the graphics context data structure and its members, see Section 5.1.

# Graphics Routines

## FILL POLYGON

Figure 6-16 Polygon Shapes Drawn in Different Coordinate Modes



ZK-0016A-GE

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---



## Graphics Routines

### FILL RECTANGLE

---

## FILL RECTANGLE

Fills the area defined by a rectangular outline in the specified drawable.

---

### VAX FORMAT

#### **X\$FILL\_RECTANGLE**

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference

---

---

### MIT C FORMAT

#### **XFillRectangle**

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, width, height)*

#### argument information

```
XFillRectangle(display, drawable_id, gc_id, x_coord, y_coord,
               width, height)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
unsigned int width, height;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) where the rectangle will be filled. The drawable identifier can be either a window or pixmap identifier.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

### ***x\_coord***

The x-coordinate of the rectangle. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

### ***y\_coord***

The y-coordinate of the rectangle. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

### ***width***

The width, in pixels, of the rectangle. The width and height define the size of the rectangular area to be filled.

### ***height***

The height, in pixels, of the rectangle. The width and height define the size of the rectangular area to be filled.

---

## DESCRIPTION

**FILL RECTANGLE** fills in a rectangular area in the specified window. The fill pattern used is specified in the fill style graphics context member.

**FILL RECTANGLE** uses the following members of the graphics context:

- Function
- Plane Mask
- Fill Style
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

**FILL RECTANGLE** also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

For more information about the graphics context data structure and its members, see Section 5.1.

When you want to fill more than one rectangular area, use **FILL RECTANGLES**. Using **FILL RECTANGLE** repeatedly is less efficient than using **FILL RECTANGLES**.

For information on drawing a rectangular outline, see the descriptions of the **DRAW RECTANGLE** and **DRAW RECTANGLES** routines.

## Graphics Routines

### FILL RECTANGLE

---

#### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## FILL RECTANGLES

Fills the areas defined by rectangular outlines in the specified drawable.

---

**VAX FORMAT**    **X\$FILL\_RECTANGLES**  
*(display, drawable\_id, gc\_id, rectangles,  
num\_rectangles)*

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
rectangles	array	x\$rectangle	read	reference
num_rectangles	longword	longword	read	reference

---

**MIT C FORMAT**    **XFillRectangles**  
*(display, drawable\_id, gc\_id, rectangles,  
num\_rectangles)*

**argument  
information**

```
XFillRectangles(display, drawable_id, gc_id, rectangles,
                num_rectangles)
Display *display;
Drawable drawable_id;
GC gc_id;
XRectangle *rectangles;
int num_rectangles;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***drawable\_id***

The identifier of the window or pixmap (drawable) where the rectangle will be filled. The drawable identifier can be either a window or pixmap identifier.

***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

## Graphics Routines

### FILL RECTANGLES

#### *rectangles*

A pointer to an array of rectangles, where each element defines a rectangular area to be filled. Each element is a rectangle data structure. The x- and y-coordinates define the upper left corner of the rectangle and are relative to the origin of the drawable. The rectangular area within each data structure is defined by the width and height. The length of the array is defined by **num\_rectangles**.

For information about the rectangle data structure, see Section 6.3.

#### *num\_rectangles*

The number of rectangular areas to be filled. This value specifies the length of the array in **rectangles**.

---

## DESCRIPTION

FILL RECTANGLES fills in rectangular areas in the specified window or pixmap (drawable). The area filled is within the outline defined by the rectangle. The fill pattern is specified by the fill style member in the associated graphics context.

The rectangles are filled in the order listed in the array **rectangles**. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the pixel is drawn multiple times.

For information about the rectangle data structure, see Section 6.3.

The x- and y-coordinates (in the **x\_coord** and **y\_coord** members) specify the position. The values you specify are relative to the origin of the drawable. The point defined by the coordinates is the upper left corner of the rectangle. The width and height (in the **width** and **height** members) specify the size.

FILL RECTANGLES uses the following members of the graphics context:

- Function
- Plane Mask
- Fill Style
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

FILL RECTANGLES also uses the following mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

## Graphics Routines

### FILL RECTANGLES

For information about the graphics context data structure and its members, see the Section 5.1.

When you want to fill one rectangular area, use `FILL RECTANGLE`.

For information on drawing a rectangular outline, see the descriptions of the `DRAW RECTANGLE` and `DRAW RECTANGLES` routines.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## Graphics Routines

### GET IMAGE

---

## GET IMAGE

Returns the contents of a rectangle to the specified image structure.

---

### VAX FORMAT

***status\_return = X\$GET\_IMAGE***

*(display, drawable\_id, x\_coord, y\_coord, width, height, plane\_mask, image\_format, ximage\_return)*

### argument information

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
plane_mask	mask_longword	uns longword	read	reference
image_format	longword	longword	read	reference
ximage_return	record	x\$image	write	reference

---

---

### MIT C FORMAT

***ximage\_return=XGetImage***

*(display, drawable\_id, x\_coord, y\_coord, width, height, plane\_mask, image\_format)*

### argument information

```
XImage *XGetImage(display, drawable_id, x_coord, y_coord, width,
                  height, plane_mask, image_format)
Display *display;
Drawable drawable_id;
int x_coord, y_coord;
unsigned int width, height;
long plane_mask;
int image_format;
```

---

### RETURNS

***status\_return (VAX only)***

Specifies whether or not the routine completed successfully.

***ximage\_return (MIT C only)***

A pointer to an image data structure containing the data requested.

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***drawable\_id***

The identifier of the window or pixmap (drawable) where the image data will be returned.

***x\_coord***

The x-coordinate of the rectangle to retrieve image data from. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***y\_coord***

The y-coordinate of the rectangle to retrieve image data from. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

***width***

The width, in pixels, of the rectangle.

***height***

The height, in pixels, of the rectangle.

***plane\_mask***

A bit mask specifying the planes to be retrieved.

***image\_format***

The format of the image. The predefined values for **format** are as follows:

VAX	C	Description
X\$C_XY_BITMAP	XYBitmap	XY bitmap format
X\$C_XY_PIXMAP	XPixmap	XY pixmap format
X\$C_Z_PIXMAP	ZPixmap	Z pixmap format

Other values specified in this argument are not valid.

***ximage\_return (VAX only)***

The returned image data structure containing the requested data.

---

**DESCRIPTION**

GET IMAGE returns an image data structure that contains the data for the rectangular image requested. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

If the value of **format** is XY bitmap or XY pixmap, only the bit planes specified in **plane\_mask** are returned. If the value of **format** is Z pixmap, all bits in all planes not specified in **plane\_mask** are set to zero.



## Graphics Routines

### GET IMAGE

GET IMAGE returns the depth of the image to the depth member of the image data structure. The depth is defined by the depth of the drawable when the drawable was created, except when getting a subset of the planes in XY Pixmap format, where the depth is given by the number of bits set to 1 in **plane\_mask**.

If the drawable is a window, the window must be mapped. The rectangular area would be contained within the window, including its borders, if the window were fully visible. In addition, the rectangular area must be fully contained within pixmaps. The returned image will include any visible portions of subwindows or overlapping windows contained in the rectangle.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

## GET PIXEL

Obtains the value of one pixel stored in an image.

---

**VAX FORMAT**    *pixel\_value\_return=X\$GET\_PIXEL*  
                  (*ximage, x\_coord, y\_coord*)

---

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
pixel_value_return	longword	uns longword	write	reference
ximage	record	x\$image	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference

---

---

**MIT C FORMAT**    *pixel\_value\_return=XGetPixel*  
                  (*ximage, x\_coord, y\_coord*)

---

**argument  
information**

```
unsigned long XGetPixel(ximage, x_coord, y_coord)
XImage *ximage;
int x_coord;
int y_coord;
```

---

**RETURNS**        *pixel\_value\_return*

The value of the pixel, as stored in the image. The pixel value is returned in normalized format (in other words, the least significant byte of the longword is the least significant byte of the pixel).

---

**ARGUMENTS**

***ximage***

The image data structure containing the pixel.

For information about the image data structure, see Section 6.5.

***x\_coord***

The x-coordinate of the pixel within the pixmap of the image. This coordinate is relative to the origin of the image. The x- and y-coordinates define the location of the pixel to obtain the value for.

***y\_coord***

The y-coordinate of the pixel within the pixmap of the image. This coordinate is relative to the origin of the image. The x- and y-coordinates define the location of the pixel to obtain the value for.

## Graphics Routines

### GET PIXEL

---

**DESCRIPTION** GET PIXEL returns a pixel value (index in a pixmap) for the specified pixel. The specified pixel is contained with the pixmap that stores the image. The image data structure specified in **ximage** specifies the exact pixmap.

---

## GET SUBIMAGE

Returns the contents of a rectangle to a location within a specified preexisting image structure.

---

### VAX FORMAT

***status\_return = X\$GET\_SUB\_IMAGE***  
*(display, drawable\_id, x\_coord, y\_coord, width, height, plane\_mask, image\_format, dst\_ximage\_return, dst\_x\_coord, dst\_y\_coord)*

### argument information

---

Argument	Usage	Data Type	Access	Mechanism
status_return	longword	longword	write	value
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
plane_mask	mask_longword	uns longword	read	reference
image_format	longword	longword	read	reference
dst_ximage_return	record	x\$image	write	reference
dst_x_coord	longword	longword	read	reference
dst_y_coord	longword	longword	read	reference

---



---

### MIT C FORMAT

***ximage\_return=XGetSubImage***  
*(display, drawable\_id, x\_coord, y\_coord, width, height, plane\_mask, image\_format, dst\_ximage\_return, dst\_x\_coord, dst\_y\_coord)*

## Graphics Routines

### GET SUBIMAGE

---

#### argument information

```
XImage *XGetSubImage(display, drawable_id, x_coord, y_coord,
                    width, height, plane_mask, image_format,
                    dst_ximage_return, dst_x_coord, dst_y_coord)
Display *display;
Drawable drawable_id;
int x_coord, y_coord;
unsigned int width, height;
long plane_mask;
int image_format;
XImage dst_ximage_return;
int dst_x_coord, dst_y_coord;
```

---

#### RETURNS

##### ***status\_return (VAX only)***

Return value that specifies whether the routine completed successfully.

##### ***ximage\_return (MIT C only)***

A pointer to a preexisting source image data structure containing the data requested.

---

#### ARGUMENTS

##### ***display***

The display information originally returned by OPEN DISPLAY.

##### ***drawable\_id***

The identifier of the window or pixmap (drawable) where the image data will be returned.

##### ***x\_coord***

The x-coordinate of the rectangle to retrieve image data from. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

##### ***y\_coord***

The y-coordinate of the rectangle to retrieve image data from. This coordinate is relative to the origin of the drawable. The x- and y-coordinates define the upper left corner of the rectangle.

##### ***width***

The width, in pixels, of the rectangle.

##### ***height***

The height, in pixels, of the rectangle.

##### ***plane\_mask***

A bit mask specifying the planes to be retrieved.

***image\_format***

The format of the image. The predefined values for **format** are as follows:

VAX	C	Description
X\$C_XY_BITMAP	XYBitmap	XY bitmap format
X\$C_XY_PIXMAP	XPixmap	XY pixmap format
X\$C_Z_PIXMAP	ZPixmap	Z pixmap format

Other values specified in this argument are not valid.

***dst\_ximage\_return***

The returned destination image data structure.

***dst\_x\_coord***

The x-coordinate of the destination rectangle. This coordinate is relative to the origin of the drawable. The destination x- and y-coordinates define the upper left corner of the destination rectangle and determine where the subimage will be placed within the destination image.

***dst\_y\_coord***

The y-coordinate of the destination rectangle. This coordinate is relative to the origin of the drawable. The destination x- and y-coordinates define the upper left corner of the destination rectangle and determine where the subimage will be placed within the destination image.

---

**DESCRIPTION**

GET SUBIMAGE returns the contents of a rectangle to a location within a specified preexisting image structure. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

If the value of **image\_format** is XY bitmap or XY pixmap, only the bit planes specified in **plane\_mask** are returned. If the value of **image\_format** is Z pixmap, all bits in all planes not specified in **plane\_mask** are set to zero.

GET SUB IMAGE does not update any fields in the destination image data structure. The depth of the destination image structure must be the same as the depth of the drawable. If the specified subimage does not fit at the specified location on the destination image, the right and bottom edges are clipped.

If the drawable is a window, the window must be mapped. The rectangular area would be contained within the window, including its borders, if the window were fully visible. In addition, the rectangular area must be fully contained within pixmaps. The returned image includes any visible portions of subwindows or overlapping windows contained in the rectangle.

# Graphics Routines

## GET SUBIMAGE

---

### X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

## PUT IMAGE

Combines the image in memory with the image in a specified rectangle on the specified drawable.

---

### VAX FORMAT

### X\$PUT\_IMAGE

*(display, drawable\_id, gc\_id, ximage, src\_x\_coord, src\_y\_coord, dst\_x\_coord, dst\_y\_coord, width, height)*

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
ximage	record	x\$image	read	reference
src_x_coord	longword	longword	read	reference
src_y_coord	longword	longword	read	reference
dst_x_coord	longword	longword	read	reference
dst_y_coord	longword	longword	read	reference
width	longword	longword	read	reference
height	longword	longword	read	reference

---

### MIT C FORMAT

### XPutImage

*(display, drawable\_id, gc\_id, ximage, src\_x\_coord, src\_y\_coord, dst\_x\_coord, dst\_y\_coord, width, height)*

#### argument information

```
XPutImage(display, drawable_id, gc_id, ximage, src_x_coord,
          src_y_coord, dst_x_coord, dst_y_coord, width, height)
Display *display;
Drawable drawable_id;
GC gc_id;
XImage *ximage;
int src_x_coord, src_y_coord;
int dst_x_coord, dst_y_coord;
unsigned int width, height;
```

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.



## Graphics Routines

### PUT IMAGE

#### ***drawable\_id***

The identifier of the window or pixmap (drawable) containing the image data. If the drawable is a window, the identifier was originally returned by CREATE SIMPLE WINDOW or CREATE WINDOW. If the drawable is a pixmap, the identifier was originally returned by CREATE PIXMAP.

#### ***gc\_id***

The identifier of the graphics context associated with the drawable. This identifier was originally returned by DEFAULT GC or CREATE GC.

#### ***ximage***

The image to be combined with the rectangle.

#### ***src\_x\_coord***

The x-coordinate specifying the offset from the left edge of the image in memory, specified by the image data structure. The source x- and y-coordinates specify the beginning point of the image in memory that will be combined with the image in the drawable.

#### ***src\_y\_coord***

The y-coordinate specifying the offset from the top edge of the image in memory, specified by the image data structure. The source x- and y-coordinates specify the beginning point of the image in memory that will be combined with the image in the drawable.

#### ***dst\_x\_coord***

The x-coordinate of the rectangle defining the image in the drawable. This coordinate is relative to the origin of the drawable. The destination x- and y-coordinates define the upper left corner in the drawable that will be combined with the image in memory.

#### ***dst\_y\_coord***

The y-coordinate of the rectangle defining the image in the drawable. This coordinate is relative to the origin of the drawable. The destination x- and y-coordinates define the upper left corner in the drawable that will be combined with the image in memory.

#### ***width***

The width of the rectangular area defining the image in memory to be combined with the image in the drawable.

#### ***height***

The height of the rectangular area defining the image in memory to be combined with the image in the drawable.

---

## DESCRIPTION

PUT IMAGE combines an image in memory with an image in a drawable.

Define the portion of the image in memory that you want to combine. That subimage is combined with the image that is currently displayed in the drawable at the position specified by ***dst\_x\_coord*** and ***dst\_y\_coord***. If XY bitmap format is used, then the depth must be one. For XY pixmap and Z pixmap, the depth must match the depth of the drawable. For XY pixmap, the image must be sent in XY format. For Z pixmap, the image must be sent in the Z format defined for the given depth.

PUT IMAGE uses the following graphics context members:

- Function
- Plane Mask
- Subwindow Mode
- Clip X Origin
- Clip Y Origin
- Clip Mask

For XY bitmap images only, PUT IMAGE also uses the foreground and background mode-dependent graphics context members. The foreground value is used for all one bits in the image and the background value is used for all zero bits in the image.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are as follows: <ul style="list-style-type: none"> <li>• In a graphics request, the root and depth of the graphics context do not match those of the drawable.</li> <li>• An input-only window is used as a drawable.</li> <li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li> <li>• An input-only window lacks this attribute.</li> </ul>
X\$C_BAD_VALUE	BadValue	Some numeric values fall outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

---

## Graphics Routines

### PUT PIXEL

---

## PUT PIXEL

Changes one pixel value within the pixmap containing an image.

---

**VAX FORMAT**    **X\$PUT\_PIXEL**  
(*ximage, x\_coord, y\_coord, pixel*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
ximage	record	x\$image	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
pixel	longword	uns longword	read	reference

---

---

**MIT C FORMAT**    **XPutPixel**  
(*ximage, x\_coord, y\_coord, pixel*)

**argument  
information**

```
int XPutPixel(ximage, x_coord, y_coord, pixel)
    XImage *ximage;
    int x_coord;
    int y_coord;
    unsigned long pixel;
```

---

**ARGUMENTS**    ***ximage***

The image data structure containing the pixel to be changed.

For information about the image data structure, see Section 6.5.

***x\_coord***

The x-coordinate of the pixel within the pixmap of the image. This coordinate is relative to the origin of the image. The x- and y-coordinates define the location of the pixel to be changed.

***y\_coord***

The y-coordinate of the pixel within the pixmap of the image. This coordinate is relative to the origin of the image. The x- and y-coordinates define the location of the pixel to be changed.

***pixel***

The new value of the pixel. The pixel value must be in normalized format (in other words, the least significant byte of the longword is the least significant byte of the pixel).

---

**DESCRIPTION**

PUT PIXEL changes the value of one pixel within a pixmap containing an image. The pixmap is specified with the image data structure. You specify the new pixel value in **pixel**.

## Graphics Routines

### SUBIMAGE

---

## SUBIMAGE

Creates a new image from an existing image.

---

### VAX FORMAT

#### **X\$SUB\_IMAGE**

*(ximage, x\_coord, y\_coord, width, height, sub\_image\_return)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
ximage	record	x\$image	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
width	longword	uns longword	read	reference
height	longword	uns longword	read	reference
sub_image_return	record	x\$image	write	reference

---

---

### MIT C FORMAT

#### **sub\_image\_return=XSubImage**

*(ximage, x\_coord, y\_coord, width, height)*

#### argument information

```
XImage *XSubImage(ximage, x_coord, y_coord, width, height)
    XImage *ximage;
    int x_coord;
    int y_coord;
    int width;
    int height;
```

---

### RETURNS

#### **sub\_image\_return (C only)**

A pointer to the image data structure created for the new sub image.

---

### ARGUMENTS

#### **ximage**

A pointer to the image data structure that references the existing image to create the new image.

#### **x\_coord**

The x-coordinate of the existing image. This coordinate is relative to origin of the image. The x- and y-coordinates define the point within the existing image where copying will start.

***y\_coord***

The y-coordinate of the existing image. This coordinate is relative to origin of the image. The x- and y-coordinates define the point within the existing image where copying will start.

***width***

The width, in pixels, of the new subimage. The width and height determine the area of the new subimage.

***height***

The height, in pixels, of the new subimage. The width and height determine the area of the new subimage.

***sub\_image\_return (VAX only)***

A pointer to the image data structure created for the new subimage.

---

**DESCRIPTION**

SUBIMAGE copies a portion of an existing image to create a new image. In the MIT C binding, it also allocates memory sufficient for a new image data structure that references the subimage; the pointer to the subimage's data structure is returned.

SUBIMAGE creates the new image by repeating GET PIXEL and PUT PIXEL calls. Therefore, this operation might be slow.



# 7

## Text Routines

---

Use the text routines to perform the following tasks:

- Computing string sizes
- Returning the logical extents of strings
- Drawing text

For concepts related to text routines and information on how to use text routines, see the *VMS DECwindows Xlib Programming Volume*.

The routines described in this chapter are listed in Table 7–1.

**Table 7–1 Text Routines**

<b>Routine Name</b>	<b>Description</b>
DRAW IMAGE STRING	Draws a string of 8-bit character image text to the screen.
DRAW IMAGE STRING 16	Draws a string of 16-bit character image text to the screen.
DRAW STRING	Draws a text string (8-bit characters) to the screen in a single font.
DRAW STRING 16	Draws a text string (16-bit characters) to the screen in a single font.
DRAW TEXT	Draws text (8-bit characters) to the screen in various fonts.
DRAW TEXT 16	Draws text (16-bit characters) to the screen in various fonts.
QUERY TEXT EXTENTS	Returns the logical extents of an 8-bit character string by querying the server.
QUERY TEXT EXTENTS 16	Returns logical extents of an 16-bit character string by querying the server.
TEXT EXTENTS	Returns logical extents of an 8-bit character string.
TEXT EXTENTS 16	Returns logical extents of an 16-bit character string.
TEXT WIDTH	Returns the length of a string composed of 8-bit characters, given the string and the font in which the string is to be written.
TEXT WIDTH 16	Returns the length of a string composed of 16-bit characters, given the string and the font in which the string is to be written.

---



# Text Routines

## 7.1 Drawing Text

### 7.1 Drawing Text

DRAW TEXT and DRAW TEXT 16 output arrays of text defined in text item structures. Each binding includes a text item structure for 8-bit strings and a text item structure for 16-bit strings. The following sections describe these structures for each binding.

#### 7.1.1 Text Item 8-Bit Data Structure

The data structure for the VAX binding is shown in Figure 7-1, and information about members in the data structure is described in Table 7-2.

Figure 7-1 Text Item Data Structure (VAX Binding)

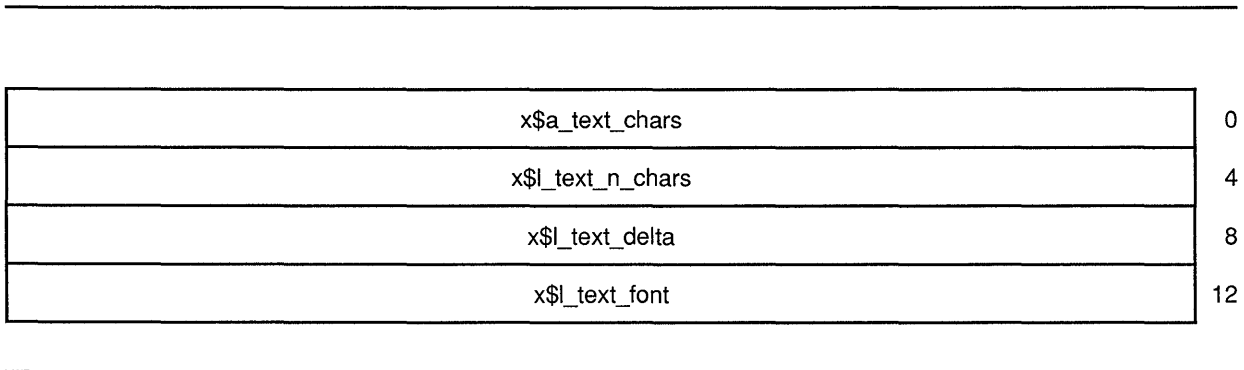


Table 7-2 Members of the Text Item Data Structure (VAX Binding)

Member Name	Contents
X\$a_TEXT_CHARS	Address of a string of characters.
X\$l_TEXT_N_CHARS	Number of characters in the string.
X\$l_TEXT_DELTA	Spacing before the start of the string. Spacing is always added to the string origin and is not dependent on the font used.
X\$l_TEXT_FONT	Identifier of the font used to print the string. If the value of this member is x\$c_none, the server uses the current font in the GC data structure. If the member has a value other than x\$c_none, the specified font is stored in the GC data structure.

The data structure for the MIT C binding is shown in Figure 7-2, and information about members in the data structure is described in Table 7-3.

**Figure 7–2 Text Item Data Structure (MIT C Binding)**

```
typedef struct {
    char *chars;
    int nchars;
    int delta;
    Font font;
} XTextItem;
```

**Table 7–3 Members of the Text Item Data Structure (MIT C Binding)**

Member Name	Contents
chars	The address of a string of characters.
nchars	Number of characters in the string.
delta	Spacing before the start of the string. Spacing is always added to the string origin and is not dependent on the font used.
font	Identifier of the font used to print the string. If the value of this member is None, the server uses the current font in the GC data structure. If the member has a value other than None, the specified font is stored in the GC data structure.

## 7.1.2 Text Item 16-Bit Data Structure

The data structure for the VAX binding is shown in Figure 7–3, and information about members in the data structure is described in Table 7–4.

**Figure 7–3 Text Item 16 Data Structure (VAX Binding)**

x\$a_tx16_chars	0
x\$l_tx16_n_chars	4
x\$l_tx16_delta	8
x\$l_tx16_font	12

**Table 7–4 Members of the Text Item 16 Data Structure (VAX Binding)**

Member Name	Contents
X\$a_TX16_CHARS	Address of a string of characters stored in a char 2b data structure.

(continued on next page)

## Text Routines

### 7.1 Drawing Text

**Table 7-4 (Cont.) Members of the Text Item 16 Data Structure (VAX Binding)**

Member Name	Contents
X\$L_TX16_N_CHARS	Number of characters in the string.
X\$L_TX16_DELTA	Spacing before the start of the string. Spacing is always added to the string origin and is not dependent on the font used.
X\$L_TX16_FONT	Identifier of the font used to print the string. If the value of this member is x\$c_none, the server uses the current font in the GC data structure. If the member has a value other than x\$c_none, the specified font is stored in the GC data structure.

The data structure for the MIT C binding is shown in Figure 7-4, and information about members in the data structure is described in Table 7-5.

**Figure 7-4 Text Item 16 Data Structure (MIT C Binding)**

```
typedef struct {
    XChar2b *chars;
    int nchars;
    int delta;
    Font font;
} XTextItem16;
```

**Table 7-5 Members of the Text Item 16 Data Structure (MIT C Binding)**

Member Name	Contents
chars	The address of a string of characters stored in a char 2b data structure. For a description of the char 2b data structure, see the description immediately following this table.
nchars	Number of characters in the string.
delta	Spacing before the start of the string. Spacing is always added to the string origin and is not dependent on the font used.
font	Identifier of the font used to print the string. If the value of this member is None, the server uses the current font in the GC data structure. If the member has a value other than None, the specified font is stored in the GC data structure.

## 7.2 Text Routines

The following pages describe the Xlib text routines.

## DRAW IMAGE STRING

Draws a string of 8-bit character image text to the screen.

**VAX FORMAT**    **X\$DRAW\_IMAGE\_STRING**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string)*

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
string	char_string	character string	read	descriptor

**MIT C FORMAT**    **XDrawImageString**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string, length)*

**argument  
information**

```
XDrawImageString(display, drawable_id, gc_id, x_coord,
                 y_coord, string, length)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
char *string;
int length;
```

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***drawable\_id***

The identifier of the window or pixmap (drawable) in which you want to write the text. This identifier is returned when you create the drawable.

***gc\_id***

The identifier of the graphics context to be used for writing the text. This identifier is returned when you create a graphics context using the CREATE GC routine.

## Text Routines

### DRAW IMAGE STRING

For image text to be visible, the foreground and background values in the graphics context must be different. Otherwise, you get a filled rectangle which may or may not show up, depending on the current window background color. This is not true, however, if the text string extends outside the background rectangle on the left or right.

#### ***x\_coord***

The x-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

#### ***y\_coord***

The y-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

#### ***string***

The text string to be written to the screen.

#### **VAX only**

The **string** argument is the address of a descriptor that points to a string.

#### ***length (MIT C only)***

The length of the string. A null character can be a valid representation, so null termination cannot be used to determine the length of the string.

---

## DESCRIPTION

DRAW IMAGE STRING draws a string of 8-bit character image text to the screen. Image text is drawn by painting *both* the foreground and background pixels of each character, using a single font.

First, a destination rectangle is drawn using the specified background color defined in graphics context. The upper left corner of the rectangle is point (x, y), defined as follows:

- x = The x-coordinate of the baseline starting point.
- y = The y-coordinate of the baseline starting point minus the font ascent. (Font ascent is measured from top to bottom.)

The width of the destination rectangle is the overall width of all characters in the string (the sum of all the widths).

The height of the rectangle is the sum of the font ascent and the font descent.

Leftbearing, rightbearing, font ascent, and font descent values are returned by QUERY TEXT EXTENTS.

After the destination rectangle is drawn, text is drawn using the specified foreground color.

DRAW IMAGE STRING uses the following graphics context members:

- Plane mask
- Foreground
- Background
- Font

- Subwindow mode
- Clip x origin
- Clip y origin
- Clip mask

For more information about graphics context members, see Chapter 5.

This routine disregards any function or fill style specified in the graphics context. DRAW IMAGE STRING always uses GXcopy as the function and solid as the fill style.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## Text Routines

### DRAW IMAGE STRING 16

---

## DRAW IMAGE STRING 16

Draws a string of 16-bit character image text to the screen.

---

**VAX FORMAT**    **X\$DRAW\_IMAGE\_STRING\_16**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string16, length)*

**argument information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
string16	array	word	read	reference
length	word	uns word	read	reference

---

---

**MIT C FORMAT**    **XDrawImageString16**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string16, length)*

**argument information**

```
XDrawImageString16(display, drawable_id, gc_id, x_coord, y_coord, string16, length)
    Display *display;
    Drawable drawable_id;
    GC gc_id;
    int x_coord, y_coord;
    XChar2b *string16;
    int length;
```

---

### ARGUMENTS

***display***  
The display information originally returned by OPEN DISPLAY.

***drawable\_id***  
The identifier of the window or pixmap (drawable) in which you want to write the text. This identifier is returned when you create the drawable.

***gc\_id***

The identifier of the graphics context to be used for writing the text. This identifier is returned when you create a graphics context using the CREATE GC routine.

For image text to be visible, the foreground and background values in the graphics context must be different. Otherwise, you get a filled rectangle which may or may not show up, depending on the current window background color. This is not true, however, if the text string extends outside the background rectangle on the left or right.

***x\_coord***

The x-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

***y\_coord***

The y-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

***string16***

The text string to be written to the screen.

***length***

The length of the string. A null character may be a valid representation, so null termination cannot be used to determine the length of the string.

---

**DESCRIPTION**

DRAW IMAGE STRING 16 draws a string of 16-bit character image text to the screen. Image text is drawn by painting *both* the foreground and background pixels of each character, using a single font.

First, a destination rectangle is drawn using the specified background color defined in graphics context. The upper left corner of the rectangle is point (x,y), defined as follows:

- x = The x-coordinate of the baseline starting point.
- y = The y-coordinate of the baseline starting point minus the font ascent. (Font ascent is measured from top to bottom.)

The width of the destination rectangle is the overall width of all characters in the string.

The height of the rectangle is the sum of the font ascent and the font descent.

Leftbearing, rightbearing, font ascent, and font descent values are returned by QUERY TEXT EXTENTS 16.

After the destination rectangle is drawn, text is drawn using the specified foreground color.

DRAW IMAGE STRING 16 uses the following graphics context members:

- Plane mask
- Foreground
- Background



## Text Routines

### DRAW IMAGE STRING 16

- Font
- Subwindow mode
- Clip x origin
- Clip y origin
- Clip mask

For more information about graphics context members, see Chapter 5.

This routine disregards any function or fill style specified in the graphics context. `DRAW IMAGE STRING` always uses `GXcopy` as the function and `solid` as the fill style.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

---

## DRAW STRING

Draws a text string (8-bit characters) to the screen in a single font, drawing only foreground pixels.

---

### VAX FORMAT

### X\$DRAW\_STRING

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
string	char_string	character string	read	descriptor

---



---

### MIT C FORMAT

### XDrawString

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string, length)*

#### argument information

```
XDrawString(display, drawable_id, gc_id, x_coord, y_coord,
            string, length)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
char *string;
int length;
```

---

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***drawable\_id***

The identifier of the drawable in which you want to write the text. The drawable can be either a window or a pixmap. This identifier is returned when you create the drawable.

#### ***gc\_id***

The identifier of the graphics context to be used for writing the text. This identifier is returned when you create a graphics context using the CREATE GC routine.

## Text Routines

### DRAW STRING

#### ***x\_coord***

The x-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

#### ***y\_coord***

The y-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

#### ***string***

A null-terminated string to be written to the screen.

#### ***length (MIT C only)***

The length of the string specified.

Because a null character can be a valid representation, null termination cannot be used to determine the length of the string.

---

## DESCRIPTION

DRAW STRING writes 8-bit character text to the screen using the font specified in the graphics context, drawing only foreground pixels.

DRAW STRING uses the following graphics context members to control the appearance of the text on the screen:

- Function
- Plane mask
- Fill style
- Font
- Subwindow mode
- Clip x origin
- Clip y origin
- Clip mask

Each character image defined by the font member of graphics context is treated as an additional mask for a fill operation on the drawable.

DRAW STRING also uses the following graphics context mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

For more information on graphics context, see the Chapter 5.

---

**X ERRORS**

<b>VAX</b>	<b>C</b>	<b>Description</b>
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## Text Routines

### DRAW STRING 16

---

## DRAW STRING 16

Draws a text string (16-bit characters) to the screen in a single font, using only foreground pixels.

---

**VAX FORMAT**    **X\$DRAW\_STRING\_16**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string16, length)*

**argument information**

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
string16	array	word	read	reference
length	word	word	read	reference

---

---

**MIT C FORMAT**    **XDrawString16**  
*(display, drawable\_id, gc\_id, x\_coord, y\_coord, string16, length)*

**argument information**

```
XDrawString16(display, drawable_id, gc_id, x_coord, y_coord,
              string16, length)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
XChar2b *string16;
int length;
```

---

**ARGUMENTS**

***display***

The display information originally returned by OPEN DISPLAY.

***drawable\_id***

The identifier of the drawable in which you want to write the text. The drawable can be either a window or a pixmap. This identifier is returned when you create the drawable.

***gc\_id***

The identifier of the graphics context to be used for writing the text. This identifier is returned when you create a graphics context using the CREATE GC routine.

***x\_coord***

The x-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

***y\_coord***

The y-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

***string16***

A null-terminated string to be written to the screen.

***length***

The length of the string specified.

Because a null character can be a valid representation, null termination cannot be used to determine the length of the string.

---

**DESCRIPTION**

DRAW STRING 16 writes 16-bit character text to the screen using the font specified in the graphics context, drawing only foreground pixels.

DRAW STRING 16 uses the following graphics context members to control the appearance of the text on the screen:

- Function
- Plane mask
- Fill style
- Font
- Subwindow mode
- Clip x origin
- Clip y origin
- Clip mask

Each character image defined by the font member of graphics context is treated as an additional mask for a fill operation on the drawable.

DRAW STRING 16 also uses the following graphics context mode-dependent members:

- Foreground
- Background
- Tile

## Text Routines

### DRAW STRING 16

- Stipple
- Ts x origin
- Ts y origin

For more information about graphics context, see Chapter 5.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## DRAW TEXT

Draws text (8-bit characters) to the screen in various fonts.

---

### VAX FORMAT

### X\$DRAW\_TEXT

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, items, num\_items)*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
items	array	vector_longword	read	reference
num_items	longword	longword	read	reference

---



---

### MIT C FORMAT

### XDrawText

*(display, drawable\_id, gc\_id, x\_coord, y\_coord, items, num\_items)*

#### argument information

```
XDrawText(display, drawable_id, gc_id, x_coord, y_coord, items,
          num_items)
Display *display;
Drawable drawable_id;
GC gc_id;
int x_coord, y_coord;
XTextItem *items;
int num_items;
```

---

### ARGUMENTS

#### *display*

The display information originally returned by OPEN DISPLAY.

#### *drawable\_id*

The identifier of the drawable in which you want to write the text. The drawable can be either a window or a pixmap. This identifier is returned when you create the drawable.



## Text Routines

### DRAW TEXT

#### ***gc\_id***

The identifier of the graphics context to be used for writing the text. This identifier is returned when you create a graphics context using the CREATE GC routine.

#### ***x\_coord***

The x-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

#### ***y\_coord***

The y-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

#### ***items***

An array of text item structures. The number of text item structures in the array is specified by **num\_items**.

#### ***num\_items***

The number of items (elements) in the specified text item array.

---

## DESCRIPTION

DRAW TEXT writes 8-bit character text to the screen using one or more fonts. The text is defined in an array of text item structures. Each text item structure contains the following information:

- Pointer to the text string
- Number of characters in the string
- Change in horizontal x position of the text before it is mapped to the drawable

Change in text position is relative to the end of the last string. The end of the last string is calculated as the starting position on the last string, plus the sum of the character widths for all characters in the string.

- Identifier of the font to be used to draw the string

For more information about the text item structure, see Section 7.1.1.

Text items are specified in the array in the order in which the text is written. That is, if DRAW TEXT is used to write three words, one in 10-point Times Roman, one in 12-point Times Roman, and one in 24-point Times Roman, the item array would have three elements, each describing the word and its respective font.

DRAW TEXT uses the following graphics context members to control the appearance of the text on the screen:

- Function
- Plane mask
- Fill style
- Subwindow mode
- Clip x origin

- Clip y origin
- Clip mask

If a font is not specified in the text item structure, the graphics context default font is used.

Each character image defined by the font member of graphics context is treated as an additional mask for a fill operation on the drawable.

DRAW TEXT also uses the following graphics context mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

For more information about graphics context, see Chapter 5.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

---

## Text Routines

### DRAW TEXT 16

---

## DRAW TEXT 16

Draws text (16-bit characters) to the screen in various fonts.

---

**VAX FORMAT**    **X\$DRAW\_TEXT\_16**  
(*display, drawable\_id, gc\_id, x\_coord, y\_coord,*  
*items16, num\_items*)

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
drawable_id	identifier	uns longword	read	reference
gc_id	identifier	uns longword	read	reference
x_coord	longword	longword	read	reference
y_coord	longword	longword	read	reference
items16	array	vector_longword	read	reference
num_items	longword	longword	read	reference

---

---

**MIT C FORMAT**    **XDrawText16**  
(*display, drawable\_id, gc\_id, x\_coord, y\_coord,*  
*items16, num\_items*)

#### argument information

---

```
XDrawText16(display, drawable_id, gc_id, x_coord, y_coord,  
            items16, num_items)  
    Display *display;  
    Drawable drawable_id;  
    GC gc_id;  
    int x_coord, y_coord;  
    XTextItem16 *items16;  
    int num_items;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***drawable\_id***  
The identifier of the drawable in which you want to write the text. The drawable can be either a window or a pixmap. This identifier is returned when you create the drawable.

***gc\_id***

The identifier of the graphics context to be used for writing the text. This identifier is returned when you create a graphics context using the CREATE GC routine.

***x\_coord***

The x-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

***y\_coord***

The y-coordinate of the starting point of the text baseline. Coordinates are specified in pixels relative to the origin of the drawable.

***items16***

An array of text item 16 structures. The number of text item 16 structures in the array is specified by **num\_items**.

***num\_items***

The number of items (elements) in the specified text item array.

---

**DESCRIPTION**

DRAW TEXT 16 writes 16-bit character text to the screen using one or more fonts. The text is defined in an array of text item 16 structures. Each text item 16 structure contains the following information:

- Pointer to the text string
  - Number of characters in the string
  - Change in horizontal x position of the text before it is mapped to the drawable
- Change in text position is relative to the end of the last string. The end of the last string is calculated as the starting position on the last string, plus the sum of the character widths for all characters in the string.
- Identifier of the font to be used to draw the string

For more information about the text item 16 structure, see Section 7.1.2.

Text items are specified in the array in the order in which the text is written. That is, if DRAW TEXT 16 is used to write three words, one in 10-point Times Roman, one in 12-point Times Roman, and one in 24-point Times Roman, the item array would have three elements, each describing the word and its respective font.

DRAW TEXT 16 uses the following graphics context members to control the appearance of the text on the screen:

- Function
- Plane mask
- Fill style
- Subwindow mode
- Clip x origin

## Text Routines

### DRAW TEXT 16

- Clip y origin
- Clip mask

If a font is not specified in the text item 16 structure, the graphics context default font is used.

Each character image defined by the font member of graphics context is treated as an additional mask for a fill operation on the drawable.

DRAW TEXT 16 also uses the following graphics context mode-dependent members:

- Foreground
- Background
- Tile
- Stipple
- Ts x origin
- Ts y origin

For more information about graphics context, see Chapter 5.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_DRAWABLE	BadDrawable	A value that you specified for a drawable argument does not name a defined window or pixmap.
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.
X\$C_BAD_MATCH	BadMatch	Possible causes are: <ul style="list-style-type: none"><li>• In a graphics request, the root and depth of the graphics context does not match that of the drawable.</li><li>• An input-only window is used as a drawable.</li><li>• One argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.</li><li>• An input-only window lacks this attribute.</li></ul>

## QUERY TEXT EXTENTS

Returns the logical extents of an 8-bit character string by querying the server.

### VAX FORMAT

### X\$QUERY\_TEXT\_EXTENTS

*(display, font\_id, string [,direction\_return]  
[,ascent\_return] [,descent\_return] [,overall\_return])*

#### argument information

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
font_id	identifier	uns longword	read	reference
string	char_string	character string	read	descriptor
direction_return	longword	longword	write	reference
ascent_return	longword	longword	write	reference
descent_return	longword	longword	write	reference
overall_return	record	x\$char_struct	write	reference

### MIT C FORMAT

### XQueryTextExtents

*(display, font\_id, string, num\_chars, direction\_return,  
ascent\_return, descent\_return, overall\_return)*

#### argument information

```
XQueryTextExtents(display, font_id, string, num_chars,
                  direction_return, ascent_return,
                  descent_return, overall_return)
Display *display;
XID font_id;
char *string;
int num_chars;
int *direction_return;
int *ascent_return, *descent_return;
XCharStruct *overall_return;
```

### ARGUMENTS

#### ***display***

The display information originally returned by OPEN DISPLAY.

#### ***font\_id***

The identifier of the font whose extents are being queried. The font identifier is returned by LOAD FONT. The address of the font data structure, which includes the font identifier, is returned by LOAD QUERY FONT.

## Text Routines

### QUERY TEXT EXTENTS

#### ***string***

The character string whose logical extents are being queried.

#### ***num\_chars (MIT C only)***

The number of characters in the string whose extents are being queried.

#### ***direction\_return***

The direction the string is painted on the screen. The **direction\_return** argument is the returned value of the direction element of the font data structure.

#### **VAX only**

This argument is optional.

#### ***ascent\_return***

The maximum ascent of the font used to draw the string. The **ascent\_return** argument is the returned value of the font ascent element of the font data structure.

#### **VAX only**

This argument is optional.

#### ***descent\_return***

The maximum descent of the font used to draw the string. The **descent\_return** argument is the returned value of the font descent element of the font data structure.

#### **VAX only**

This argument is optional.

#### ***overall\_return***

The minimum left bearing, maximum right bearing, string width, maximum character ascent, and maximum character descent.

Values are returned from a character structure.

For information about the character structure, see Chapter 13.

#### **VAX only**

This argument is optional.

---

## DESCRIPTION

QUERY TEXT EXTENTS returns the logical extents of a specified 8-bit string. Unlike TEXT EXTENTS, which determines the sizes of characters directly from the font data structure, QUERY TEXT EXTENTS computes the size of the string by querying the server. Because this requires extra overhead, use QUERY TEXT EXTENTS only when LOAD FONT has loaded the font used to write the string specified by **string**.

Given an 8-bit string, QUERY TEXT EXTENTS returns the following information about the extents:

- Direction the font is painted
- Ascent above the baseline used for determining line spacing

## Text Routines

### QUERY TEXT EXTENTS

- Descent below the baseline used for determining line spacing
- Character extents

The following string extents are returned:

Extent	Value
Leftbearing	The minimum left bearing of all characters in the string
Rightbearing	The maximum right bearing of all characters in the string
Width	The sum of all character widths in the string
Ascent	The maximum ascent of all characters in the string
Descent	The maximum descent of all characters in the string

When a font has no defined default character, undefined characters in the specified string are given zero character metric values.

---

## X ERRORS

QUERY TEXT EXTENTS generates the following errors:

VAX	C	Description
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.



## Text Routines

### QUERY TEXT EXTENTS 16

---

## QUERY TEXT EXTENTS 16

Returns logical extents of a 16-bit character string by querying the server.

---

**VAX FORMAT**    **X\$QUERY\_TEXT\_EXTENTS\_16**  
*(display, font\_id, string16, num\_chars  
[,direction\_return] [,ascent\_return] [,descent\_return]  
[,overall\_return])*

#### argument information

---

Argument	Usage	Data Type	Access	Mechanism
display	identifier	uns longword	read	reference
font_id	identifier	uns longword	read	reference
string16	array	word	read	reference
num_chars	word	word	read	reference
direction_return	longword	longword	write	reference
ascent_return	longword	longword	write	reference
descent_return	longword	longword	write	reference
overall_return	record	x\$char_struct	read	reference

---

---

**MIT C FORMAT**    **XQueryTextExtents16**  
*(display, font\_id, string16, num\_chars,  
direction\_return, ascent\_return, descent\_return,  
overall\_return)*

#### argument information

```
XQueryTextExtents16(display, font_id, string16, num_chars,  
                    direction_return, ascent_return,  
                    descent_return, overall_return)  
  
Display *display;  
XID font_id;  
XChar2b *string16;  
int num_chars;  
int *direction_return;  
int *ascent_return, *descent_return;  
XCharStruct *overall_return;
```

---

**ARGUMENTS**    ***display***  
The display information originally returned by OPEN DISPLAY.

***font\_id***

The identifier of the font whose extents you want to know. The font identifier is returned by LOAD FONT. The address of the font data structure, which includes the font identifier, is returned by LOAD QUERY FONT.

***string16***

The character string whose logical extents are being queried.

***num\_chars***

The number of characters in the string whose extents are being queried.

***direction\_return***

The direction the string is painted on the screen. The **direction\_return** argument is the returned value of the direction element of the font data structure.

**VAX only**

This argument is optional.

***ascent\_return***

Maximum ascent of the font used to draw the string. The **ascent\_return** argument is the returned value of the font ascent element of the font data structure.

**VAX only**

This argument is optional.

***descent\_return***

Maximum descent of the font used to draw the string. The **descent\_return** argument is the returned value of the font descent element of the font data structure.

**VAX only**

This argument is optional.

***overall\_return***

The minimum left bearing, maximum right bearing, string width, maximum character ascent, and maximum character descent.

Values are returned from a character structure.

For information about the structure, see Chapter 13.

**VAX only**

This argument is optional.

---

**DESCRIPTION**

QUERY TEXT EXTENTS 16 returns the logical extents of a specified 16-bit string. Unlike TEXT EXTENTS 16, which determines the sizes of characters directly from the font data structure, QUERY TEXT EXTENTS 16 computes the size of the string by querying the server. Because this results in additional overhead, use QUERY TEXT EXTENTS 16 only when

## Text Routines

### QUERY TEXT EXTENTS 16

LOAD FONT has loaded the font used to write the string specified by **string16**.

Given a 16-bit string, QUERY TEXT EXTENTS 16 returns the following information about the extents:

- Direction the font is painted
- Ascent above the baseline used for determining line spacing
- Descent below the baseline used for determining line spacing
- Character extents

The following string extents are returned:

Extent	Value
Leftbearing	The minimum left bearing of all characters in the string
Rightbearing	The maximum right bearing of all characters in the string
Width	The sum of all character widths in the string
Ascent	The maximum ascent of all characters in the string
Descent	The maximum descent of all characters in the string

When a font has no defined default character, undefined characters in the specified string are given zero character metric values.

---

## X ERRORS

VAX	C	Description
X\$C_BAD_FONT	BadFont	A value that you specified for a font argument does not name a defined font (or, in some cases, graphics context).
X\$C_BAD_GC	BadGC	A value that you specified for a graphics context argument does not name a defined graphics context.

---

## TEXT EXTENTS

Returns logical extents of an 8-bit character string in a given font.

---

### VAX FORMAT

### X\$TEXT\_EXTENTS

*(font\_ptr, string [,direction\_return] [,ascent\_return]  
[,descent\_return] [,overall\_return])*

#### argument information

Argument	Usage	Data Type	Access	Mechanism
font_ptr	record	x\$font_struct	read	reference
string	char_string	character string	read	descriptor
direction_return	longword	longword	write	reference
ascent_return	longword	longword	write	reference
descent_return	longword	longword	write	reference
overall_return	record	x\$char_struct	write	reference

---

### MIT C FORMAT

### XTextExtents

*(font\_ptr, string, num\_chars, direction\_return,  
ascent\_return, descent\_return, overall\_return)*

#### argument information

```
XTextExtents(font_ptr, string, num_chars, direction_return,
             ascent_return, descent_return, overall_return)
XFontStruct *font_ptr;
char *string;
int num_chars;
int *direction_return;
int *ascent_return, *descent_return;
XCharStruct *overall_return;
```

---

### ARGUMENTS

#### **font\_ptr**

The address of a font data structure. TEXT EXTENTS uses the character information contained in the font data structure to determine extents. The address of the font data structure is returned by LOAD QUERY FONT or QUERY FONT.

#### **string**

The character string whose logical extents are being queried.

#### **num\_chars (MIT C only)**

The number of characters in the string whose extents are being queried.

## Text Routines

### TEXT EXTENTS

#### ***direction\_return***

The direction the font is painted on the screen. The **direction\_return** argument is the returned value of the direction element of the font data structure.

#### **VAX only**

This argument is optional.

#### ***ascent\_return***

The maximum ascent of the font that draws the string. The **ascent\_return** argument is the returned value of the font ascent element of the font data structure.

#### **VAX only**

This argument is optional.

#### ***descent\_return***

Maximum descent of the font that draws the string. The **descent\_return** argument is the returned value of the font descent element of the font data structure.

#### **VAX only**

This argument is optional.

#### ***overall\_return***

The minimum left bearing, maximum right bearing, string width, maximum character ascent, and maximum character descent.

For more information about the character structure, see Chapter 13.

#### **VAX only**

This argument is optional.

---

## DESCRIPTION

TEXT EXTENTS returns the logical extents of a specified 8-bit string in a given font. Unlike QUERY TEXT EXTENTS, which queries the server for the sizes of characters in the string, TEXT EXTENTS uses the pointer returned by LOAD QUERY FONT to query the font data structure directly.

Given an 8-bit string, TEXT EXTENTS returns the following information about the extents:

- Direction the font is painted
- Ascent above the baseline used for determining line spacing
- Descent below the baseline used for determining line spacing
- Character extents

## Text Routines

### TEXT EXTENTS

The following string extents are returned:

<b>Extent</b>	<b>Value</b>
Leftbearing	The minimum left bearing of all characters in the string
Rightbearing	The maximum right bearing of all characters in the string
Width	The sum of all character widths in the string
Ascent	The maximum ascent of all characters in the string
Descent	The maximum descent of all characters in the string

When a font has no defined default character, undefined characters in the specified string receive zero character metric values.

## TEXT EXTENTS 16

Returns logical extents of a 16-bit character string in a given font.

**VAX FORMAT**    **X\$TEXT\_EXTENTS\_16**  
*(font\_ptr,string, num\_chars [,direction\_return]  
[,ascent\_return] [,descent\_return] [,overall\_return])*

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
font_ptr	record	x\$font_struct	read	reference
string	array	word	read	reference
num_chars	word	word <sup>1</sup>	read	reference
direction_return	longword	longword	write	reference
ascent_return	longword	longword	write	reference
descent_return	longword	longword	write	reference
overall_return	record	x\$char_struct	write	reference

<sup>1</sup>unsigned

**MIT C FORMAT**    **XTextExtents16**  
*(font\_ptr,string, num\_chars, direction\_return,  
ascent\_return, descent\_return, overall\_return)*

**argument  
information**

```
XTextExtents16(font_ptr, string, num_chars, direction_return,
               ascent_return, descent_return, overall_return)
XFontStruct *font_ptr;
XChar2b *string;
int num_chars;
int *direction_return;
int *ascent_return, *descent_return;
XCharStruct *overall_return;
```

**ARGUMENTS**

**font\_ptr**  
The address of a font data structure. TEXT EXTENTS uses the character information contained in the font data structure to determine extents. The address of the font data structure is returned by LOAD QUERY FONT or QUERY FONT.

**string**  
The character string whose logical extents are being queried.

***num\_chars***

The number of characters in the string whose extents are being queried.

***direction\_return***

The direction the string is painted on the screen. The **direction\_return** argument is the returned value of the direction element of the font data structure.

**VAX only**

This argument is optional.

***ascent\_return***

Maximum ascent of the font used to draw the string. The **ascent\_return** argument is the returned value of the font ascent element of the font data structure.

**VAX only**

This argument is optional.

***descent\_return***

Maximum descent of the font used to draw the string. The **descent\_return** argument is the returned value of the font descent element of the font data structure.

**VAX only**

This argument is optional.

***overall\_return***

The minimum left bearing, maximum right bearing, string width, maximum character ascent, and maximum character descent of the string.

For more information about the character structure, see Chapter 13.

**VAX only**

This argument is optional.

---

**DESCRIPTION**

TEXT EXTENTS 16 returns the logical extents of a specified 16-bit string of a given font. Unlike QUERY TEXT EXTENTS 16, which queries the server for the sizes of characters in the string, TEXT EXTENTS 16 uses the pointer returned by LOAD QUERY FONT to query the font data structure directly.

Given a 16-bit string, TEXT EXTENTS 16 returns the following information about extents:

- Direction the font is painted
- Ascent above the baseline used for determining line spacing
- Descent below the baseline used for determining line spacing
- Character extents



## Text Routines

### TEXT EXTENTS 16

The following string extents are returned:

Extent	Value
Leftbearing	The minimum left bearing of all characters in the string
Rightbearing	The maximum right bearing of all characters in the string
Width	The sum of all character widths in the string
Ascent	The maximum ascent of all characters in the string
Descent	The maximum descent of all characters in the string

When fonts are defined with linear indexing rather than 2-byte matrix indexing, the server interprets each 2-byte character as a 16-bit number with the first byte as most significant.

When a font has no defined default character, undefined characters in the specified string receive zero character metric values.

---

## TEXT WIDTH

Returns the length of a string composed of 8-bit characters, given the string and the font in which the string is to be written.

---

**VAX FORMAT**     *width\_return = X\$TEXT\_WIDTH*  
                          (*font\_ptr, string*)

**argument  
information**

Argument	Usage	Data Type	Access	Mechanism
width_return	longword	uns longword	write	value
font_ptr	record	x\$font_struct	read	reference
string	char_string	character string	read	descriptor

---

**MIT C FORMAT**     *width\_return = XTextWidth*  
                          (*font\_ptr, string, count*)

**argument  
information**

```
int XTextWidth(font_ptr, string, count)
    XFontStruct *font_ptr;
    char *string;
    int count;
```

---

**RETURNS**

***width\_return***

The length, in pixels, of the specified string, measured using the character information stored in the specified font.

---

**ARGUMENTS**

***font\_ptr***

The address of a font data structure. TEXT WIDTH uses the character information contained in the structure to calculate the length. The address of the structure is returned by LOAD QUERY FONT and QUERY FONT.

***string***

The character string to be measured for length. The string must consist of 8-bit characters.

***count (MIT C only)***

The character count of the named string.

## Text Routines

### TEXT WIDTH

---

#### DESCRIPTION

TEXT WIDTH returns the length of the specified string.

Length, measured in the direction of text painting, is the sum of all the characters in the string.

This information is stored in a font structure associated with the font used initially to create the string. The identifier of the structure is returned by FONT or QUERY FONT. For more information about the font data structure, see Chapter 13.

Typically, this routine is used to ensure that a proposed character string does not exceed some maximum size, as when you are putting a label in a box on the screen.

---

## TEXT WIDTH 16

Returns the length of a string composed of 16-bit characters, given the string and the font in which the string is to be written.

---

**VAX FORMAT**     *width\_return = X\$TEXT\_WIDTH\_16*  
                           (*font\_ptr, string16, count*)

**argument  
information**

---

Argument	Usage	Data Type	Access	Mechanism
width_return	longword	uns longword	write	value
font_ptr	record	x\$font_struct	read	reference
string16	array	word	read	reference
count	word	uns word	read	reference

---



---

**MIT C FORMAT**     *width\_return = XTextWidth16*  
                           (*font\_ptr, string16, count*)

**argument  
information**

```
int XTextWidth16(font_ptr, string16, count)
    XFontStruct *font_ptr;
    XChar2b *string16;
    int count;
```

---

**RETURNS**

***width\_return***

The length, in pixels, of the specified string, measured using the character information stored in the specified font.

---

**ARGUMENTS**

***font\_ptr***

The address of a font data structure. TEXT WIDTH 16 uses the character information contained in the font data structure to calculate the length. The address of the font data structure is returned by LOAD QUERY FONT and QUERY FONT.

***string16***

The character string to be measured for length. The string must consist of 16-bit characters.

***count***

The character count of the named string.

## Text Routines

### TEXT WIDTH 16

---

#### DESCRIPTION

TEXT WIDTH 16 returns the length of the specified string.

Length, measured in the direction of text painting, is the sum of the width of all the characters in the string.

This information is stored in a font structure associated with the font used initially to create the string. The identifier of the structure is returned by FONT or QUERY FONT. For more information about the font data structure, see Chapter 13.

Typically, this routine is used to ensure that a proposed character string does not exceed some maximum size.