

MicroVMS Workstation Graphics Programming Guide

Order Number: AI-GI10B-TN

May 1986

This document provides programming information about the MicroVMS Workstation graphics software. It describes the general concepts and specific routine calls which are used in writing application programs.

Revision/Update Information: This manual supersedes the *MicroVMS Workstation Graphics Programming Guide*, Version 2.0.

Software Version: MicroVMS Workstation Graphics Software
Version 3.0

**digital equipment corporation
maynard, massachusetts**

May 1986

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.

Copyright ©1986 by Digital Equipment Corporation

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

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	
DECUS	RSTS	
DECwriter	RSX	

digital

ZK-3164

**HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS**

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

* Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminister, Massachusetts 01473.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by T_EX, the typesetting system developed by Donald E. Knuth at Stanford University. T_EX is a trademark of the American Mathematical Society.

Contents

Preface	xxvii
New and Changed Features	xxxiii

PART I MicroVMS Workstation Graphics Concepts

Chapter 1 System Description

1.1	Overview	1-1
1.2	VAXstation Hardware	1-1
1.2.1	Processor	1-2
1.2.2	Monitor	1-2
1.2.3	Keyboard	1-3
1.2.4	Mouse	1-3
1.2.5	Tablet	1-3
1.2.6	Communications Board	1-4
1.2.7	Printer	1-4
1.3	Software	1-4
1.3.1	Graphics Routine Types	1-4
1.3.2	Human Interface	1-5
1.3.2.1	Terminal Emulation	1-6
1.3.2.2	Communication Tools	1-6
1.3.3	Windowing Feature	1-7
1.3.4	Graphics Capabilities	1-7

Chapter 2 Display Management Concepts

2.1	Overview	2-1
2.1.1	Summary	2-1
2.2	Coordinate Systems	2-3
2.2.1	Device-Independent Coordinate Systems	2-3
2.2.1.1	World Coordinates	2-4
2.2.1.2	Normalized Coordinates	2-5
2.2.2	Device-Dependent Coordinate Systems	2-6
2.2.2.1	Absolute Device Coordinates	2-6
2.2.2.2	Viewport-Relative Device Coordinates	2-7
2.3	Virtual Displays	2-8
2.4	Display Windows	2-9
2.5	Display Viewports	2-10
2.6	Display Window and Viewport Scaling	2-11
2.6.1	Distortion of Graphic Objects	2-12
2.7	Display Lists	2-13
2.8	Generic Encoding and UIS Metafiles	2-13

Chapter 3 Graphic Objects and Attributes

3.1	Overview	3-1
3.2	Summary	3-1
3.3	Text and Graphics Routines	3-2
3.4	Attributes	3-2
3.4.1	General Attributes	3-3
3.4.2	Text Attributes	3-3
3.4.3	Graphics Attributes	3-5
3.4.4	Window Attribute	3-6
3.5	Attribute Blocks	3-6
3.5.1	Attribute Block 0	3-6
3.6	Segments	3-7
3.7	Viewing Transformations	3-7
3.8	Two-Dimensional Geometric Transformations	3-7

Chapter 4 Color Concepts

4.1	Overview	4-1
4.2	Color Hardware Systems	4-1
4.3	Raster Graphics Concepts	4-1
4.3.1	Hardware Interpretation of Pixel Values	4-2
4.3.2	Color Representation Models	4-6
4.3.3	Color Palette	4-6
4.4	UIS Virtual Color Maps	4-7
4.4.1	Reserved Hardware Color Map Entries	4-9
4.5	UIS Color Map Segments	4-11
4.6	Shareable Virtual Color Maps	4-11
4.7	Miscellaneous UIS Color Concepts	4-11
4.7.1	Standard and Preferred Colors	4-11
4.7.2	Monochrome, Intensity, and Color Compatibility Features	4-12
4.7.3	Color Value Conversion	4-12
4.7.4	Set Colors and Realized Colors	4-13
4.7.5	Color Regeneration Characteristics	4-13

Chapter 5 Input Devices

5.1	Overview	5-1
5.1.1	VAXstation Input Devices	5-1
5.2	Pointers	5-2
5.2.1	Mouse	5-2
5.2.2	Tablet	5-3
5.3	Keyboards	5-4
5.3.1	Virtual Keyboards	5-4

PART II How to Program with MicroVMS Workstation Graphics

Chapter 6 Programming Considerations

6.1	Overview	6-1
6.2	Calling UIS Routines	6-1
6.2.1	Calling Sequences	6-2
6.2.1.1	Call Type	6-2
6.2.1.2	Routine Name	6-2
6.2.1.3	Argument List	6-2
6.3	Argument Characteristics	6-3
6.3.1	VMS Usage	6-3
6.3.2	Type	6-3
6.3.2.1	VAX Standard Data Types	6-3
6.3.3	Access	6-5
6.3.4	Mechanism	6-5
6.3.4.1	VAX FORTRAN Built-In Functions	6-7
6.4	UIS Constants	6-9
6.5	Condition Values Signaled	6-9
6.6	Additional Program Components	6-9
6.7	Notes to Programmers	6-10
6.7.1	VAX C Programmers	6-10
6.7.2	VAX PASCAL Programmers	6-11
6.7.3	VAX PL/I Programmers	6-12
6.8	Programming Examples	6-12
6.8.1	Structure of Programming Tutorial	6-12
6.9	Program Execution	6-13
6.9.1	Compiling Your Program	6-13
6.9.2	Linking the Object Module	6-14
6.9.3	Running the Executable Image	6-14

Chapter 7 Creating Basic Graphic Objects

7.1	Overview	7-1
7.2	Step 1—Creating a Virtual Display	7-1
7.2.1	Specifying Coordinate Values	7-2
7.2.2	Programming Options	7-2
7.2.3	Program Development	7-3
7.3	Step 2—Creating Graphics and Text	7-4
7.3.1	Graphics Drawing Operations	7-4
7.3.2	Programming Options	7-4
7.3.3	Program Development	7-6
7.4	Step 3—Creating a Display Window	7-6
7.4.1	Programming Options	7-7
7.4.2	Program Development	7-7
7.4.3	Calling UIS\$CIRCLE, UIS\$ELLIPSE, UIS\$PLOT, UIS\$TEXT, and UIS\$CREATE_WINDOW	7-8

Chapter 8 Display Windows and Viewports

8.1	Overview	8-1
8.2	Windowing Routines	8-1
8.3	Step 1—Creating Many Display Windows	8-2
8.3.1	Programming Options	8-2
8.3.2	Program Development	8-4
8.3.3	Calling UIS\$CREATE_WINDOW	8-5
8.4	Step 2—Deleting and Erasing Display Windows	8-7
8.4.1	Programming Options	8-7
8.4.2	Program Development	8-7
8.4.3	Calling UIS\$DELETE_WINDOW	8-9
8.5	Step 3—Manipulating Display Windows and Viewports	8-12
8.5.1	Programming Options	8-12
8.5.2	Program Development I	8-13
8.5.3	Calling UIS\$MOVE_WINDOW	8-15
8.5.4	Program Development II	8-18
8.5.5	Calling UIS\$POP_VIEWPORT and UIS\$PUSH_VIEWPORT	8-20
8.5.6	Program Development III	8-24
8.5.7	Requesting General Placement and No Border	8-25
8.5.8	Program Development IV	8-27
8.5.9	Calling UIS\$MOVE_AREA	8-29

8.6	World Coordinate Transformations	8-29
8.6.1	Programming Options	8-29
8.6.2	Program Development	8-29
8.6.3	Calling UIS\$CREATE_TRANSFORMATION	8-30

Chapter 9 General Attributes

9.1	Overview	9-1
9.2	Attributes—How to Use Them	9-1
9.2.1	Attribute Blocks	9-2
9.2.2	Modifying General Attributes	9-3
9.3	Structure of Graphic Objects	9-3
9.4	UIS Writing Modes	9-6
9.4.1	Using General Attributes	9-8
9.4.1.1	Programming Options	9-8
9.4.1.2	Program Development I	9-9
9.4.1.3	Calling UIS\$SET_BACKGROUND_INDEX, UIS\$SET_WRITING_INDEX, and UIS\$SET_WRITING_MODE	9-11
9.4.1.4	Program Development II	9-14
9.4.1.5	Using Device-Dependent Writing Modes	9-17

Chapter 10 Text Attributes

10.1	Overview	10-1
10.2	Structure of Text	10-1
10.2.1	Monospaced and Proportionally Spaced Fonts	10-2
10.2.2	Lines of Text	10-2
10.2.3	Character Strings	10-4
10.2.4	Character Cell	10-9
10.3	Using Text Attributes	10-21
10.3.1	Modifying Text Attributes	10-22
10.4	Programming Options	10-23
10.4.1	Program Development I	10-25
10.4.2	Calling UIS\$SET_FONT and UIS\$NEW_TEXT_LINE	10-27
10.4.3	Program Development II	10-29
10.4.4	Calling UIS\$SET_CHAR_SPACING	10-30
10.4.5	Program Development III	10-31
10.4.6	Calling UIS\$SET_POSITION and UIS\$SET_ALIGNED_POSITION	10-32
10.4.7	Program Development IV	10-33

10.4.8	Calling UIS\$SET_CHAR_SLANT	10-34
10.4.9	Program Development V	10-35
10.4.10	Calling UIS\$SET_TEXT_SLOPE	10-36
10.4.11	Program Development VI	10-36
10.4.12	Calling UIS\$SET_CHAR_ROTATION	10-37
10.4.13	Program Development VII	10-39
10.4.14	Calling UIS\$SET_CHAR_SIZE	10-41

Chapter 11 Graphics and Windowing Attributes

11.1	Overview	11-1
11.2	Using Graphics Attributes	11-1
11.2.1	Modifying Graphics and Windowing Attributes	11-1
11.2.2	Programming Options	11-2
11.2.2.1	Program Development I	11-4
11.2.2.2	Calling UIS\$SET_ARC_TYPE and Using Fill Patterns	11-5
11.2.2.3	Program Development II	11-8
11.2.2.4	Calling UIS\$SET_LINE_WIDTH	11-9
11.2.2.5	Program Development III	11-10
11.2.2.6	Calling UIS\$SET_LINE_WIDTH and UIS\$SET_LINE_STYLE	11-11
11.2.2.7	Program Development IV	11-11
11.2.2.8	Calling UIS\$SET_FONT and UIS\$SET_FILL_PATTERN	11-14
11.2.3	Using the Windowing Attribute	11-14
11.2.3.1	Programming Options	11-14
11.2.3.2	Program Development	11-14
11.2.3.3	Calling UIS\$SET_CLIP	11-17

Chapter 12 Inquiry Routines

12.1	Overview	12-1
12.2	Inquiry Routines—How to Use Them	12-1
12.2.1	Using Inquiry Routines	12-1
12.2.1.1	Programming Options	12-2
12.2.1.2	Program Development I	12-5
12.2.1.3	Invoking UIS\$GET_FONT_SIZE, UIS\$GET_DISPLAY_SIZE, and UIS\$GET_VIEWPORT_SIZE	12-7
12.2.1.4	Program Development II	12-8
12.2.1.5	Invoking UIS\$GET_ARC_TYPE, UIS\$GET_FILL_PATTERN, and UIS\$GET_FONT	12-10

Chapter 13 Display Lists and Segmentation

13.1	Overview	13-1
13.2	Display Lists	13-1
13.3	Segments	13-2
13.3.1	Identifiers and Object Types	13-3
13.3.2	Programming Options	13-5
13.3.3	Program Development I	13-7
13.3.3.1	Calling <code>UIS\$DISABLE_DISPLAY_LIST</code> and <code>UIS\$ENABLE_DISPLAY_LIST</code>	13-8
13.3.3.2	Program Development II	13-9
13.3.3.3	Calling <code>UIS\$GET_NEXT_OBJECT</code> , <code>UIS\$GET_OBJECT_ATTRIBUTES</code> , and <code>UIS\$GET_ROOT_SEGMENT</code>	13-13
13.3.3.4	Program Development III	13-15
13.3.3.5	Calling <code>UIS\$GET_PARENT_SEGMENT</code>	13-19
13.4	More About Segments	13-21
13.4.1	Programming Options	13-21
13.4.2	Program Development I	13-22
13.4.2.1	Calling <code>UIS\$SET_INSERTION_POSITION</code>	13-26
13.4.2.2	Program Development II	13-29
13.4.2.3	Calling <code>UIS\$BEGIN_SEGMENT</code> and <code>UIS\$END_SEGMENT</code>	13-31

Chapter 14 Geometric and Attribute Transformations

14.1	Overview	14-1
14.2	Geometric Transformations	14-1
14.2.1	Translating Graphic Objects	14-1
14.2.2	Scaling Graphic Objects	14-3
14.2.2.1	Uniformly Scaled Graphic Objects	14-6
14.2.2.2	Differentially Scaled Graphic Objects	14-7
14.2.3	Rotating Graphic Objects	14-8
14.2.4	Programming Options	14-10
14.2.5	Program Development I	14-10
14.2.6	Calling <code>UIS\$TRANSFORMATION_OBJECT</code>	14-12
14.2.7	Program Development II	14-12
14.2.8	Calling <code>UIS\$COPY_OBJECT</code>	14-15
14.3	Attribute Transformations	14-17
14.3.1	Programming Options	14-17
14.3.2	Program Development	14-17
14.3.3	Requesting Attribute Transformations	14-18

Chapter 15 Metafiles and Private Data

15.1	Overview	15-1
15.2	Display Lists and UIS Metafiles	15-1
15.2.1	Generic Encoding of Graphics and Attribute Routines	15-2
15.2.1.1	Normalized Coordinates	15-2
15.2.1.2	Interpreting the User Buffer	15-3
15.2.2	Creating UIS Metafiles	15-11
15.2.3	Structure of a UIS Metafile	15-12
15.2.4	Programming Options	15-14
15.2.5	Program Development I	15-14
15.2.5.1	Calling UIS\$EXTRACT_HEADER, UIS\$EXTRACT_REGION, and UIS\$EXTRACT_TRAILER	15-17
15.3	Display Lists and Private Data	15-19
15.3.1	Using Private Data	15-19
15.3.2	Programming Options	15-20
15.3.3	Program Development II	15-20
15.3.3.1	Calling UIS\$PRIVATE and UIS\$EXTRACT_PRIVATE	15-25

Chapter 16 Programming in Color

16.1	Overview	16-1
16.2	Color and Intensity Routines—How to Use Them	16-1
16.2.1	Step 1—Creating a Virtual Color Map	16-2
16.2.2	Step 2—Setting Virtual Color Map Attributes	16-2
16.2.3	Step 3—Setting Entries in the Virtual Color Map	16-3
16.2.4	Programming Options	16-3
16.2.5	Program Development I	16-4
16.2.6	Program Development II	16-6
16.2.6.1	Program Development III	16-7
16.3	Color Map Segments	16-9
16.3.1	Programming Options	16-10
16.3.2	Program Development	16-10
16.3.3	Calling UIS\$CREATE_COLOR_MAP_SEG	16-11
16.4	Color and Intensity Inquiry Routines	16-11
16.4.1	Programming Options	16-11
16.4.2	Program Development I	16-12
16.4.2.1	Calling UIS\$GET_COLORS, UIS\$GET_HW_COLOR_INFO, UIS\$GET_WRITING_INDEX	16-14
16.4.3	Program II—Creating an HSV Color Wheel	16-15

Chapter 17 Asynchronous System Trap Routines

17.1	Overview	17-1
17.1.1	Using AST Routines	17-1
17.1.2	AST-Enabling Routines	17-2
17.2	Using Keyboard and Pointer Devices	17-3
17.2.1	Using AST Routines with Virtual Keyboards	17-3
17.2.1.1	Step 1—Creating a Virtual Keyboard	17-3
17.2.1.2	Step 2—Binding the Virtual Keyboard to the Display Window	17-3
17.2.1.3	Step 3—Enabling Virtual Keyboard AST Routines	17-4
17.2.2	Programming Options	17-4
17.2.3	Program Development	17-5
17.2.4	Calling Keyboard Routines	17-7
17.2.5	Using AST Routines with Pointer Devices	17-8
17.2.5.1	Mouse	17-8
17.2.5.2	Tablet	17-8
17.2.5.3	Step 1—Create an AST Routine	17-9
17.2.5.4	Step 2—Enable the AST Routine	17-9
17.2.6	Programming Options	17-9
17.2.7	Program Development	17-10
17.2.8	Calling UIS\$SET_POINTER_AST and UIS\$SET_POINTER_PATTERN	17-12
17.3	Manipulating Display Windows and Viewports	17-13
17.3.1	Using AST Routines to Modify the Window Options Menu	17-14
17.3.1.1	Step 1—Create an AST Routine	17-14
17.3.1.2	Step 2—Enable the AST Routine	17-14
17.3.2	Programming Options	17-15
17.3.3	Program Development	17-16
17.3.4	Calling UIS\$SET_RESIZE_AST	17-20
17.3.5	Calling UIS\$SET_SHRINK_TO_ICON_AST	17-21
17.3.6	Calling UIS\$SET_CLOSE_AST	17-22

PART III UIS Routines

Chapter 18 UIS Routine Descriptions

18.1	Overview	18-1
18.1.1	Format Heading	18-3
18.1.2	Returns Heading	18-5
18.1.3	Arguments Heading	18-6
18.2	Functional Organization of UIS Routines	18-6
	UIS\$BEGIN_SEGMENT	18-9
	UIS\$CIRCLE	18-11
	UIS\$CLOSE_WINDOW	18-14
	UIS\$COPY_OBJECT	18-15
	UIS\$CREATE_COLOR_MAP	18-20
	UIS\$CREATE_COLOR_MAP_SEG	18-23
	UIS\$CREATE_DISPLAY	18-26
	UIS\$CREATE_KB	18-28
	UIS\$CREATE_TB	18-31
	UIS\$CREATE_TERMINAL	18-32
	UIS\$CREATE_TRANSFORMATION	18-34
	UIS\$CREATE_WINDOW	18-37
	UIS\$DELETE_COLOR_MAP	18-46
	UIS\$DELETE_COLOR_MAP_SEG	18-47
	UIS\$DELETE_DISPLAY	18-48
	UIS\$DELETE_KB	18-49
	UIS\$DELETE_OBJECT	18-50
	UIS\$DELETE_PRIVATE	18-51
	UIS\$DELETE_TB	18-52
	UIS\$DELETE_TRANSFORMATION	18-53
	UIS\$DELETE_WINDOW	18-54
	UIS\$DISABLE_DISPLAY_LIST	18-55
	UIS\$DISABLE_KB	18-58
	UIS\$DISABLE_TB	18-59
	UIS\$DISABLE_VIEWPORT_KB	18-60
	UIS\$ELLIPSE	18-61
	UIS\$ENABLE_DISPLAY_LIST	18-65
	UIS\$ENABLE_KB	18-68
	UIS\$ENABLE_TB	18-70
	UIS\$ENABLE_VIEWPORT_KB	18-71
	UIS\$END_SEGMENT	18-72
	UIS\$ERASE	18-73
	UIS\$EXECUTE	18-75
	UIS\$EXECUTE_DISPLAY	18-77

UIS\$EXPAND_ICON	18-78
UIS\$EXTRACT_HEADER	18-81
UIS\$EXTRACT_OBJECT	18-83
UIS\$EXTRACT_PRIVATE	18-85
UIS\$EXTRACT_REGION	18-88
UIS\$EXTRACT_TRAILER	18-91
UIS\$FIND_PRIMITIVE	18-93
UIS\$FIND_SEGMENT	18-95
UIS\$GET_ABS_POINTER_POS	18-97
UIS\$GET_ALIGNED_POSITION	18-98
UIS\$GET_ARC_TYPE	18-100
UIS\$GET_BACKGROUND_INDEX	18-102
UIS\$GET_BUTTONS	18-103
UIS\$GET_CHAR_ROTATION	18-105
UIS\$GET_CHAR_SIZE	18-106
UIS\$GET_CHAR_SLANT	18-108
UIS\$GET_CHAR_SPACING	18-110
UIS\$GET_CLIP	18-112
UIS\$GET_COLOR	18-115
UIS\$GET_COLORS	18-118
UIS\$GET_CURRENT_OBJECT	18-121
UIS\$GET_DISPLAY_SIZE	18-123
UIS\$GET_FILL_PATTERN	18-126
UIS\$GET_FONT	18-129
UIS\$GET_FONT_ATTRIBUTES	18-131
UIS\$GET_FONT_SIZE	18-135
UIS\$GET_HW_COLOR_INFO	18-137
UIS\$GET_INTENSITIES	18-141
UIS\$GET_INTENSITY	18-144
UIS\$GET_KB_ATTRIBUTES	18-146
UIS\$GET_LINE_STYLE	18-148
UIS\$GET_LINE_WIDTH	18-150
UIS\$GET_NEXT_OBJECT	18-153
UIS\$GET_OBJECT_ATTRIBUTES	18-155
UIS\$GET_PARENT_SEGMENT	18-158
UIS\$GET_POINTER_POSITION	18-160
UIS\$GET_POSITION	18-162
UIS\$GET_PREVIOUS_OBJECT	18-164
UIS\$GET_ROOT_SEGMENT	18-167
UIS\$GET_TB_INFO	18-169
UIS\$GET_TB_POSITION	18-172
UIS\$GET_TEXT_FORMATTING	18-173
UIS\$GET_TEXT_MARGINS	18-175
UIS\$GET_TEXT_PATH	18-177

UIS\$GET_TEXT_SLOPE	18-179
UIS\$GET_VCM_ID	18-181
UIS\$GET_VIEWPORT_ICON	18-182
UIS\$GET_VIEWPORT_POSITION	18-184
UIS\$GET_VIEWPORT_SIZE	18-186
UIS\$GET_VISIBILITY	18-188
UIS\$GET_WINDOW_ATTRIBUTES	18-190
UIS\$GET_WINDOW_SIZE	18-191
UIS\$GET_WRITING_INDEX	18-192
UIS\$GET_WRITING_MODE	18-194
UIS\$GET_WS_COLOR	18-195
UIS\$GET_WS_INTENSITY	18-198
UIS\$HLS_TO_RGB	18-200
UIS\$HSV_TO_RGB	18-202
UIS\$IMAGE	18-204
UIS\$INSERT_OBJECT	18-209
UIS\$LINE	18-210
UIS\$LINE_ARRAY	18-213
UIS\$MEASURE_TEXT	18-215
UIS\$MOVE_AREA	18-221
UIS\$MOVE_VIEWPORT	18-224
UIS\$MOVE_WINDOW	18-226
UIS\$NEW_TEXT_LINE	18-228
UIS\$PLOT	18-229
UIS\$PLOT_ARRAY	18-232
UIS\$POP_VIEWPORT	18-234
UIS\$PRESENT	18-236
UIS\$PRIVATE	18-237
UIS\$PUSH_VIEWPORT	18-239
UIS\$READ_CHAR	18-241
UIS\$RESIZE_WINDOW	18-243
UIS\$RESTORE_CMS_COLORS	18-246
UIS\$RGB_TO_HLS	18-247
UIS\$RGB_TO_HSV	18-249
UIS\$SET_ADDOPT_AST	18-251
UIS\$SET_ALIGNED_POSITION	18-253
UIS\$SET_ARC_TYPE	18-255
UIS\$SET_BACKGROUND_INDEX	18-258
UIS\$SET_BUTTON_AST	18-260
UIS\$SET_CHAR_ROTATION	18-264
UIS\$SET_CHAR_SIZE	18-267
UIS\$SET_CHAR_SLANT	18-271
UIS\$SET_CHAR_SPACING	18-273
UIS\$SET_CLIP	18-278

UIS\$SET_CLOSE_AST	18-281
UIS\$SET_COLOR	18-283
UIS\$SET_COLORS	18-286
UIS\$SET_EXPAND_ICON_AST	18-289
UIS\$SET_FILL_PATTERN	18-291
UIS\$SET_FONT	18-295
UIS\$SET_GAIN_KB_AST	18-297
UIS\$SET_INSERTION_POSITION	18-299
UIS\$SET_INTENSITIES	18-302
UIS\$SET_INTENSITY	18-304
UIS\$SET_KB_AST	18-306
UIS\$SET_KB_ATTRIBUTES	18-308
UIS\$SET_KB_COMPOSE2	18-311
UIS\$SET_KB_COMPOSE3	18-313
UIS\$SET_KB_KEYTABLE	18-315
UIS\$SET_LINE_STYLE	18-317
UIS\$SET_LINE_WIDTH	18-320
UIS\$SET_LOSE_KB_AST	18-324
UIS\$SET_MOVE_INFO_AST	18-326
UIS\$SET_POINTER_AST	18-328
UIS\$SET_POINTER_PATTERN	18-332
UIS\$SET_POINTER_POSITION	18-335
UIS\$SET_POSITION	18-337
UIS\$SET_RESIZE_AST	18-339
UIS\$SET_SHRINK_TO_ICON_AST	18-344
UIS\$SET_TB_AST	18-346
UIS\$SET_TEXT_FORMATTING	18-349
UIS\$SET_TEXT_MARGINS	18-353
UIS\$SET_TEXT_PATH	18-355
UIS\$SET_TEXT_SLOPE	18-358
UIS\$SET_WRITING_INDEX	18-361
UIS\$SET_WRITING_MODE	18-363
UIS\$SHRINK_TO_ICON	18-365
UIS\$SOUND_BELL	18-369
UIS\$SOUND_CLICK	18-370
UIS\$TEST_KB	18-371
UIS\$TEXT	18-372
UIS\$TRANSFORM_OBJECT	18-376

PART IV UIS Device Coordinate (UISDC) Routines

Chapter 19 UIS Device Coordinate Graphics Routines

19.1	Overview	19-1
19.2	UISDC Routines—How to Use Them	19-1
	UISDC\$ALLOCATE_DOP	19-3
	UISDC\$CIRCLE	19-5
	UISDC\$ELLIPSE	19-7
	UISDC\$ERASE	19-10
	UISDC\$EXECUTE_DOP_ASYNC	19-11
	UISDC\$EXECUTE_DOP_SYNC	19-13
	UISDC\$GET_ALIGNED_POSITION	19-14
	UISDC\$GET_CHAR_SIZE	19-16
	UISDC\$GET_CLIP	19-18
	UISDC\$GET_POINTER_POSITION	19-20
	UISDC\$GET_POSITION	19-22
	UISDC\$GET_TEXT_MARGINS	19-23
	UISDC\$GET_VISIBILITY	19-25
	UISDC\$IMAGE	19-27
	UISDC\$LINE	19-31
	UISDC\$LINE_ARRAY	19-33
	UISDC\$LOAD_BITMAP	19-35
	UISDC\$MEASURE_TEXT	19-37
	UISDC\$MOVE_AREA	19-39
	UISDC\$NEW_TEXT_LINE	19-41
	UISDC\$PLOT	19-42
	UISDC\$PLOT_ARRAY	19-44
	UISDC\$QUEUE_DOP	19-46
	UISDC\$READ_IMAGE	19-47
	UISDC\$SET_ALIGNED_POSITION	19-50
	UISDC\$SET_BUTTON_AST	19-52
	UISDC\$SET_CHAR_SIZE	19-54
	UISDC\$SET_CLIP	19-56
	UISDC\$SET_POINTER_AST	19-58
	UISDC\$SET_POINTER_PATTERN	19-61
	UISDC\$SET_POINTER_POSITION	19-64
	UISDC\$SET_POSITION	19-65
	UISDC\$SET_TEXT_MARGINS	19-66
	UISDC\$TEXT	19-68

A Summary of UIS Calling Sequences

A.1	UIS Calling Sequences	A-1
-----	---------------------------------	-----

B Summary of UISDC Calling Sequences

B.1	UISDC Calling Sequences	B-1
-----	-----------------------------------	-----

C UIS Fonts

C.1	Overview	C-1
C.2	UIS Multinational Character Set Fonts	C-1
C.2.1	UIS Multinational Character Set Font Specifications	C-5
C.3	UIS Technical Character Set Fonts	C-10
C.3.1	UIS Technical Character Set Font Specifications	C-14

D UIS Fill Patterns

E Error Messages

F Obsolete Routines

Glossary

Index

Figures

1-1	Typical MicroVMS Workstation Hardware	1-2
2-1	Virtual Display, Display Window, and Display Viewport	2-3
2-2	World Coordinate System and Virtual Display	2-5
2-3	Absolute Device Coordinates	2-7
2-4	Mapping a Display Window to a Display Viewport	2-8
2-5	Display Window in a Virtual Display	2-10
2-6	Displaying a Graphic Object	2-11
2-7	Display List Extraction	2-14
4-1	Bitplane Configuration in Single- and Multiplane Systems	4-2
4-2	Direct Color Values	4-3
4-3	Hardware Color Map	4-4

4-4	Mapped Color Values in Four-Plane System . . .	4-5
4-5	RGB and Intensity Color Values as Hardware Color Map Entries	4-6
4-6	Swapping Virtual Color Maps	4-8
4-7	Reserved Hardware Color Map Entries in a 4-Plane Color System	4-10
6-1	Passing Arguments	6-8
7-1	Mapping a Bitmap to a Raster	7-5
7-2	Display Viewport and Graphic Objects	7-8
8-1	Aspect Ratios of the Display Window and Display Viewport	8-3
8-2	Four Display Viewports	8-6
8-3	Objects Within Different Windows	8-10
8-4	Display Window Deletion	8-11
8-5	Before Panning the Virtual Display	8-15
8-6	Panning the Virtual Display	8-17
8-7	Occluding a Display Viewport	8-21
8-8	Popping a Display Viewport	8-22
8-9	Pushing a Display Viewport	8-23
8-10	General Placement and No Border	8-26
8-11	Moving Graphic Objects Within the Virtual Display	8-28
8-12	World Coordinate Transformations	8-31
9-1	Structure of Graphic Objects	9-5
9-2	UIS Device-Independent Writing Modes	9-12
9-3	Bit Set Mode	9-18
9-4	Bit Clear Mode	9-19
9-5	Bit Set Negate Mode	9-20
9-6	Bit Clear Negate Mode	9-21
9-7	Copy Mode	9-22
9-8	Copy Negate Mode	9-23
10-1	Character Cell	10-2
10-2	Monospaced and Proportionally Spaced Characters	10-3
10-3	Text Path	10-3
10-4	Text Slope	10-5
10-5	Character Spacing	10-7
10-6	Simple Character Rotation	10-10
10-7	Character Rotation with Slope Manipulation	10-11
10-8	Text Path Manipulation Without Character Rotation	10-13

10-9	Character Slanting	10-18
10-10	Character Slanting and Rotation with Slope Manipulation	10-19
10-11	Character Scaling	10-21
10-12	UIS Fonts	10-28
10-13	Character and Line Spacing	10-31
10-14	Baseline and Top of Character Cell	10-33
10-15	Character Slanting	10-34
10-16	Manipulating the Text Baseline	10-36
10-17	Character Rotation Without Slanting	10-38
10-18	Character Rotation with Slanting	10-39
10-19	Manipulating Character Size	10-41
11-1	Closing an Arc	11-6
11-2	Filling a Closed Arc	11-7
11-3	Line Width	11-9
11-4	Modifying Line Width and Style	11-11
11-5	Vertical Bar Graph	11-15
11-6	Clipping rectangles	11-17
12-1	Centering Text	12-7
12-2	Pie Graph	12-11
13-1	Binary Encoded Instruction	13-2
13-2	Nested Segments	13-3
13-3	Disabling a Display List	13-8
13-4	After Display List Execution	13-9
13-5	Tree Diagram—Program WALK	13-10
13-6	Display List Elements	13-13
13-7	Contents of the Display List	13-14
13-8	Traversing Upward Along the Segment Path	13-19
13-9	Searching Downward Through a Segment	13-19
13-10	Contents of the Display List Drawn in the Virtual Display	13-20
13-11	Before Display List Modification	13-27
13-12	Executing the Modified Display List	13-28
13-13	Verifying the Contents of the Display List	13-29
13-14	Text Output During Execution	13-31
13-15	Final Text Output	13-32
14-1	Translating a Graphic Object	14-2
14-2	Simple Scaling	14-4
14-3	Complex Scaling	14-5
14-4	Uniformly Scaling a Graphic Object	14-6
14-5	Differentially Scaling a Graphic Object	14-7

14-6	Simple Rotation of a Graphic Object	14-9
14-7	Complex Rotation of a Rectangle	14-13
14-8	Complex Rotation of a Triangle	14-16
14-9	Modifying Attributes with a Transformation . . .	14-19
14-10	Modifying Attributes with a Copy	14-20
15-1	Binary Encoded Instruction	15-2
15-2	Extended Binary Encoded Instruction	15-2
15-3	Format of Attribute-Related Argument	15-5
15-4	Format of Graphics- and Text-Related Argument	15-5
15-5	Structure of UIS Metafile	15-13
15-6	Original Objects Drawn in the Virtual Display	15-18
15-7	After Buffer Execution	15-19
15-8	Private Data	15-25
15-9	Verifying the Contents of the Temporary Array and User Buffer	15-26
15-10	Hot Air Balloon	15-27
16-1	Different Types of Information Returned from Inquiry Routines	16-15
17-1	Writing Characters to a Display Viewport	17-8
17-2	Default Pointer Pattern	17-12
17-3	New Pointer Pattern	17-13
17-4	Unresized Window and Viewport	17-20
17-5	Resized Window and Viewport	17-21
17-6	Icon	17-21
18-1	Functional Categories of UIS Routines	18-7
C-1	Font 1	C-1
C-2	Font 2	C-1
C-3	Font 3	C-2
C-4	Font 4	C-2
C-5	Font 5	C-2
C-6	Font 6	C-3
C-7	Font 7	C-3
C-8	Font 8	C-3
C-9	Font 9	C-4
C-10	Font 10	C-4
C-11	Font 11	C-4
C-12	Font 12	C-5
C-13	Font 13	C-5
C-14	Font 14	C-5

C-15	Font 15	C-10
C-16	Font 16	C-11
C-17	Font 17	C-11
C-18	Font 18	C-11
C-19	Font 19	C-11
C-20	Font 20	C-12
C-21	Font 21	C-12
C-22	Font 22	C-12
C-23	Font 23	C-13
C-24	Font 24	C-13
C-25	Font 25	C-13
C-26	Font 26	C-14
D-1	PATT\$C_VERT1_1 and PATT\$C_VERT1_3 ..	D-1
D-2	PATT\$C_VERT2_2 and PATT\$C_VERT3_1 ..	D-2
D-3	PATT\$C_VERT1_7 and PATT\$C_VERT2_6 ..	D-2
D-4	PATT\$C_VERT4_4 and PATT\$C_VERT6_2 ..	D-2
D-5	PATT\$C_HORIZ1_1 and PATT\$C_HORIZ1_3	D-3
D-6	PATT\$C_HORIZ2_2 and PATT\$C_HORIZ3_1	D-3
D-7	PATT\$C_HORIZ1_7 and PATT\$C_HORIZ2_6	D-3
D-8	PATT\$C_HORIZ4_4 and PATT\$C_HORIZ6_2	D-4
D-9	PATT\$C_GRID4 and PATT\$C_GRID8	D-4
D-10	PATT\$C_UPDIAG1_3 and PATT\$C_UPDIAG2_2	D-4
D-11	PATT\$C_UPDIAG3_1 and PATT\$C_UPDIAG1_7	D-5
D-12	PATT\$C_UPDIAG2_6 and PATT\$C_UPDIAG4_4	D-5
D-13	PATT\$C_UPDIAG6_2 and PATT\$C_DOWNDIAG1_3	D-5
D-14	PATT\$C_DOWNDIAG2_2 and PATT\$C_DOWNDIAG3_1	D-6
D-15	PATT\$C_DOWNDIAG1_7 and PATT\$C_DOWNDIAG2_6	D-6
D-16	PATT\$C_DOWNDIAG4_4 and PATT\$C_DOWNDIAG6_2	D-6
D-17	PATT\$C_BRICK_HORIZ and PATT\$C_BRICK_VERT	D-7
D-18	PATT\$C_BRICK_DOWNDIAG and PATT\$C_BRICK_UPDIAG	D-7

D-19	PATT\$C_GREY4_16D and PATT\$C_GREY12_16D	D-7
D-20	PATT\$C_BASKET_WEAVE and PATT\$C_SCALE_DOWN	D-8
D-21	PATT\$C_SCALE_UP and PATT\$C_SCALE_RIGHT	D-8
D-22	PATT\$C_SCALE_LEFT and PATT\$C_GREY1_16	D-8
D-23	PATT\$C_GREY2_16 and PATT\$C_GREY3_16	D-9
D-24	PATT\$C_GREY4_16 and PATT\$C_GREY5_16	D-9
D-25	PATT\$C_GREY6_16 and PATT\$C_GREY7_16	D-9
D-26	PATT\$C_GREY8_16 and PATT\$C_GREY9_16	D-10
D-27	PATT\$C_GREY10_16 and PATT\$C_GREY11_16	D-10
D-28	PATT\$C_GREY12_16 and PATT\$C_GREY13_16	D-10
D-29	PATT\$C_GREY14_16 and PATT\$C_GREY15_16	D-11

Tables

4-1	Hardware Color Map Characteristics	4-4
4-2	Color Palette	4-7
6-1	VAX Standard Data Types	6-4
6-2	Entry Point and Symbol Definition Files	6-10
7-1	Types of Coordinates	7-2
8-1	UIS Windowing Routines	8-2
9-1	Attribute Block 0	9-2
9-2	Default Settings of General Attributes	9-3
9-3	UIS Writing Modes	9-6
10-1	Default Settings of Text Attributes in Attribute Block 0	10-22
11-1	Default Settings of Graphics and Windowing Attributes	11-2
12-1	Inquiry Routines	12-2
15-1	Generic Encoding Symbols and Opcodes	15-3
15-2	Arguments of Binary Encoded Instructions	15-6
15-3	Structure of UIS Metafiles	15-11

16-1	Color and Intensity Routines	16-3
16-3	Color and Intensity Inquiry Routines	16-11
17-1	AST-Enabling Routines	17-2
17-2	Connecting Physical Keyboards and Virtual Keyboards	17-4
17-3	Disconnecting Physical Keyboards and Virtual Keyboards	17-4
18-1	Main Headings in the Routine Template	18-1
18-2	General Rules of Syntax	18-4
A-1	Summary of UIS Calling Sequences	A-1
B-1	Summary of UISDC Calling Sequences	B-1
C-1	Font 1— DTABER0003WK00PG0001UZZZZ02A000 . . .	C-6
C-2	Font 2— DTABER0103WK00GG0001UZZZZ02A000 . . .	C-6
C-3	Font 3— DTABER0M03CK00GG0001UZZZZ02A000 . . .	C-6
C-4	Font 4— DTABER0R03WK00GG0001UZZZZ02A000 . . .	C-7
C-5	Font 5— DTABER0R07SK00GG0001UZZZZ02A000 . . .	C-7
C-6	Font 6— DTERMING03CK00PG0001UZZZZ02A000 . . .	C-7
C-7	Font 7— DTERMINM06OK00PG0001UZZZZ02A000 . . .	C-8
C-8	Font 8— DTABER0003WK00GG0001UZZZZ02A000 . . .	C-8
C-9	Font 9— DTABER0G03CK00GG0001UZZZZ02A000 . . .	C-8
C-10	Font 10— DTABER0I03WK00PG0001UZZZZ02A000 . . .	C-9
C-11	Font 11— DTABER0M06OK00GG0001UZZZZ02A000 . . .	C-9
C-12	Font 12— DTABER0R03WK00PG0001UZZZZ02A000 . . .	C-9
C-13	Font 13— DTABER0R07SK00PG0001UZZZZ02A000 . . .	C-10
C-14	Font 14— DTERMINM03CK00PG001UZZZZ02A000 . . .	C-10
C-15	Font 15— DVWSVT0G03CK00GG0001QZZZZ02A000 . . .	C-14
C-16	Font 16— DVWSVT0G03CK00PG0001QZZZZ02A000 . . .	C-15

C-17	Font 17— DVWSVT0I03WK00GG0001QZZZZ02A000 . .	C-15
C-18	Font 18— DVWSVT0I03WK00PG0001QZZZZ02A000 . .	C-15
C-19	Font 19— DVWSVT0N03CK00GG0001QZZZZ02A000 . .	C-16
C-20	Font 20— DVWSVT0N03CK00PG0001QZZZZ02A000 . .	C-16
C-21	Font 21— DVWSVT0N06OK00GG0001QZZZZ02A000 . .	C-16
C-22	Font 22— DVWSVT0N06OK00PG0001QZZZZ02A000 . .	C-17
C-23	Font 23— DVWSVT0R03WK00GG0001QZZZZ02A000 . .	C-17
C-24	Font 24— DVWSVT0R03WK00PG0001QZZZZ02A000 . .	C-17
C-25	Font 25— DVWSVT0R07SK00GG0001QZZZZ02A000 . .	C-18
C-26	Font 26— DVWSVT0R07SK00GG0001QZZZZ02A000 . .	C-18

Preface

This programming guide describes the MicroVMS workstation graphics software. It contains general information about basic MicroVMS graphics concepts, a tutorial for learning to program with MicroVMS graphics, and complete descriptions and reference information about the system routines for all callable functions.

Intended Audience

This guide is intended for general users and programmers who want to learn the concepts and use appropriate routines in graphics application programs.

Structure of This Document

This guide is divided into four major sections, MicroVMS Workstation Graphics Concepts, How to Program with MicroVMS Workstation Graphics, UIS Routine Descriptions, and UIS Device Coordinate (UISDC) Routines. These sections are briefly described in the following paragraphs.

Part I — MicroVMS Workstation Graphics Concepts

This section contains five chapters which provide a general overview of the basic concepts of MicroVMS workstation graphics.

- Chapter 1 — System Description
This chapter briefly describes the hardware, software, and options that are parts of the MicroVMS workstation system.
- Chapter 2 — Display Management Concepts
This chapter discusses the concepts of world coordinates, device coordinates, virtual displays, windows, viewports, window and viewport scaling, and distortion of graphic objects.
- Chapter 3 — Graphic Objects and Attributes
This chapter describes and shows the relationship between graphics routines, attribute blocks, text attributes, graphics attributes, and segments.

xxviii Preface

- Chapter 4 — Color Concepts
This chapter discusses the various color and intensity environments supported by the VAXstation color systems.
- Chapter 5 — Input Devices
This chapter shows how the workstation input devices relate to the workstation graphics system.

Part II — How to Program with MicroVMS Workstation Graphics

This section contains step-by-step tutorial information about writing application programs using MicroVMS graphics. Practical programming examples are provided throughout this section. It is divided according to routine functions into the following chapters:

- Chapter 6 — Programming Considerations
This chapter describes the programming interface and topics relating to program execution.
- Chapter 7 — Creating Basic Graphic Objects
This chapter describes the underlying structures and shows how to create graphic objects.
- Chapter 8 — Display Windows and Viewports
This chapter shows how to create and manipulate display windows and display viewports.
- Chapter 9 — General Attributes
This chapter describes writing modes, display background and foreground, and the writing index.
- Chapter 10 — Text Attributes
This chapter describes how attributes may be used to enhance and modify text.
- Chapter 11 — Graphics Attributes
This chapter describes how attributes may be used to enhance and modify the appearance of graphic objects.
- Chapter 12 — Inquiry Routines
This chapter discusses how information can be returned to the application program.
- Chapter 13 — Display Lists and Segmentation
This chapter describes how to create and manipulate display lists and segments.

- Chapter 14 — Geometric and Attribute Transformations

This chapter describes the various ways graphic objects and components of graphic objects can be manipulated with the respect to the coordinate space.

- Chapter 15 — Metafiles and Private Data

This chapter discusses how to extract the contents of a display list and store the data in a buffer or external file. There is additional information about how to associate private data with a graphics display.

- Chapter 16 — Programming in Color

The chapter describes how to create and display graphic objects in color.

- Chapter 17 — Asynchronous System Trap Routines

This chapter discusses how to make use of program-related events to increase the interactive nature of your applications.

Part III — UIS Routine Descriptions

This section contains reference material about the device-independent MicroVMS workstation graphics routines.

- Chapter 18 — UIS Routines Descriptions
- UIS Routine Descriptions

Part IV — UIS Device Coordinate (UISDC) Routines

This section contains reference material about device-dependent MicroVMS workstation graphics routines.

- Chapter 19 — UIS Device Coordinate Graphics Routines
- UISDC Routines

Appendix A — Summary of UIS Calling Sequences

Appendix B — Summary of UISDC Calling Sequences

Appendix C — UIS Fonts

Appendix D — UIS Fill Patterns

Appendix E — Error Messages

Appendix F — Obsolete Routines

Glossary

NOTE: For documentation on VMS data types, see Appendix A of the *MicroVMS Workstation Version 3.0 Release Notes*.

How To Use This Guide

This guide is designed so that it can be used in two different ways:

- It can be used as a learning tool by general users and programmers new to graphics software and MicroVMS workstation graphics.
- It can be used as a reference tool by programmers already familiar with graphics software in general and/or MicroVMS workstation graphics.

Inexperienced User

If you are unfamiliar with the MicroVMS workstation graphics system, you should begin by reading Part I of this guide. It gives you an overview of the graphics concepts discussed in subsequent sections of the book.

The programming tutorial in Part II provides a step-by-step approach for learning how to write applications that take advantage of the graphics capabilities of the MicroVMS workstation.

Part III provides you with reference information about all of the UIS routines used in MicroVMS workstation graphics. It is easier to use after you have read Part II of this guide.

Part IV contains appendices that provide reference material about UISDC graphics routines and error messages.

Experienced User

Once you have become familiar with MicroVMS workstation graphics, you will seldom need to refer to Part I of this guide, except when reviewing basic concepts.

Refer to Part II for examples and suggestions on the proper use of MicroVMS workstation graphics routines.

Part III is an alphabetically arranged reference section that you can use to get detailed descriptions of MicroVMS workstation graphics routines. Before using this section, you should already be familiar with Parts I and II of this guide.

Part IV contains appendices that provide reference material about UISDC graphics routines and error messages.

Associated Documents

The following MicroVMS manuals are related to this guide:

- *MicroVMS Workstation User's Guide*
- *MicroVMS Workstation Video Device Driver Manual*
- *MicroVMS Workstation Guide to Printing Graphics*
- *MicroVMS User's Manual*
- *MicroVMS User's Primer*
- *MicroVMS Programmer's Manual*
- *MicroVMS FORTRAN Programmer's Primer*
- *MicroVMS Programming Pocket Reference*
- *Installing or Upgrading MicroVMS From Diskettes*
- *Installing or Upgrading MicroVMS From a Tape Cartridge*

Conventions Used in This Document

This manual uses the following conventions:

Convention	Meaning
<code>RET</code>	A symbol with a one- to six-character abbreviation indicates that you press a key on the terminal, for example, <code>RET</code> .
<code>CTRL/x</code>	The phrase CTRL/x indicates that you must press the key labeled CTRL while you simultaneously press another key, for example, CTRL/C, CTRL/Y, CTRL/O.
\$ SHOW TIME 05-JUN-1986 11:55:22	Command examples show all output lines or prompting characters that the system prints or displays in black letters. All user-entered commands are shown in red letters.
\$ TYPE MYFILE.DAT . . .	Vertical series of periods, or ellipsis, mean either that not all the data that the system would display in response to the particular command is shown or that not all the data a user would enter is shown.
file-spec,...	Horizontal ellipsis indicates that additional parameters, values, or information can be entered.

Convention	Meaning
[logical-name]	Square brackets indicate that the enclosed item is optional. (Square brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

New and Changed Features

The following sections describes changes to the programming interface since UIS Version 2.0.

New UIS Routines

The following UIS routines were added.

Function	Routine
AST-enabling	UIS\$SET_ADDOPT_AST
	UIS\$SET_EXPAND_ICON_AST
	UIS\$SET_TB_AST
	UIS\$SET_SHRINK_TO_ICON_AST
Color	UIS\$CREATE_COLOR_MAP
	UIS\$CREATE_COLOR_MAP_SEG
	UIS\$DELETE_COLOR_MAP
	UIS\$DELETE_COLOR_MAP_SEG
	UIS\$GET_COLORS
	UIS\$GET_HW_COLOR_INFO
	UIS\$GET_INTENSITIES
	UIS\$GET_VCM_ID
	UIS\$HLS_TO_RGB
	UIS\$HSV_TO_RGB
	UIS\$RESTORE_CMS_COLORS
	UIS\$RGB_TO_HLS
	UIS\$RGB_TO_HSV
UIS\$SET_INTENSITIES	

xxxiv New and Changed Features

Function	Routine
Display list	UIS\$COPY_OBJECT
	UIS\$DELETE_OBJECT
	UIS\$DELETE_PRIVATE
	UIS\$EXECUTE
	UIS\$EXECUTE_DISPLAY
	UIS\$EXTRACT_HEADER
	UIS\$EXTRACT_OBJECT
	UIS\$EXTRACT_PRIVATE
	UIS\$EXTRACT_REGION
	UIS\$EXTRACT_TRAILER
	UIS\$FIND_PRIMITIVE
	UIS\$FIND_SEGMENT
	UIS\$GET_CURRENT_OBJECT
	UIS\$GET_NEXT_OBJECT
	UIS\$GET_OBJECT_ATTRIBUTES
	UIS\$GET_PARENT_SEGMENT
	UIS\$GET_PREVIOUS_OBJECT
	UIS\$GET_ROOT_SEGMENT
	UIS\$INSERT_OBJECT
	UIS\$PRIVATE
	UIS\$SET_INSERTION_POSITION
UIS\$TRANSFORM_OBJECT	
Graphics	UIS\$LINE
	UIS\$LINE_ARRAY
Keyboard and pointer	UIS\$CREATE_TB
	UIS\$DELETE_TB
	UIS\$DISABLE_TB
	UIS\$ENABLE_TB
	UIS\$GET_TB_INFO
	UIS\$GET_TB_POSITION

Function	Routine
Text	UIS\$GET_CHAR_ROTATION
	UIS\$GET_CHAR_SIZE
	UIS\$GET_CHAR_SLANT
	UIS\$GET_FONT_ATTRIBUTES
	UIS\$GET_TEXT_FORMATTING
	UIS\$GET_TEXT_MARGINS
	UIS\$GET_TEXT_PATH
	UIS\$GET_TEXT_SLOPE
	UIS\$SET_CHAR_ROTATION
	UIS\$SET_CHAR_SIZE
	UIS\$SET_CHAR_SLANT
	UIS\$SET_TEXT_FORMATTING
	UIS\$SET_TEXT_MARGINS
	UIS\$SET_TEXT_PATH
UIS\$SET_TEXT_SLOPE	
Windowing	UIS\$EXPAND_ICON
	UIS\$GET_VIEWPORT_ICON
	UIS\$GET_WINDOW_SIZE
	UIS\$SHRINK_TO_ICON

New UISDC Routines

The following UISDC routines are new for Version 3.0.

- UISDC\$ALLOCATE_DOP
- UISDC\$EXECUTE_DOP_ASYNCH
- UISDC\$EXECUTE_DOP_SYNCH
- UISDC\$GET_CHAR_SIZE
- UISDC\$GET_TEXT_MARGINS
- UISDC\$LINE
- UISDC\$LINE_ARRAY
- UISDC\$LOAD_BITMAP
- UISDC\$QUEUE_DOP
- UISDC\$SET_CHAR_SIZE
- UISDC\$SET_TEXT_MARGINS

New Chapters

Three new chapters describing color concepts and color programming considerations have been added since Version 2.0.

- Color Concepts
- Geometric and Attribute Transformations
- Programming in Color

New UIS Writing Modes

Five new writing modes have been added since Version 2.0.

- `UIS$C_MODE_BIC`
- `UIS$C_MODE_BICN`
- `UIS$C_MODE_BIS`
- `UIS$C_MODE_BISN`
- `UIS$C_MODE_COPYN`

New Technical Character Set Fonts

Twelve new technical character set fonts have been added since Version 2.0.

New Text Attributes

The following new text attributes have been added to the programming interface.

- Character rotation
- Character scaling
- Character slant
- Text formatting
- Text margins
- Text path
- Text slope

Changes to Existing UIS Routines

UIS\$BEGIN_SEGMENT

UIS\$BEGIN_SEGMENT now returns segment identifier that can be referenced by other display list routines. For example, this allows traversing segments and segment paths.

UIS\$MEASURE_TEXT and UIS\$TEXT

You can now use control lists with UIS\$TEXT and UIS\$MEASURE_TEXT.

UIS\$DISABLE_DISPLAY_LIST and UIS\$ENABLE_DISPLAY_LIST

Additional arguments have been included that control display screen and display list updates.

UIS\$SET_POINTER_PATTERN and UISDC\$SET_POINTER_PATTERN

If you are using a color system, you can now specify a pointer pattern outline.

Display Lists and Segmentation

The chapter on display lists and segmentation has been expanded with more examples.

UIS Metafiles

You can create and store metafiles of generically encoded instructions as files and reexecute the file.

Shrinking Viewports and Expanding Icons

Applications can now shrink display viewports and expand icons.

Obsolete Version 2.0 UIS Routines

The following routines are obsolete.

- UIS\$GET_LEFT_MARGIN
- UIS\$SET_LEFT_MARGIN
- UISDC\$GET_LEFT_MARGIN
- UISDC\$SET_LEFT_MARGIN

PART I MicroVMS Workstation Graphics Concepts

Chapter 1

System Description

1.1 Overview

This chapter introduces the MicroVMS workstation graphics system. It is divided into two parts:

- A summary of typical workstation hardware
- A description of the graphics software

1.2 VAXstation Hardware

The MicroVMS workstation can be used as a *standalone* system. It has all the components necessary to run programs and perform tasks without being connected to a host computer. It can also be connected to a host computer and used as a part of a network in a larger system.

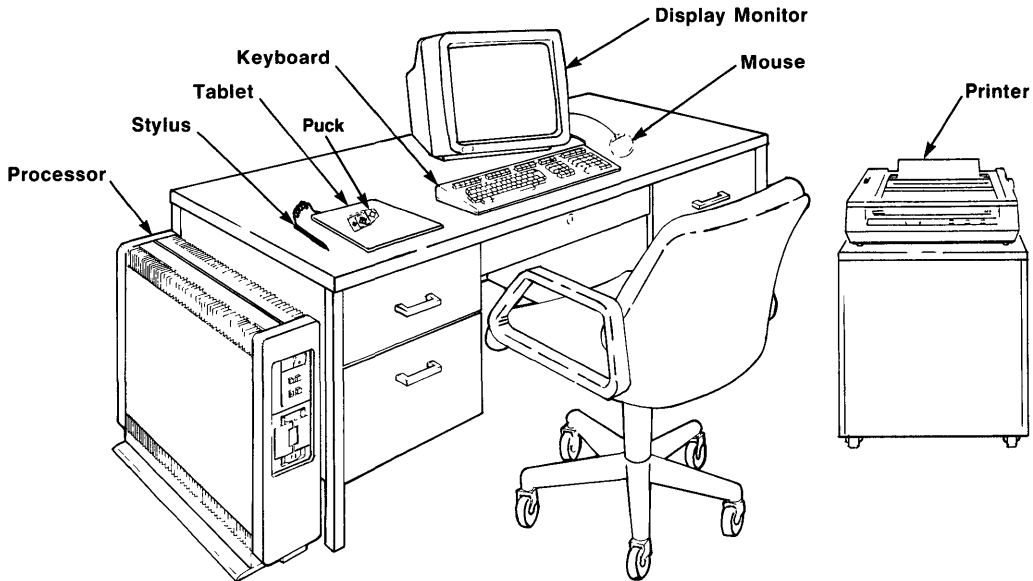
The MicroVMS workstation typically consists of a configuration of the following hardware:

- Processor
- Display monitor
- Keyboard
- Three-button mouse or a tablet
- Communications board
- Printer

An illustration of the typical MicroVMS workstation hardware is provided in Figure 1-1.

1-2 System Description

Figure 1-1 Typical MicroVMS Workstation Hardware



ZK-4616-85

1.2.1 Processor

The processor is the heart of the MicroVMS workstation system. The processor contains the disk drives, all of the memory, any options, and communications hardware for the system. Usually, it houses both fixed and flexible disk drives. The amount of memory it has can vary, depending upon the options installed.

1.2.2 Monitor

The monitor displays text and graphics information. It is a high-resolution bitmap device that can be used to display black-and-white, grey scale, or color graphics.

1.2.3 Keyboard

The keyboard used with the workstation is the DIGITAL LK201, a standard low-profile style keyboard. This keyboard consists of:

- A top row of function keys which are user-definable
- A numeric keypad which is also user definable
- A special keypad which has arrow keys and function keys
- A standard alphanumeric keypad

Some of the top row of function keys are control keys that enable the user to:

- Hold the screen
- Display the operator window
- Switch the windowing system
- Change the active window

In this row, there are also keys that call functions such as cancel, exit, help, and provide aid in editing.

The function keys and numeric keypad keys can be defined by an application program to perform functions suited to a particular application. The arrow keys can be used to move the keyboard cursor within applications. The alphanumeric keypad is similar in function to a typewriter keyboard.

1.2.4 Mouse

The three-button mouse is a medium-resolution, relative pointing device. It is the primary means for a user to point to an object on the screen. When the mouse is rolled on a flat surface, the pointer on the screen moves in a similar fashion. The buttons are used to make selections.

1.2.5 Tablet

The tablet is a high-resolution, absolute positioning device. It consists of a flat tablet, a puck with buttons, and a pen with buttons. When the puck or pen are moved on the tablet, the pointer on the display screen moves in an identical fashion. The buttons are used for selection.

1-4 System Description

1.2.6 Communications Board

The communications board allows the system to be connected with and communicate with other computers.

1.2.7 Printer

The MicroVMS workstation can have a printer connected to the processor's console port or can access printers located at remote location through the network. You can print any rectangular portion of display screen.

1.3 Software

The MicroVMS workstation graphics software is a versatile graphics and windowing interface. It is designed to be used on any of the MicroVAX family of workstation products (such as VAXstations). This graphics interface allows the user to write application programs in VAX MACRO, VAX BLISS, and many other high-level languages. Application programs written to take advantage of this software will be able to create and manipulate windows, display multiple styles of text and sizes, receive input, and draw graphic objects in the created windows.

1.3.1 Graphics Routine Types

The MicroVMS workstation graphics software is composed of callable routines that can be accessed from a high-level programming language. An application program can perform graphics and windowing functions by making calls to the appropriate routines. This software contains routines for creating display windows, drawing lines and text, and building graphic objects.

Routines fall into the following general categories:

- AST-enabling routines
- Attribute routines
- Color routines
- Display list routines
- Graphics and text routines
- Inquiry routines
- Keyboard routines
- Pointer routines
- Sound routines

- Windowing routines
- Device coordinate routines

1.3.2 Human Interface

The MicroVMS workstation provides an interface between the graphics software and the user. This interface is called the *human interface* because it acts to aid the human operator to use the workstation.

One of the things that this interface does is make it easy for the user to create new terminal windows on the screen. The MicroVMS workstation provides the operator with the capability of having the equivalent of many terminals at his or her disposal. A user can easily create emulated DIGITAL VT220 or Tektronix TEK4014 terminals by merely selecting a menu item which creates a window on the screen.

The operator can also control the placement of windows on the screen. Windows can be moved anywhere on the screen (or even partially off of it). They can be hidden from view, pushed behind other windows, popped in front of other windows, and so on. The following list shows some of the operations that are possible.

- Create a new VT220 or TEK4014 terminal window
- Move a window to a different part of the screen
- Push a window behind other windows
- Pop a window in front of other windows
- Shrink a viewport to a icon
- Change the size of a window
- Delete a window
- Switch the keyboard from one window to another
- Suspend all screen activity (hold screen)
- Print any portion (or all) of a window or the screen
- Set workstation attributes
- Get online help

1-6 System Description

1.3.2.1 Terminal Emulation

You can create emulated terminals on the MicroVMS workstation. The programming interface and the capabilities of emulated terminals are the same as the programming interface and capabilities of the corresponding real terminal. The appearance of an emulated terminal on the MicroVMS workstation screen is similar to that of the corresponding real terminal. (It will not be completely identical due to hardware differences.)

An advantage of having several terminal windows is that a job can be started on one terminal, and while it's left running, another terminal can be created and another job started. The user can create as many terminals as desired and switch back and forth between them at will.

VT220/TEK4014

The VAXstation can emulate the DIGITAL VT220 or Tektronix TEK4014 terminal. There can be any number of VT220 or TEK4014 windows on the screen simultaneously. However, only one window may use the keyboard at any one time. The keyboard is assigned to a window by the operator.

VT220 ANSI and DIGITAL private escape sequences, and TEK4014 escape sequences, are interpreted and translated into the appropriate graphics routines.

Programs written using the VAX/VMS operating system will operate in a VT100 or VT220 workstation window without modification.

1.3.2.2 Communication Tools

Users can communicate with the software interface through either the mouse, tablet, or keyboard.

Mouse and Tablet

The mouse and tablet control a cursor called a *pointer* on the screen. When the mouse or tablet is manipulated by the user, the pointer moves on the screen. The pointer is used by an operator to point to things on the screen, such as an item in a menu. The buttons associated with mouse and tablet are used to make selections. The pointer, in combination with buttons on the mouse, can perform several tasks:

- Point to objects on the screen
- Select objects on the screen
- Move objects around on the screen
- Push and pop windows on the screen

- Call menus to the screen
- Switch the keyboard between emulated terminals or windows
- Perform application designated functions

Keyboard

You can use the keyboard to perform the following functions:

- Respond to system prompts
- Provide control keys, such as `HOLD SCREEN` and `CYCLE`
- Provide special keys, such as `HELP`
- Enter data and information into a screen window
- Move a cursor in a window on the screen
- Perform application specific functions

1.3.3 Windowing Feature

The graphics software allows a large number of windows to be created and maintained at the same time. Graphics routines are provided to handle the creation, deletion, and manipulation of overlapping windows. Windows can be popped to the front of the screen, pushed to the background, moved around the screen to a new position, and completely deleted from the screen. The amount and size of information that appears in a window can also be controlled.

1.3.4 Graphics Capabilities

Routines are provided to create new displays and draw graphics within the created displays. A display list, which is an encoded description of the routines used to create the contents of a display, is kept in memory. The display list enables a program to easily pan and zoom portions of a display without having to redraw the entire display. Scaling of the display is done automatically by the graphics software. A display, or a portion of a display, can be mapped into one or more windows on the screen.

Chapter 2

Display Management Concepts

2.1 Overview

This chapter discusses the basic concepts involved in creating a graphic object and displaying it on the workstation screen. Some of the topics covered in this chapter are as follows:

- Virtual displays
- Display windows
- Display viewports
- World and device coordinates
- Display window and viewport scaling

2.1.1 Summary

The MicroVMS workstation graphics software enables application programs to build graphic objects and display them on the workstation screen.

An application program that takes full advantage of the capabilities of the MicroVMS workstation graphics can do the following things:

- Create a virtual display.
- Draw graphics and text into the virtual display.
- Open windows into the virtual display for viewing on an output device.
- Map the windows into display viewports on the workstation screen.
- Manipulate the windows and viewports to display as much or as little of the virtual display as desired.
- Pan, zoom in and out, resize, and duplicate the display windows.
- Manipulate display lists.

2-2 Display Management Concepts

To do these things, an application program must first create a *virtual display* in which to build the object. A virtual display can be thought of as a conceptual display space that has no actual physical size or shape. This conceptual display space, called the *world coordinate system*, is defined by the application program in terms of *world coordinates*. World coordinates are arbitrary units of measure selected by the application program that specify locations (or points) in the world coordinate system using values that are convenient to the application.

World coordinates are automatically translated to *normalized coordinates* (by the graphics software) before being mapped to an output device. Normalized coordinates convert user world coordinates into a single device-independent coordinate system so that the user does not have to deal with several coordinate systems. Normalized coordinates are automatically mapped to the device-dependent coordinates of the physical output device.

A graphic object constructed in a virtual display is not available for display on an output device until a *display window* and *display viewport* are created by the application program.

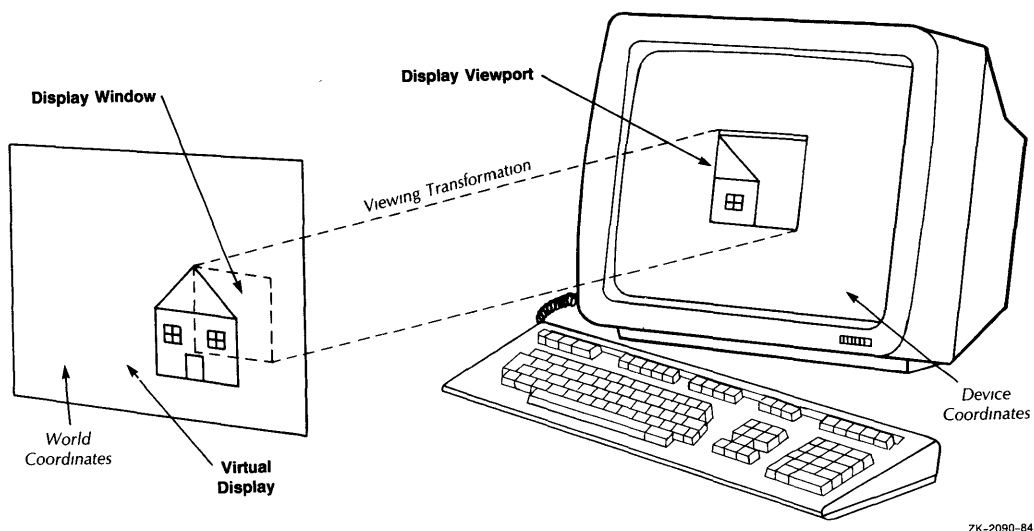
A display window defines what portion of the graphic object in a virtual display is to be viewed. By creating the display window, the program is making the information in the virtual display potentially visible to the user. The information in the display window is not actually visible to a user until the display window is mapped to a display viewport.

A display viewport is the physical region on a display device that is created by the MicroVMS workstation software and controlled by the user. The display viewport is the physical representation of the display window that is mapped to it. It enables a user to view the graphic object that is inside the display window. Figure 2-1 illustrates the relationship between the virtual display, display window, and display viewport.

Physical device coordinates are used in mapping a display window to a display viewport. Physical device coordinates are the physical points on the display screen that are used to locate the graphic object. The process of mapping a graphic object from the world coordinates of the display window to the device coordinates of the display viewport is called a *viewing transformation*. Viewing transformations are handled automatically by the graphics software.

The world coordinates of the display window can be manipulated in relation to the world coordinates of the virtual display to achieve the effects of panning and zooming the graphic object in the display viewport.

Figure 2-1 Virtual Display, Display Window, and Display Viewport



ZK-2090-84

2.2 Coordinate Systems

The MicroVMS workstation graphics environment can be thought of as a two dimensional plane. Because of this, the *Cartesian coordinate system* applies in describing points within this environment. Cartesian coordinates take the form of x,y , where x is the horizontal axis and y is the vertical axis. A point on this plane is specified by a coordinate pair. The area of this plane that is specified by coordinate pairs is called the coordinate space.

The MicroVMS workstation graphics software makes use of four Cartesian coordinate systems: world, normalized, absolute, and viewport-relative device coordinates.

2.2.1 Device-Independent Coordinate Systems

Device-independent coordinate systems mediate between the requirements of the application program and the graphics subsystem versus those of the output device.

2-4 Display Management Concepts

2.2.1.1 World Coordinates

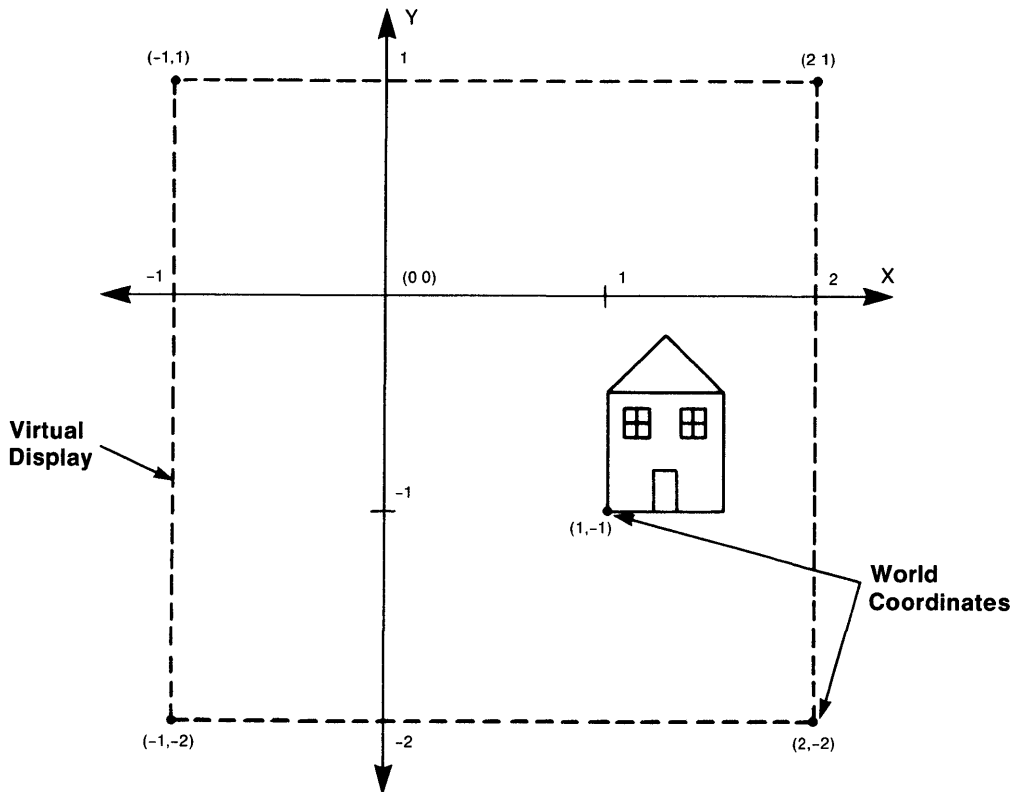
An application program uses world coordinates to describe a virtual display and to build a graphic object within it. Initially, the application program creates a virtual display and specifies a convenient world coordinate system to use when referring to the virtual display. Next, the program specifies the size and location of objects to be created within the virtual display, using the same coordinates.

World coordinates are device-independent Cartesian coordinates that are specified by the application program. They provide a means of locating the points in a virtual display. The range of world coordinate values is specified when the virtual display is created. In this way, the virtual display can be created to any proportions that are selected by the application program. World coordinate values are given as floating-point numbers.

The world coordinate system can represent any unit of measure. World coordinates enable application programs to use convenient increments of measurement when constructing a graphic object. If the program is accessing information from a data base, it could specify world coordinates that are meaningful for the data used. For instance, if an application is drawing a chart showing the sales of a company's product during a holiday season, it could use convenient measurements representing units sold in thousands versus the time in weeks. Or, if the application program is drawing a graphic object, it could use measurements that make sense for the object. For example, a virtual display containing a map of the United States might logically have world coordinates representing measurements in miles or kilometers. A floor plan of a house might likely use world coordinates representing feet and inches, or meters and centimeters.

Figure 2-2 shows a world coordinate system that describes a virtual display in which an object has been constructed.

Figure 2-2 World Coordinate System and Virtual Display



ZK-4617-85

2.2.1.2 Normalized Coordinates

Normalized coordinates are device-independent coordinates that are defined by the graphics software. They are used to describe the virtual display in physical terms that any output device can use. An output device cannot use the arbitrary world coordinates that an application program uses to describe a virtual display. Instead, each kind of output device has its own device-specific coordinates that it uses to locate and build the graphic object. Normalized coordinates can be thought of as a way for the graphics software to normalize these different coordinate systems so that a graphic object can be mapped from a virtual display to any output device.

2-6 Display Management Concepts

Normalized coordinates are not directly used or manipulated by application programs. They are used internally by the graphics software. The mapping of normalized coordinates into device-specific display coordinates is handled entirely by the software.

Normalized coordinates provide a means of delaying the actual mapping of an application program's world coordinates to device-specific coordinates until the actual output device is established.

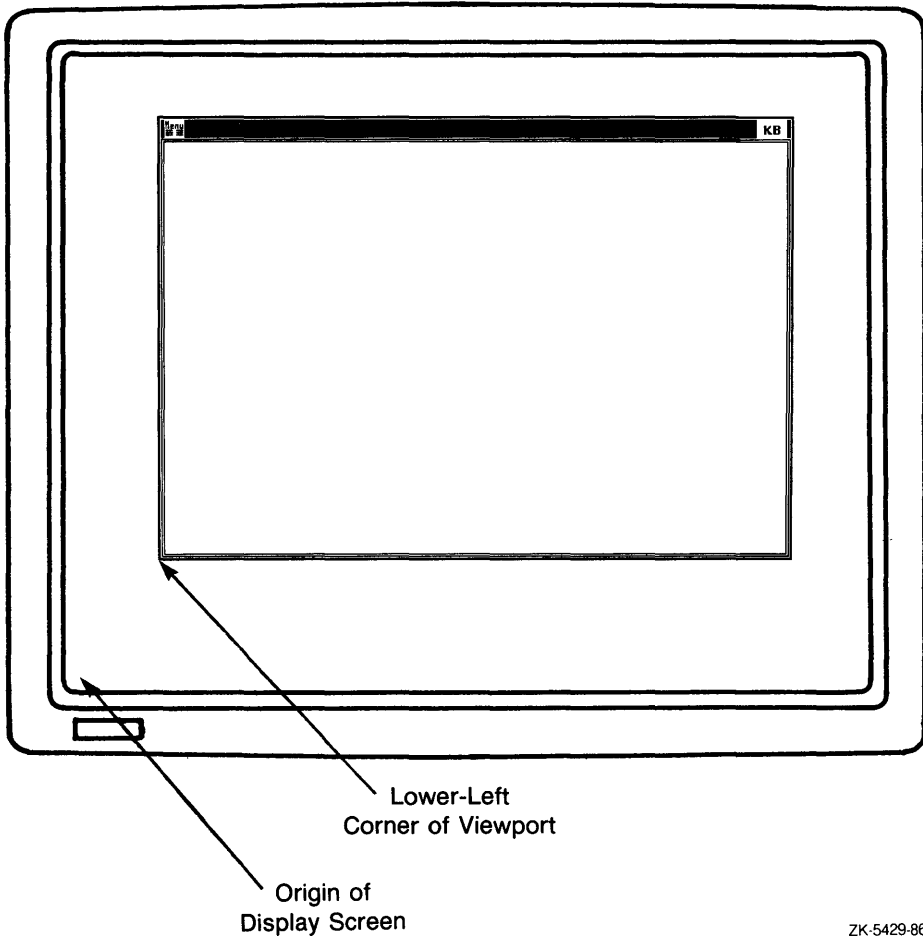
2.2.2 Device-Dependent Coordinate Systems

Output devices use device-dependent coordinate systems to map graphic objects on the display screen or to print objects on a printer. Device-dependent coordinates are physical device coordinates that denote some physical unit of measure such as pixels, centimeters, or inches. Such physical device coordinates reflect device-dependent mapping and drawing characteristics of the output device.

2.2.2.1 Absolute Device Coordinates

Absolute device coordinates are physical device-dependent Cartesian coordinates that specify positions on the MicroVMS workstation display screen. The position is specified in centimeters relative to the lower-left corner of the display screen. Typically, viewport placement, pointer position, and tablet placement use absolute coordinates. Figure 2-3 illustrates viewport placement on the VAXstation screen relative to the lower-left corner of the screen.

Figure 2-3 Absolute Device Coordinates



ZK-5429-86

2.2.2.2 Viewport-Relative Device Coordinates

Many MicroVMS workstation graphics software routines utilize a special type of physical device coordinates called *viewport relative* device coordinates. Viewport relative device coordinates are physical device coordinates that specify positions within a display viewport. The position specified is relative to the lower-left corner of the viewport. Viewport-relative device coordinates are always positive.

Viewport-relative device coordinates are specified in units of *pixels*. A pixel is the smallest displayable unit on a display screen. The MicroVMS workstation graphics software takes care of all mapping of display windows to the display screen.

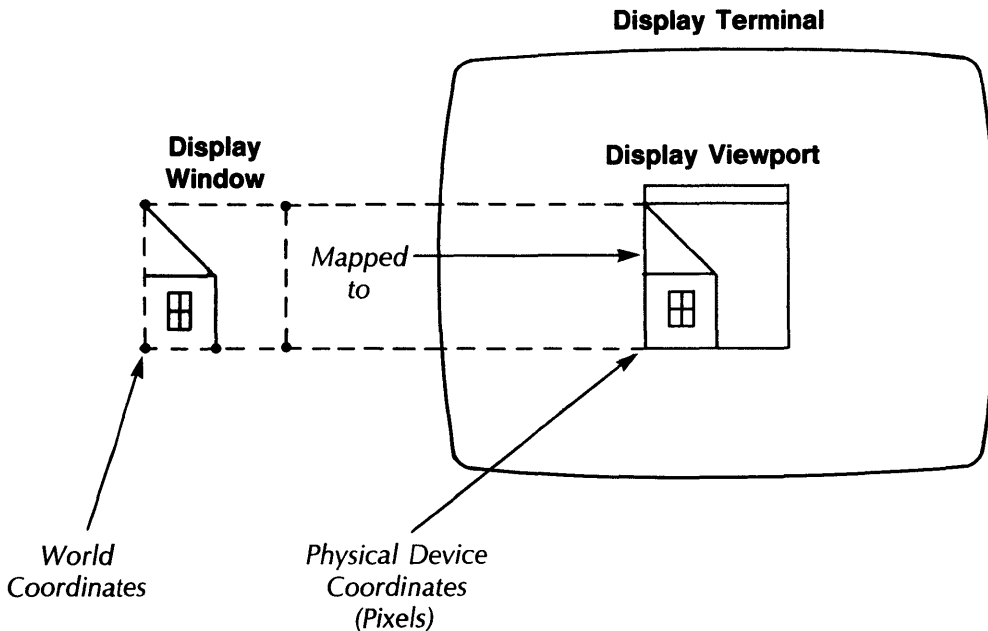
2-8 Display Management Concepts

Viewport-relative device coordinates are used in mapping graphic objects from a display window to a display viewport on a physical display device.

In order to display a graphic object in a display viewport on a display device, the world coordinates of the object must be transformed to the viewport-relative device coordinates of the display device.

Figure 2-4 shows an object in a display window being mapped to a display viewport on a physical display device. In this illustration, the world coordinates of the display window undergo a viewing transformation to the physical device coordinates of the display device.

Figure 2-4 Mapping a Display Window to a Display Viewport



ZK-4624-85

2.3 Virtual Displays

A virtual display is a conceptual display space created by an application program. It is used by an application program as the place where graphic objects are constructed. All text and graphics output of the application program are written to a virtual display.

A virtual display has no physical size (dimensions of length and width). Therefore, objects constructed in a virtual display also have no actual physical dimensions. You cannot measure a virtual display or the graphic objects within it.

Instead, a virtual display and the objects within it have relative sizes and proportions. The comparison of the relative proportions of the vertical and horizontal components of an object in a virtual display is called the *aspect ratio* of the object. The aspect ratio is used in referring to an object's relative size in a virtual display.

To create a virtual display, an application program specifies a coordinate range in the world coordinate system. The coordinate range establishes the relative size, or aspect ratio, of the virtual display. Objects constructed in the virtual display are also specified in terms of world coordinates and also have an aspect ratio. The aspect ratio will later affect how the virtual display and the objects it contains map to the display window.

Refer back to Figure 2-2 which shows a graphic object drawn in a virtual display. Both the virtual display and the graphics object are specified in terms of world coordinates.

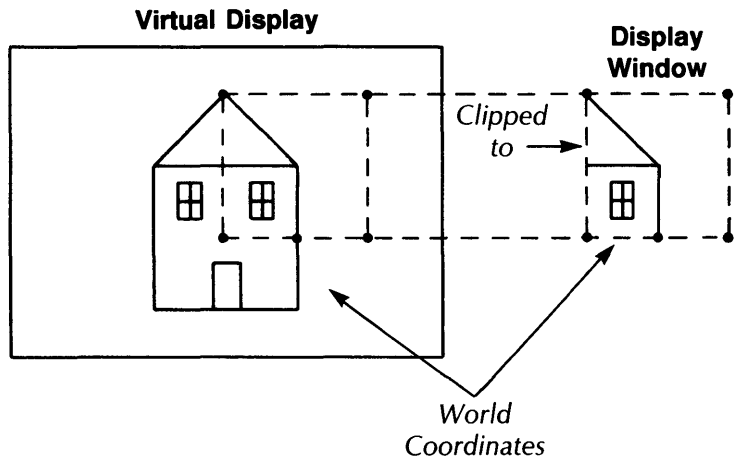
2.4 Display Windows

A display window is used to display all or a part of the contents of a virtual display. Display windows are created by an application program. A display window is used by the application program to control how much of a virtual display is potentially available for the user to view. A display window can be the size of an entire virtual display or just a small portion of it. There can be one or several display windows active at one time in a virtual display.

The relative proportions and location of a display window are specified by an application program in terms of world coordinates. Therefore, the amount of the virtual display that is encompassed by a display window is relative to the world coordinates of the virtual display. By specifying the proportions and location of the display window, an application program determines what portion of the graphic object within a virtual display is viewable.

The world coordinate boundaries of a display window define what is called a *clipping rectangle*. Any graphic object that lies within the clipping rectangle is potentially visible in the display viewport. Objects that fall outside of the clipping rectangle are not viewable and are clipped from the window as illustrated in Figure 2-5.

Figure 2-5 Display Window in a Virtual Display



ZK-4625-85

2.5 Display Viewports

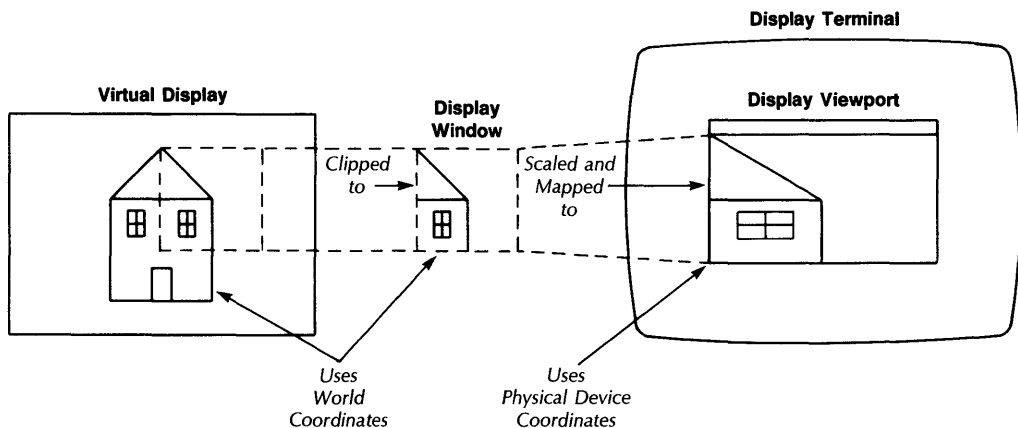
A display viewport is the area of the physical display screen to which a display window is mapped. The display viewport is the user's means of viewing the contents of a display window. A display viewport is always associated with a display window and is the mechanism by which the display window is displayed on the screen.

They can vary in size and shape, and can be located anywhere on the physical display screen. There can be as many viewports as desired on the screen at a time. If viewports overlap each other, they will occlude in the areas that overlap. The last viewport created will be on top and visible. However, the operator can modify which viewport is on top at any one time.

The display window is mapped and scaled to the display viewport automatically by the graphics software. Normally, the display window is mapped to the display viewport on a one-to-one basis. That is, the boundaries of the display viewport always implicitly default to the same size and shape as that of the display window. However, it is possible for the application program to explicitly specify that the display window be of a size or shape that is different than that of the display viewport; or, that the display viewport be of a size or shape that is different from that of the display window. The effects that are achieved when the display window and display viewport are of a different size or shape are discussed in the following sections of this chapter.

Refer to Figure 2-6 for an illustration of the relationship between the virtual display, the display window and the display viewport. This illustration shows how a graphics object in a virtual display is clipped to the display window, scaled and mapped into a display viewport, and displayed on a physical display device such as a terminal screen.

Figure 2-6 Displaying a Graphic Object



ZK-4618-85

2.6 Display Window and Viewport Scaling

Graphic objects on the display screen can be magnified or reduced in size by manipulating the relative sizes of the display window and the display viewport. The following list describes the various effects that can be achieved and the method used to accomplish each effect.

Magnifying

To magnify the graphic object, use one of these two methods:

- Decrease the size of the display window without altering the viewport size.
- Increase the size of the display viewport without altering the window size.

2-12 Display Management Concepts

Reducing

To reduce the graphic object, use one of these two methods:

- Increase the size of the display window without altering the viewport size.
- Decrease the size of the display viewport without altering the window size.

Panning

To pan the graphic object, use this method:

- Move the display window within the virtual display without altering the display viewport.

Changing View Size

You can change the area of the virtual display that is being viewed, without performing scaling, in the following ways:

- To increase the area of the virtual display being viewed, expand both the display window and the display viewport proportionately.
- To decrease the area of the virtual display being viewed, contract both the display window and the display viewport proportionately.

2.6.1 Distortion of Graphic Objects

The aspect ratio of the virtual display, the display window, and the display viewport are the factors that determine whether a graphic object will be distorted when it is mapped to the display screen. The display viewport can have any proportions width to height that is specified (within the limits of the display device). If the proportions of the display viewport do not match the proportions of the display window, a stretching or squeezing effect occurs with the graphic object. The exact effect depends upon the proportional differences between the viewport and window. This happens because the graphics software is trying to make the display window fit the display viewport. The transformation of the graphic object affects different types of objects in different ways:

- Straight lines remain straight, but may differ in length and slope, depending upon the window size and the coordinate system.
- Curved lines can change somewhat in shape. The amount and nature of the change depends upon the characteristics of the graphic object and the mapping (transformation) from display window to viewport.

- Arcs change their shape and size. For instance, an ellipse may change its proportions.
- Graphics text (specifically character size and spacing) is not adjusted to fit the required number of characters into the display viewport. The size and spacing of text characters is fixed and will not distort. However, the starting position of the text may change, depending upon the transformation which occurs between window and viewport.

Distortion can be corrected in the following way:

The application program can create a display viewport whose proportions are appropriate for a particular graphics window in world coordinate space. Because the display window can have any proportions in world coordinate space, a display viewport of the proper proportions for a display window that is square, tall and narrow, short and wide, or any other proportions, can be created.

2.7 Display Lists

A display list is a device-independent encoding of the exact contents of a virtual display. The graphics software maintains and uses display lists to achieve the following goals:

- Allow the automatic management of panning, zooming, resizing, and duplication of display windows.
- Allow the structuring of virtual display objects.
- Allow objects in a virtual display to be viewed simultaneously within several display viewports.
- Allow the storage and reexecution of UIS pictures
- Allow editing of UIS pictures

2.8 Generic Encoding and UIS Metafiles

Whenever a graphic object is drawn in the virtual display or an attribute is modified, an encoded entry of the object or attribute modification is added to the display list.

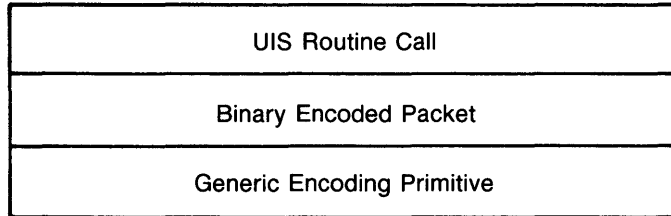
2-14 Display Management Concepts

Such entries allow any application to extract arbitrary output from a virtual display, give it to an intelligent application or store the data as a *generically encoded* file or buffer known as a *metafile*, and then later reexecute the generically encoded binary stream into a new virtual display.

Generic encoding is both device independent and self describing.

When UIS routines are executed, a binary encoded packet of values is constructed and stored as display list entries. When the binary encoded packet is extracted from the display list used, it becomes a generically encoded UIS metafile. Such metafiles can be reexecuted to invoke the appropriate internal generic encoding routines.

Figure 2-7 Display List Extraction



ZK-5428-86

Although many UIS routines have corresponding generic encoding primitives there is not necessarily a one-to-one mapping between UIS routines and generic encoding routines or between the UIS routine arguments and generic encoding routine arguments.

Chapter 3

Graphic Objects and Attributes

3.1 Overview

This chapter discusses the basic building blocks that are used in constructing graphic objects in a virtual display. These basic components are:

- Text and graphics routines
- Attributes and attribute modification routines
- Attribute blocks
- Segments

These topics are discussed in greater detail in the following sections of this chapter.

3.2 Summary

Text and graphics routines (sometimes called output routines) are the fundamental building blocks that an application program uses to create graphic objects. These routines are used to specify lines, circles, text, or other graphic objects. The particular details (or attributes) of the way a text or graphic object look when it is displayed is determined by the *attribute block* associated with it.

An attribute block is a group, or set, of *attributes*. Attributes are values which specify various things about the appearance of a text or graphic object. Every text and graphics routine used in an application program is required to specify an attribute block that it will use.

Attribute routines are used in an application program to specify or change the current value of an attribute associated with an attribute block. The changed attribute value affects subsequent text and graphics routines that use the changed attribute block. An attribute routine is required to specify which attribute block in the application program it is affecting.

3-2 Graphic Objects and Attributes

Application programs are allowed to group associated attribute, graphics and text routines together. A group of attribute, graphics and text routines is called a *segment*. Segments provide the program with a convenient way of viewing several attribute, graphics and text routines as a single unit.

An application program can associate graphics and text routines or even entire segments with *application-specific* data. The application program is allowed to store data which is application-specific in the generic encoding stream. In this way, if a portion of a display screen is copied, stored and then later used (restored) the program will be able to associate internal information with the graphic object.

3.3 Text and Graphics Routines

Graphics and text routines map objects directly into the virtual display. They can be used to create new objects or modify an existing one. Application programs use text and graphics routines to draw lines, circles, text, and other graphic objects. They can be combined in various ways to form a desired graphic object.

Each text and graphics routine has two required arguments: one argument that specifies the virtual display in which to draw a graphic object, and another argument that specifies the attribute block to be used when drawing the graphic object.

The way that a text or graphics routine draws a graphic object is influenced by several factors. One of the major factors which determines the appearance of a graphic object is the attributes that are associated with it.

3.4 Attributes

Attributes specify the appearance characteristics of graphic objects created by text and graphics routines. They are the factors that influence the way a graphic object appears on a display device. Color intensity, style, mode, width, and so on, are all characteristics that attributes can determine. Once specified, attribute values stay the same until explicitly changed. For example, if the line width is decreased, all lines drawn are drawn to that thickness unless the line width is changed. If the application program increases the line width, all lines are drawn to the same increased thickness until the line width is changed again.

Each type of graphic and text object has a set of unique attributes. For example, attributes that affect graphics do not affect text; the opposite is also true. There are, however, general attributes that affect all routines. For example the background has an attribute that can be set to determine the way the background will appear. The background can be thought of as all parts of a display that are not covered by an object created by a text or graphics routine.

Attributes can be divided into the following general categories:

- General attributes
- Text attributes
- Graphics attributes
- Window attribute

These categories are discussed in the following sections of this chapter.

3.4.1 General Attributes

General attributes apply to all types of text and graphics routines. General attributes include the following kinds of attributes:

- Writing color
- Background color
- Writing mode

Writing Color

The writing color attribute assigns the writing color. This attribute is used by all text and graphics routines (such as lines, text, etc.). It is expressed by specifying an index into a color map.

Background Color

This attribute assigns the background color. It is expressed by specifying an index into a color map.

Writing Mode

This attribute assigns the mode of writing text or graphics. In particular, the writing mode determines the exact way that a text or graphics routine will use the writing and background colors to display a graphic object.

3.4.2 Text Attributes

Font set

The font set attribute specifies the font set that is used to define text characters. Fonts express the size and shape of the characters in physical dimensions. This attribute enables text to be displayed in the right size by display routines during text plotting. You can choose from a variety of multinational character set fonts and technical character set fonts.

3-4 Graphic Objects and Attributes

Character spacing

The character spacing attribute defines character spacing for width and height of character sizes. It is defined as the additional unit of increment beyond the normal character size for highly spaced characters. This attribute is specified as a floating-point number. It is multiplied by the normal character size to produce the actual spacing distance. If zeros are specified, then no additional spacing is performed. Negative values are also allowed. When used, the spacing is reduced instead of increased. Negative values for this attribute can cause the characters to overlap in some cases.

Text Path

The text path is the direction of text drawing. The text path specification consists of two parts—the major path and the minor path. The major path refers to the direction in which character are drawn on a line. The minor path refers to the direction used for beginning a new line of text. The following table lists the major path and minor path available.

- Left to right (default major text path)
- Right to left
- Bottom to top
- Top to bottom (default minor text path)

Text Slope

Text slope represents the angle between the actual path of text drawing and the major text path. The actual path of text drawing connects the baseline points of each character cell.

Text Margins

The text margins attribute specifies a starting margin and the x coordinate distance to the ending margin.

Text Formatting

The text formatting attribute along with the text margins attribute positions text flush against either or both margins, centered, or with no formatting at all. UIS supports four types of text formatting modes as follows:

- Left justification
- Right justification
- Center justification
- Full justification

Character Rotation

Individual characters are rotated counterclockwise from 0 to 360 degrees. The angle of rotation is the angle between the baseline vector of the character cell and the actual path of the text drawing.

Character Slant

The character slant attribute specifies the angle between the character cell's up vector and baseline vector. The angle of character slant can be expressed as a negative or positive value.

Character Size

Character scaling allows you to increase the height and width of characters drawn in the virtual display.

3.4.3 Graphics Attributes

Graphics attributes, or line attributes, affect graphic objects such as lines, polylines, polygons, rectangles, arcs, and curves. They determine the line style and width, and control filling of objects, among other things.

Current Line Drawing Width

The current line drawing width sets the line width in terms of world or device coordinate units. Line width is specified as a floating-point number that is either interpreted as a world coordinate width or multiplied with the standard line width for a device to produce the desired line width.

Line Style

The line style attribute sets the current line style of line routines. It is a bit vector that is used to indicate the color of each pixel to be drawn. The color can be designated to be either the same as the foreground or the background. The bit vector is repeated as many times as necessary to draw all the pixels in the line.

Fill Pattern

The fill pattern attribute specifies the fill character to be used for filling closed figures such as polygons, circles, and ellipses. The fill pattern is specified as a font file and the index of a character in that font file. The pattern defined by the character is used to fill the figure. Refer to Appendix D of this manual for further information about fill patterns.

3-6 Graphic Objects and Attributes

Arc Type

The arc type attribute specifies the way an open arc of a circle or ellipse should be closed. This attribute can have the following values:

- Open—when specified as open, the arc is not closed off.
- Pie—when specified as pie, two radii are drawn from the endpoints of the arc to the centerpoint (forming a pie shape).
- Chord—when specified as chord, a line is drawn between the two endpoints of the arc connecting them together.

3.4.4 Window Attribute

Clipping Rectangle

The clipping rectangle is the area of a virtual display that is made available for the user to view. The clipping rectangle is specified as the corners of a world coordinate rectangle that all drawing operations are clipped to. Objects, or parts of objects, outside of the clipping rectangle cannot be viewed.

3.5 Attribute Blocks

An attribute block is a set of attribute values that describes the appearance of any graphic object that is created by an application program. Each attribute block contains attributes for graphics, text, and general display characteristics such as writing mode and background and foreground indices.

There can be up to 256 different attribute blocks addressable at any one time. They are addressed by numbers ranging from 0 to 255. Application programs assign and use attribute block numbers.

3.5.1 Attribute Block 0

Attribute block 0 is a special attribute block that is specified by the graphics software. This attribute block contains a standard set of text and graphics attributes. The attributes in this block cannot be modified by the application program. Attribute block 0 is read only. There is no convention on the naming and usage of attribute blocks, with the exception of attribute block 0. This attribute block is reserved by the graphics software as a default attribute block.

Attribute block 0 provides default attribute values that can be used by an application program. It also serves as an attribute block template for an application programmer to use when creating alternate attribute blocks.

3.6 Segments

A segment is a designated group of attribute block, graphics and text objects. Segments allow the application program to use a special attribute without the need for knowing which particular attribute blocks are not being used by other parts of the program. Another major use of segmentation is to implement transformations either on a per-segment basis or on the entire segment tree. This provides convenience for the programmer and increased modularity for the program.

Nested Segments

Segments can be nested. Each nested segment uses the current set of attribute blocks of higher level segments. This makes it simpler to create segments without having to redefine attribute blocks. However, modifications of attribute blocks in a segment do not affect the attribute blocks of higher level segments.

Extracting and Reexecuting Segments

An application program can take the contents of a file containing a display list of a virtual display and execute it into another virtual display as a segment. The attributes of the original virtual display should not affect the virtual display segment which is being inserted.

3.7 Viewing Transformations

The *viewing transformation* is the mapping of the display window to the display viewport. The viewing transformation can affect the appearance of a graphic object when it is viewed on a display screen. The shapes of the display window and the display viewport will affect the way text and graphic objects look when they are displayed.

3.8 Two-Dimensional Geometric Transformations

Geometric transformations can also alter the way graphic objects are displayed through scaling, translation, and rotation. All of these methods involve manipulation of the object's angular orientation or shape in the virtual display.

3-8 Graphic Objects and Attributes

Scaling

The term *scaling* applies to the proportional expansion or reduction of graphic objects on the display screen. For example, if the display window and viewport shapes are different in proportion, the graphics software has to squeeze or stretch the window to fit the viewport. The distortion of the graphics window causes distortion of the graphic objects in that window. Different graphic objects are affected in different ways. Refer to Chapter 2 for further information about the distortion of graphic objects.

Translation

The points that define the position of graphic object in a coordinate system are *translated* when its coordinates are altered without changing its angular relationship with other object or the implied angular relationship between the object and the coordinate system. For example, two lines are moved in the coordinate system, and yet remain parallel.

Rotation

When a graphic object turns on a pivotal point or axis, it is rotating. It can rotate with respect to some point on its surface, or it can *revolve* around some external point. In order to give the appearance of rotation on the display screen, you must first translate the axis of the object to the *origin* or center of the coordinate system.

Chapter 4

Color Concepts

4.1 Overview

Depending on the type of VAXstation available to you, you can display graphic objects in black-and-white, grey scale, or color. The VAXstation offers you a number of color options. However, there are several concepts you should be aware of at the outset. This chapter discusses these concepts and the features of the color subsystem in the following topics:

- Color hardware systems
- UIS virtual color maps
- Miscellaneous color concepts

See Chapter 16 for more information about programming in color.

4.2 Color Hardware Systems

There are three types of VAXstation hardware systems: (1) *monochrome* displays black and white only, (2) *intensity* displays shades of gray or achromatic color, and (3) *color* displays shades, tints, hues or chromatic colors. UIS supports all three color systems.

4.3 Raster Graphics Concepts

The VAXstation display screen consists of a set of picture elements called *pixels*. Pixels are the smallest displayable unit of a graphic object. The rectangular set of pixels on the VAXstation screen is a *raster*. Graphic objects are written by illuminating the necessary pixels along the path of points that geometrically describe the object. Each pixel has an address and a binary value associated with it. Pixel values determine the color of graphic objects.

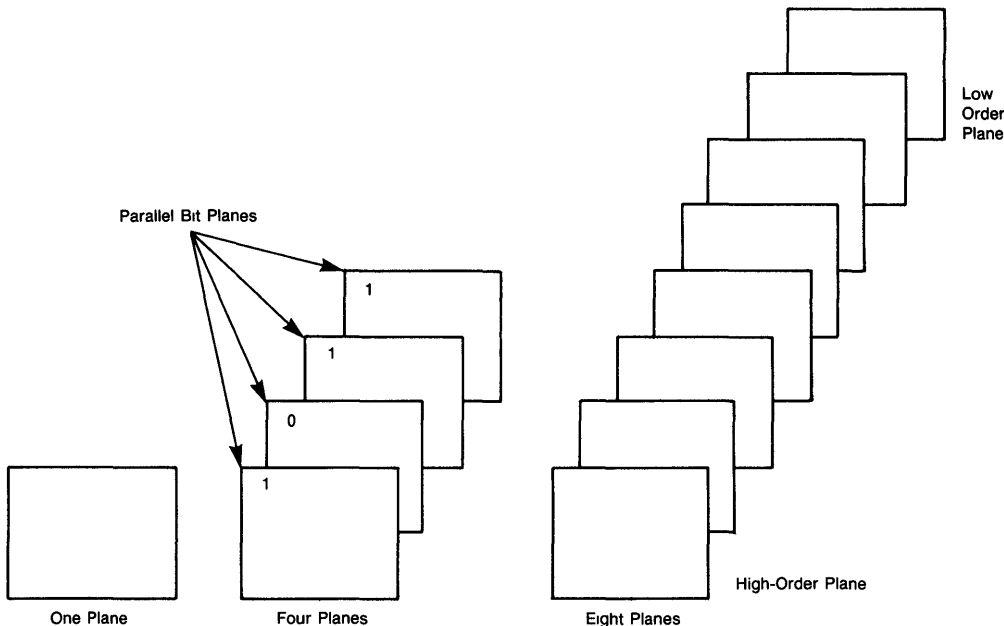
4.3.1 Hardware Interpretation of Pixel Values

The number of possible pixel values depends on the number of bit planes or *planes* of memory that the system hardware supports. You can think of a plane as an allocation of memory where each bit on a plane maps to a pixel on the display screen. Conversely, each pixel has an address in memory. The following table shows the relationship between the number of planes supported in hardware and the number of the possible pixel values.

Workstation	Number of Planes	Number of Possible Values
Monochrome	1	2
Intensity or color	4 or 8	16 or 256

Figure 4-1 show how pixel values are represented in single- and multiplane systems.

Figure 4-1 Bitplane Configuration in Single- and Multiplane Systems



ZK 5242 86

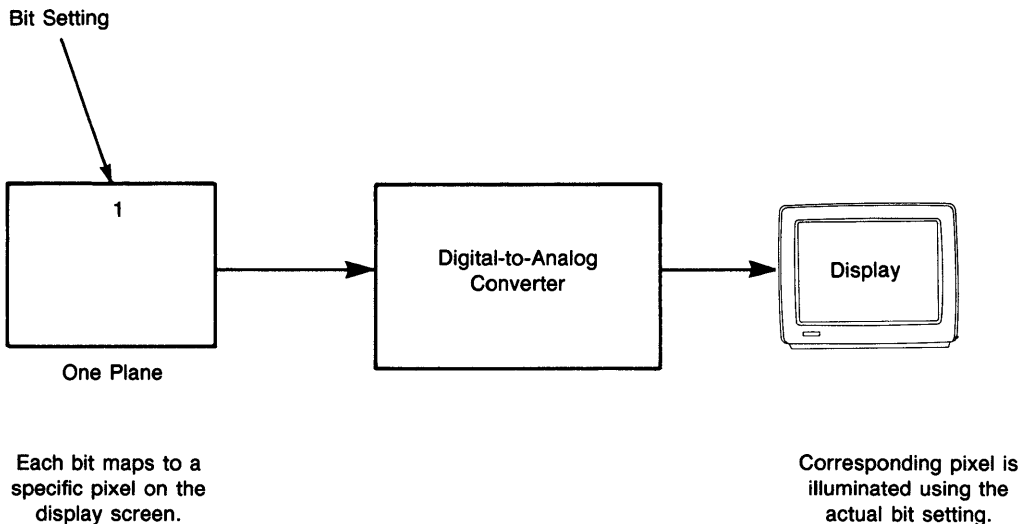
In Figure 4-1, a pixel on the VAXstation screen is associated with four corresponding bits in memory on each bit plane of a four-plane system. If the bit settings are arranged as a binary value corresponding to the high- and low-order planes, they would appear in the following order: 1011_2 .

Therefore, the pixel value would be 11_{10} . A pixel in a four-plane system can have a maximum of 16 values. The pixel value can be used in two different ways, as a *direct color value* or as a *mapped color value*.

Direct Color Value

If the pixel value were used as a direct color value, each of the possible pixel values would directly specify a color. In other words, the pixel value would be sent directly to system hardware, such as a digital-to-analog converter, and would be used as the actual color value of the graphic object. For example, the VAXstation monochrome system, which is a one-plane system, interprets pixel values as direct color values where 0 is black and 1 is white.

Figure 4-2 Direct Color Values



ZK-5240-86

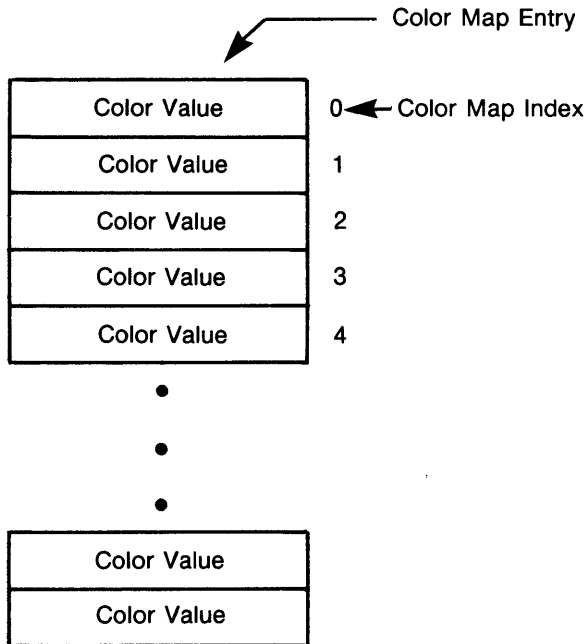
Mapped Color Value

When pixel values are interpreted as mapped color values, they indirectly specify an actual color value located in a hardware *color look-up* table or hardware color map.

The pixel value is an *index* to an entry in the color map.

4-4 Color Concepts

Figure 4-3 Hardware Color Map



ZK-5241-86

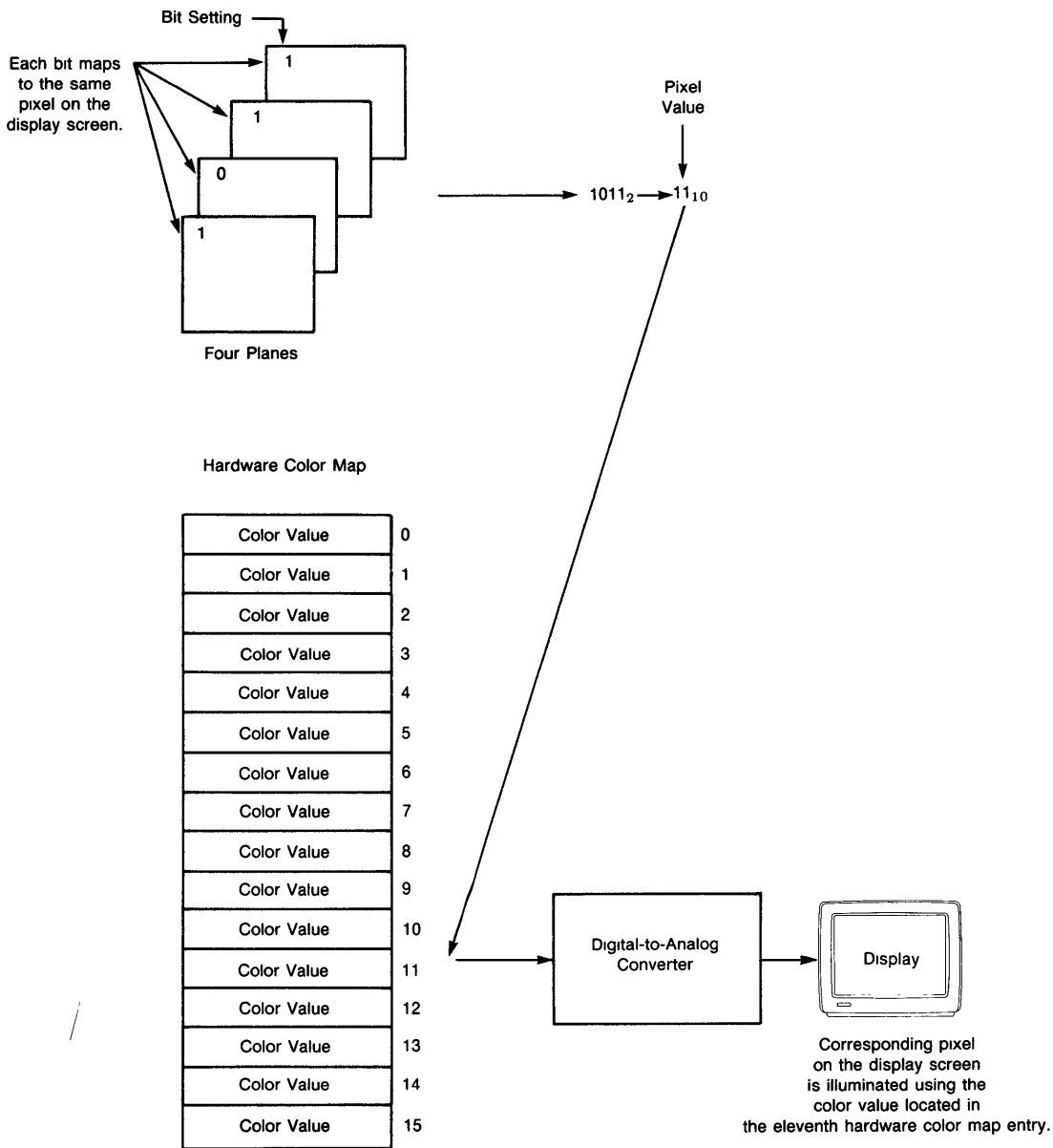
The size of the hardware color map is identical to the number of possible pixel values and is the maximum number of colors that can be displayed simultaneously. Table 4-1 lists the size of the hardware color map in intensity and color systems.

Table 4-1 Hardware Color Map Characteristics

System	Number of Planes	Number of Entries
Intensity	Four	16
	Eight	256
Color	Four	16
	Eight	256

Each hardware color map entry contains a color value to be displayed for each pixel. Conversely, the value of each pixel is the hardware color map index of a color map entry containing the actual color value. The pixel on the VAXstation screen is illuminated using this color value.

Figure 4-4 Mapped Color Values in Four-Plane System



4-6 Color Concepts

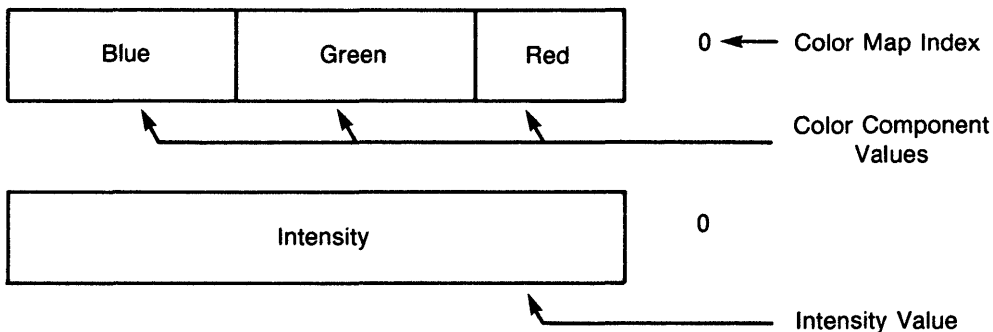
For example, an eight-plane VAXstation intensity or color system has a hardware color map with 256 entries. Each color map entry contains color values that are *RGB color components* and that define the desired color.

4.3.2 Color Representation Models

Color values are expressed according to the requirements of the particular color representation model used. Three well known color representation models are hue lightness saturation (HLS), hue saturation value (HSV), and red green blue (RGB). The UIS base color model is the RGB model. RGB color values are in the range 0.0 to 1.0, inclusive. Red, green, and blue color component values comprise a single color value on a VAXstation color system.

Intensity values, the color values associated with shades of gray are specified as a single value in the range 0.0 to 1.0, inclusive. Figure 4-5 shows RGB and intensity color values as hardware color map entries.

Figure 4-5 RGB and Intensity Color Values as Hardware Color Map Entries



ZK-5239-86

4.3.3 Color Palette

Your *color palette* is the number of possible colors that you can specify. Table 4-2 show the color palette available on each color system.

Color Palette Size and Direct Color Systems

On direct color systems, the palette size is identical to the number of simultaneously displayable colors. For example, the size of the color palette of a VAXstation monochrome system is 2. Only two possible colors, black and white, can be displayed simultaneously on the screen.

Color Palette Size and Mapped Color Systems

On mapped color systems, the palette size is, typically, much greater than the number of the simultaneously displayable colors. The palette size is determined by the precision of color components' specification. For example, on VAXstation color system, each color component can be specified with eight binary bits of precision for each red, green, and blue color components or 2^{24} or 16,177,216 possible colors.

Table 4-2 Color Palette

System	Possible Colors
Monochrome	black and white
Intensity	up to 2^{24} shades of gray
Color	up to 2^{24} chromatic colors

4.4 UIS Virtual Color Maps

An application that uses hardware color resources, that is, the hardware color map must be aware of the hardware system limitations. The application must know the color characteristics of the hardware as well. Is the system direct color or mapped color? What is the precision of the color representation values for each RGB color component? What is the range of possible pixel values?

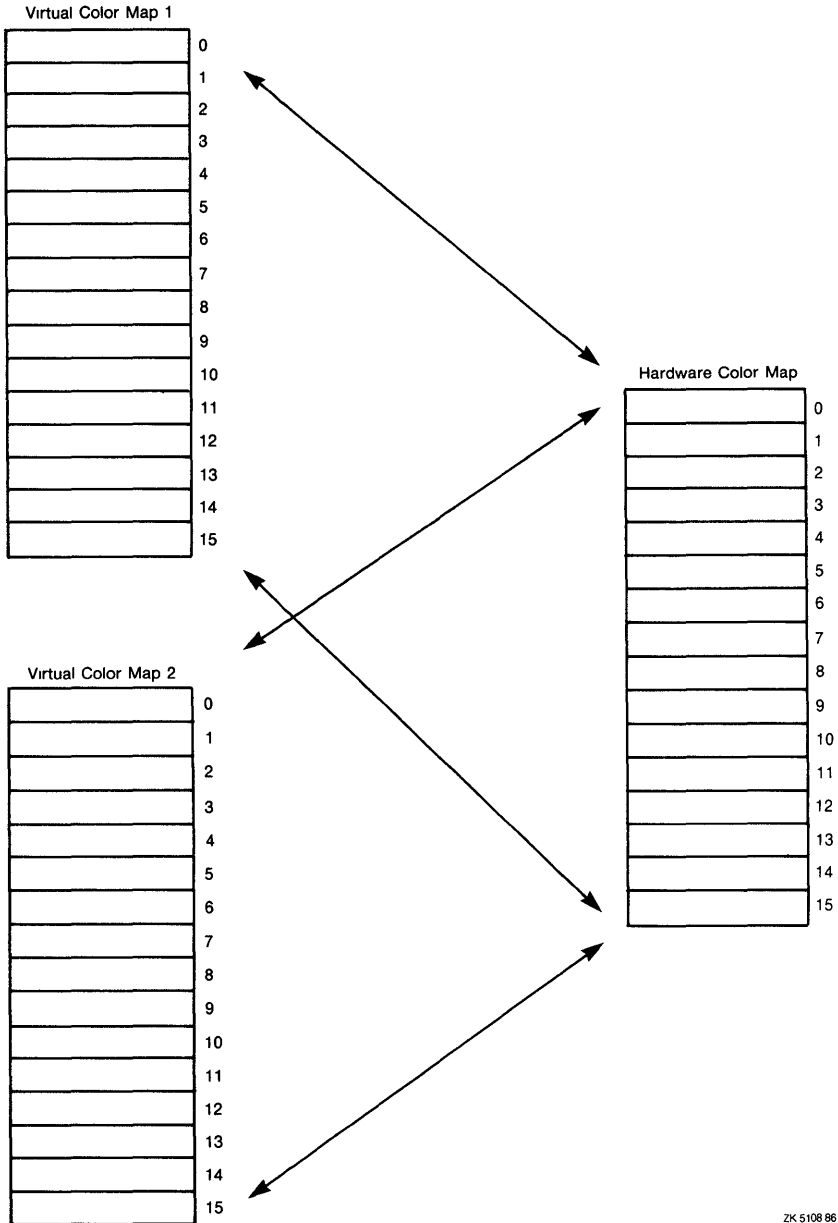
The hardware color map contains a finite number of entries—for example, 16 entries in a four-plane system. Concurrent processes executing in the same display space must somehow share system color resources.

Why Use Virtual Color Maps?

The *virtualization* of the hardware color map solves problems arising from individual applications requiring large amounts of system resources. It also solves the problem of many processes competing for finite color resources. The use of virtual color maps is analogous to the use of virtual memory in a multiprogramming environment where many processes must access physical memory. When concurrent processes require collectively more color map entries than exist in the hardware color map, the color values associated with each competing process are swapped in and out of the hardware color map as *virtual* color maps. Swapping virtual color maps in and out of the hardware color map is a means of arbitrating hardware color map use across applications. The process of loading or writing values of the virtual color map into the hardware lookup table is transparent to the user. Figure 4-6 illustrates the swapping of two 16-entry virtual color maps into a 16-entry hardware color map.

4-8 Color Concepts

Figure 4-6 Swapping Virtual Color Maps



Applications see only a virtual color map, not the underlying hardware resources. Each virtual display has a virtual color map associated with it.

Characteristics of Virtual Color Maps

A virtual color map is flexible enough to meet the needs of a wide range of applications. Virtual color map size can range from 2 to 32,768 entries. If you do not specify a virtual color map, a two-entry virtual color map is created by default. The virtual color map size does not have to match that of the hardware color map. Although virtual color maps are potentially shareable among applications, they are private by default. Virtual color maps can be specified as *resident*, that is, nonswappable in the hardware color map. The following table show how virtual color map entries are initialized.

Virtual Color Map Entry	Color Value
0	Default window background color
1	Default window foreground color

All other entries are undefined.

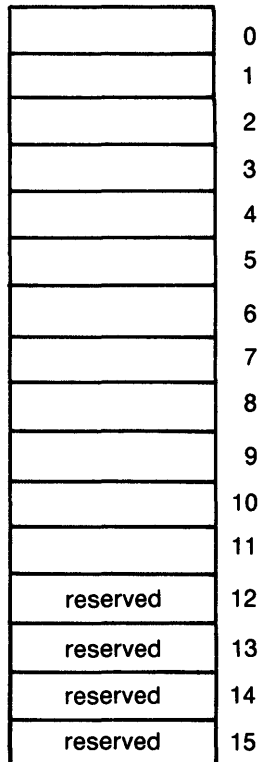
UIS transparently reconciles differences between the virtual color map model and the hardware color resources. UIS manages the concurrent use of these resources across applications.

For information about creating and using virtual color maps, see Chapter 16.

4.4.1 Reserved Hardware Color Map Entries

On mapped color systems, due to hardware limitations, the hardware color system or the UIS window management software preallocates some of the hardware color map entries for special purposes. For example, pointer colors, window background and foreground colors, and display screen color are allocated reserved entries in the hardware color map. Figure 4-7 describes reserved entries in a hardware color map in a four-plane system.

Figure 4-7 Reserved Hardware Color Map Entries in a 4-Plane Color System



ZK-5430-86

Whenever a virtual color map exceeds the size of the hardware color map less the reserved entries, the results are unpredictable. For more information about obtaining the hardware color map characteristics using the programming interface, see Chapter 16.

4.5 UIS Color Map Segments

The use of color map segments represents a device-specific binding of a virtual color map to the underlying hardware color resources, that is, the hardware color map. In a color mapped color system, color map segments are bound to specific hardware color map entries and swapped in and out of the hardware color map based on system and user events. Usually, applications need not worry about color map segments. UIS handles the device-specific binding automatically. Applications may want to use color map segments for the following reasons:

- Applications can control explicitly the binding of the virtual color map and the hardware color map.
- Applications are not transported to different hardware configurations, for example, four-plane to eight-plane systems or VAXstation color and intensity systems to VAXstation monochrome systems.

4.6 Shareable Virtual Color Maps

By default, virtual color maps are private. Yet, they may be shared among cooperating application programs to define a uniform color regime and to conserve hardware color map entries. Shared virtual color maps have names, an ASCII string from 1 to 15 characters and a name space (UIC group or system). For example, UIS uses a system-wide, shared color map to display terminal emulator windows and the window and screen menus.

4.7 Miscellaneous UIS Color Concepts

The following sections contain additional information about the UIS color subsystem.

4.7.1 Standard and Preferred Colors

VAXstation color and intensity systems supports two sets of symbolically defined colors. Workstation *standard* colors and intensity values are a set of colors used for specific purposes within the workstation environment. For example, the default window background and foreground, cursor background and foreground colors, and the display screen color are the workstation standard colors.

Workstation *preferred* colors are a set of colors representing the user's preference for the eight combinations of the RGB primary colors. For example, workstation preferred colors are used to define a particular shade of red, rather than a full intensity red. In an intensity system, preferred colors may be used to define a base white level from which preferred shades of gray are derived. Preferred values are

4-12 Color Concepts

simply a mechanism for conveniently maintaining and communicating a user's color preferences to an application.

Values for standard and preferred colors are set using the workstation setup mechanism. Standard and preferred color and intensity values can be returned using `UIS$GET_WS_COLOR` and `UIS$GET_WS_INTENSITY`.

4.7.2 Monochrome, Intensity, and Color Compatibility Features

Two types of calls are provided to change or retrieve color map entries. `UIS$SET_COLOR` and `UIS$SET_INTENSITY` both load a single color value in a color map entry. Both routines can be used in any of the three hardware color environments—monochrome, intensity, or color.

Color System	Compatibility Feature
Monochrome	UIS chooses the color (black or white) closest to the color specified by the application.
Intensity ¹	<code>UIS\$SET_COLOR</code> converts the specified RGB values to an equivalent gray level using an equation. <code>UIS\$SET_INTENSITY</code> sets the requested gray level directly.
Color ²	<code>UIS\$SET_COLOR</code> sets the requested RGB color values directly. <code>UIS\$SET_INTENSITY</code> converts the specified intensity value to an equivalent RGB value using an equation.

¹The color-to-intensity equation is $I = 0.30R + 0.59G + 0.11B$. Color television broadcasts transmitted for reception by noncolor television sets are processed in this manner.

²The intensity-to-color equation is $R = I, G = I, B = I$.

4.7.3 Color Value Conversion

Routines are provided to convert color values in applications using other color representation models.

- Hue lightness saturation (HLS)
- Hue saturation value (HSV)

In both models, hue values are specified from *0.0* to *360.0*, inclusive, where red = *0.0*. Values for lightness, saturation, and value are between *0.0* and *1.0*, inclusive.

4.7.4 Set Colors and Realized Colors

UIS routines that *set* or load color map entries in the virtual color map accept `F_floating` point values between *0.0* and *1.0*, inclusive. The precision of the `F_floating` point data type is approximately seven decimal places.

The precision for the color representation for a particular device may not be enough to represent accurately the requested `F_floating` point value.

In this case, the *set* color value (`F_floating`) differs from the *realized* color value (device precision). An application can determine realized color values using `UIS$GET_COLOR(S)` and including the optional parameter. See Chapter 16 for details.

4.7.5 Color Regeneration Characteristics

Color regeneration is a hardware characteristic that specifies whether changing a color map entry affects the color of existing graphic objects (retroactive regeneration) or only graphic objects drawn after the color map is changed (sequential regeneration).

The following table summarizes regeneration characteristics of direct and mapped color systems.

System	Regeneration Characteristics
Direct color	Usually sequential
Mapped color	Usually retroactive

An application can determine the hardware color regeneration characteristics by calling `UIS$GET_HW_INFO`.

Chapter 5

Input Devices

5.1 Overview

This chapter discusses the devices that enable user and application program interaction. Some of the topics covered in this chapter are:

- Pointing devices
- Virtual keyboards
- Physical keyboards

5.1.1 VAXstation Input Devices

Application programs and users interact through input devices. The types of input devices that a VAXstation typically utilizes are:

- Keyboard
- Mouse
- Tablet

The keyboard allows you to initiate program interaction and respond to application program prompts by pressing a key or entering data. The mouse and tablet let you communicate with an application program by pointing to objects or items with a *pointer* and by making selections with buttons.

5-2 Input Devices

5.2 Pointers

There are two types of pointing devices that can be used with the workstation, a mouse and a tablet. The workstation supports the use of only one pointing device at a time.

Application programs receive input from a pointing device by either polling or soliciting interrupts. To do this, programs use pointer input routines. Because only one pointer input device can be used at a time, applications use the same set of pointer input routines to get input from either the mouse or the tablet. The actual pointer input device being used is transparent to an application.

The programming interface lets you set the pattern or the position of the cursor that is synchronized with the pointing device.

5.2.1 Mouse

The mouse is a small hand-held device with three buttons on the top and a roller-ball on the bottom. Associated with the mouse, on the display screen, is an arrow-shaped cursor (or pointer).

The user is able to manipulate items on the display screen by the combined use of the mouse-controlled pointer and the mouse buttons. By moving the mouse in any direction on a flat surface, the ball on the bottom is turned, causing the pointer on the screen to move. In this way, the pointer can be moved in any direction and placed at any desired position on the display screen. By pressing the buttons on the mouse, the user can select items in a menu and perform a variety of other functions.

The mouse is a *relative pointing device*. The mouse reports only its relative movement to the workstation. The mouse can be picked up and placed in different position without any change in the position of the pointer on the screen. Consequently, the workstation keeps track of the current mouse position, only when the mouse is moved on a surface.

Some of the ways that application programs can use the pointer are as follows:

- To create menus from which the user selects items
- To read the position of the pointer and the state of the mouse buttons

The workstation human interface implements menus that allow users to create, select, move, and delete objects on the display screen. Application programs can create menus that do the same things. To select a menu item, the user moves the pointer to the region of the desired item and presses one of the mouse buttons. The application program predefines items and specifies the action to be taken when the user selects an item.

Application programs can detect when the pointer is moved across the boundary of a window or a mouse button is pressed within a window. Programs can also read the current pointer location and current button state. When the pointer is moved to the border, or outside, of a screen viewport, the human interface detects interrupts from the mouse. If the pointer is positioned inside of a viewport that is mapped to an application-created window, the application program can receive these interrupts.

5.2.2 Tablet

The tablet is an optional input device that can be used with the workstation. Tablets operate in much the same way as a mouse. An application program uses the same routines to receive information from a tablet as it does for the mouse. This is possible because the actual physical input device being used is transparent to an application program.

The tablet is an *absolute pointing device*. That is, it reports all movement to the workstation. For example, if the pen or stylus is picked up and moved to another position on the tablet, the pointer will change its position on the screen to match the movement.

A tablet is composed of the following parts:

- Tablet
- Puck
- Stylus

Tablet

The tablet is a flat square device with a surface similar to a table top. It is used in conjunction with a puck and/or stylus to locate points on the display screen. When the puck and/or stylus are moved on the surface of the tablet, the pointer on the display screen moves in an identical fashion. If you pick up the puck and place it in different region of the tablet, the pointer on the display screen would reflect this change. The tablet has a grid that senses a change in the position of the pen or stylus.

Puck

The puck is a hand-held device which is moved on the tablet to locate points on the display screen. The puck has cross-hair markings used for precision in positioning it on the tablet. It also has four buttons which can be used for various purposes, depending upon the application.

5-4 Input Devices

Stylus

The stylus is a hand held device which resembles a pen. It is moved on the tablet to locate points on the display screen. The stylus has greater precision than the puck in locating positions. The stylus can also have buttons, usually one is located on the outside of the barrel and one on the tip. The functions of these buttons are application specific.

5.3 Keyboards

It is important to be able to distinguish between a physical keyboard (the workstation keyboard) and a *virtual keyboard* (a simulated keyboard).

The physical keyboard is the actual workstation keyboard. You can press its keys to respond to prompts from the application program, or you can type and enter data into the currently active display window. The workstation can have only one physical keyboard attached to it at any one time.

A virtual keyboard is a conceptual keyboard that does not have an actual physical existence. Rather, a virtual keyboard is a simulated keyboard that exists in software and is associated with a display window. Each application may have one or more virtual keyboards attached to it. Virtual keyboards provide the means for applications to share the single physical keyboard.

5.3.1 Virtual Keyboards

A virtual keyboard is not an actual physical keyboard; but instead can be considered a simulated keyboard. Virtual keyboards are conceptual in nature and exist only in software. Virtual keyboards have much the same relationship to the physical keyboard as virtual displays have to the physical display screen.

Application programs can read from the physical (workstation) keyboard, assign the physical keyboard to a display window, and modify the characteristics of a physical keyboard associated with a window. Programs are able to do this by means of *virtual keyboard routines*. These routines can establish one or more virtual keyboards. They enable applications to manipulate the workstation keyboard by referring to the established virtual keyboards.

You can think of virtual keyboards in the following way. The VAXstation supports multiple windows with multiple processes running simultaneously. Normally, these windows and processes require keyboard input at various times. Therefore, each window may need to have a keyboard associated with it. Consequently, there is a need for several keyboards (one for each window). Because there is only one physical keyboard available, it must be shared among several windows. The way that this is done is through the concept of virtual keyboards.

Virtual keyboards provide a way for each window to have its own keyboard. There can be one, or several, display windows and virtual keyboards active on the display screen at one time. However, the physical keyboard can be connected to only one virtual keyboard at a time. A virtual keyboard can be attached to more than one display window at a time; however, each display window may have only one virtual keyboard attached to it.

The user has control over the association between the physical keyboard and the various virtual keyboards that exist at any point in time. A user can connect the workstation keyboard to different windows by manipulating the display viewports to which the virtual keyboards are connected. The user determines which window the workstation keyboard is attached to, and in that way, which process is receiving keyboard input. In this way, the user determines which window on the screen is currently active.

When the user switches the keyboard between windows, the workstation gives notification of which window has the keyboard. It places a small KB icon in the upper right corner of all windows that are able to use the keyboard. The KB icon is highlighted in the window that is currently active. An application can restrict windows from receiving keyboard input. Display windows that do not interact with the keyboard will not have the KB icon.

PART II How to Program with MicroVMS Workstation Graphics

Chapter 6

Programming Considerations

6.1 Overview

The User Interface Services (UIS) graphics software package allows you to create application programs that call system routines. Using UIS system routines, you can create virtual displays, display windows, viewports, graphic images, and text. These *callable* routines can be accessed through high-level programming languages as well as VAX MACRO and VAX BLISS. The programming examples used in succeeding chapters to illustrate the capabilities of the UIS graphics software are written in VAX FORTRAN. This chapter discusses the following topics:

- Calling UIS routines
- Argument characteristics
- Constants
- Condition values
- Additional program components
- Program execution

Refer to the *MicroVMS Programming Support Manual* for additional information about other callable routines.

6.2 Calling UIS Routines

Your application programs must contain references or calls to specific UIS system routines to draw and manipulate graphic images and text. These CALL statements and language-specific function declarations invoke the UIS system routines through the VAX Procedure Calling Standard.

6-2 Programming Considerations

6.2.1 Calling Sequences

The format of a call to UIS, or the *calling sequence*, consists of the elements that make up the statement and their positional order. Refer to Tables A-1 and B-1 in the appendices for summaries of UIS and UISDC calling sequences, respectively.

6.2.1.1 Call Type

Calls to UIS system routines from application programs, typically specify the function name and an argument list as follows:

```
vd_id=UIS$CREATE_DISPLAY(-1.0,-1.0,+1.0,+1.0,width,height)
```

However, some UIS routines are *functions* and return values to the calling program. The preceding example shows such a call from a VAX FORTRAN program. It also returns a value, the virtual display identifier, to the `vd_id` argument. Such return values are stored in variables that are often arguments (where applicable) in subsequent routine calls.

UIS routines that are not functions must be called using an explicit VAX FORTRAN CALL statement.

```
CALL UIS$PLOT(vd_id,1,-1.0,-1.0)
```

There is no standard call type used by all programming languages to invoke the UIS system routines. This manual does not attempt to describe the ways in which each high-level programming language calls a UIS system routine but uses VAX FORTRAN as an example of a typical call syntax. For specific information about calling syntax, please refer to the appropriate language user's guide.

6.2.1.2 Routine Name

You must identify the system routine you are calling by specifying its routine name, for example, `UIS$MOVE_AREA`. The routine name consists of a symbol prefix identifying the system facility (UIS\$) and the symbol name indicating what operation it performs (MOVE_AREA). The routine name is also known as the *entry point name*.

6.2.1.3 Argument List

The argument list is the list of parameters to be passed to the UIS routine. This list, typically, follows the function name as a parenthetical expression containing arguments separated by commas. You can substitute your own argument names in place of the formal parameter names. However, whenever you invoke a UIS routine, you must maintain the positional order of the parameters in the argument list. The following example illustrates positional order of the parameters:

```
CALL UIS$CIRCLE(VD_ID,ATB,CENTER_X,CENTER_Y,XRADIUS,START_DEG,END_DEG)
```

6.3 Argument Characteristics

Because the arguments in your routine call are the means of passing data to the called routine, you should keep in mind the characteristics of arguments—VMS Usage, type, access, mechanism.

6.3.1 VMS Usage

The *VMS Usage* entry contains the name of a VMS data type that has special meaning in the VMS operating system environment.

The VMS Usage entry is NOT a traditional data type such as the VAX standard data types byte, word, longword and so on. It is significant only within the context of the VMS operating system environment and is intended solely to expedite data declarations within application programs.

Refer to Appendix A in the *MicroVMS Workstation Version 3.0 Release Notes* for a complete listing of VMS usage entries and implementation charts for each VAX language supported by UIS. The implementation charts describe how to code the VMS usage entry in the programming language of your application.

6.3.2 Type

The *type* characteristic refers to the standard data type of the argument, that is, whether the argument is a word, longword, floating point number, and so forth. Depending on the programming language you are using, you may be required to declare certain data types locally within your program. These locally declared data structures provide data type definitions for the arguments in subsequent calls to UIS routines.

6.3.2.1 VAX Standard Data Types

When a calling program passes an argument to a system routine, the routine expects the argument to be of a particular data type. The routine descriptions in Part III indicate the expected data types for each argument.

Properly speaking, an argument does not have a data type; rather, the data specified by an argument has a data type. The argument is merely the vehicle for the passing of data to the called routine.

Nevertheless, the phrase “argument data type” is frequently used to describe the data type of the data that is specified by the argument. This terminology is used because it is simpler and more straightforward than the strictly accurate phrase “data type of the data specified by the argument.”

6-4 Programming Considerations

The following table contains the data types allowed by the VAX Procedure Calling Standard.

Table 6-1 VAX Standard Data Types

Data Type	Symbolic Code
Absolute date and time	DSC\$_K_DTYPE_ADT
Byte integer (signed)	DSC\$_K_DTYPE_B
Bound label value	DSC\$_K_DTYPE_BLV
Bound procedure value	DSC\$_K_DTYPE_BPV
Byte (unsigned)	DSC\$_K_DTYPE_BU
COBOL intermediate temporary	DSC\$_K_DTYPE_CIT
D_floating	DSC\$_K_DTYPE_D
D_floating complex	DSC\$_K_DTYPE_DC
Descriptor	DSC\$_K_DTYPE_DSC
F_floating	DSC\$_K_DTYPE_F
F_floating complex	DSC\$_K_DTYPE_FC
G_floating	DSC\$_K_DTYPE_G
G_floating complex	DSC\$_K_DTYPE_GC
H_floating	DSC\$_K_DTYPE_H
H_floating complex	DSC\$_K_DTYPE_HC
Longword integer (signed)	DSC\$_K_DTYPE_L
Longword (unsigned)	DSC\$_K_DTYPE_LU
Numeric string, left separate sign	DSC\$_K_DTYPE_NL
Numeric string, left overpunched sign	DSC\$_K_DTYPE_NLO
Numeric string, right separate sign	DSC\$_K_DTYPE_NR
Numeric string, right overpunched sign	DSC\$_K_DTYPE_NRO
Numeric string, unsigned	DSC\$_K_DTYPE_NU
Numeric string, zoned sign	DSC\$_K_DTYPE_NZ
Octaword integer (signed)	DSC\$_K_DTYPE_O
Octaword (unsigned)	DSC\$_K_DTYPE_OU
Packed decimal string	DSC\$_K_DTYPE_P
Quadword integer (signed)	DSC\$_K_DTYPE_Q
Quadword (unsigned)	DSC\$_K_DTYPE_QU
Character string	DSC\$_K_DTYPE_T
Aligned bit string	DSC\$_K_DTYPE_V
Varying character string	DSC\$_K_DTYPE_VT

Table 6-1 (Cont.) VAX Standard Data Types

Data Type	Symbolic Code
Unaligned bit string	DSC\$_K_DTYPE_VU
Word integer (signed)	DSC\$_K_DTYPE_W
Word (unsigned)	DSC\$_K_DTYPE_WU
Unspecified	DSC\$_K_DTYPE_Z
Procedure entry mask	DSC\$_K_DTYPE_ZEM
Sequence of instruction	DSC\$_K_DTYPE_ZI

Refer to the *MicroVMS Programming Support Manual* for more information about VAX standard data types.

6.3.3 Access

The *access* characteristic describes how a calling routine will use the data specified by the argument. Following is a list of the most common types of argument access:

- Read only access—the UIS routine uses the data specified by the argument as input only.
- Write only access—the UIS routine uses the argument as a location to return data only.
- Modify access—the UIS routine uses the data specified by the argument as input for its operation and then writes data to that argument.

6.3.4 Mechanism

VAX language extensions provide the means of reconciling the different argument passing mechanisms within a programming language. The VAX Procedure Calling Standard provides three ways by which all application programs may pass arguments to a system routine.

- By value—the argument contains the actual data to be used by the routine, the actual data is said to be passed to the routine by value.
- By reference—the argument contains the address of the location in memory of the actual data to be used by the routine, the actual data is said to be passed to the routine by reference.

6-6 Programming Considerations

- By descriptor—the argument contains the address of a descriptor, the actual data is said to be passed by descriptor.

A descriptor consists of two or more longwords (depending on the type of descriptor used) that describe the location, length, and data type of the data to be used by the called routine.

All language processors, except VAX MACRO and VAX BLISS, pass arguments by reference or descriptor by default. Some high-level languages including VAX FORTRAN set up the descriptors and arrays for you.

The following list contains the passing mechanisms allowed by the VAX Procedure Calling Standard.

Passing Mechanism	Descriptor Code
By value	
By reference	
By reference, array reference	
By descriptor	
By descriptor, fixed-length	DSC\$K_CLASS_S
By descriptor, dynamic string	DSC\$K_CLASS_D
By descriptor, array	DSC\$K_CLASS_A
By descriptor, procedure	DSC\$K_CLASS_P
By descriptor, decimal string	DSC\$K_CLASS_SD
By descriptor, noncontiguous array	DSC\$K_CLASS_NCA
By descriptor, varying string	DSC\$K_CLASS_VS
By descriptor, varying string array	DSC\$K_CLASS_VSA
By descriptor, unaligned bit string	DSC\$K_CLASS_UBS
By descriptor, unaligned bit array	DSC\$K_CLASS_UBA
By descriptor, string with bounds	DSC\$K_CLASS_SB
By descriptor, unaligned bit string 1 with bounds	DSC\$K_CLASS_UBSB

Refer to the *MicroVMS Programming Support Manual* for more information about passing mechanisms.

6.3.4.1 VAX FORTRAN Built-In Functions

VAX FORTRAN also supports explicit argument passing mechanisms, or *built-in* functions that do not require formal data declarations. Built-in functions are specified only in the argument list of the call (with one exception) and are used when data must be passed to a subroutine written in a programming language other than VAX FORTRAN. The four VAX FORTRAN built-in functions are as follows:

- %VAL—specifies that the argument must be passed as a value.
- %REF—specifies that the argument must be passed as the address of the actual data.
- %DESCR—specifies that the argument must be passed as the address of a descriptor that points to the actual data.
- %LOC—returns the virtual address of the actual data.

The built-in function %LOC can be used outside an argument list to obtain the address of a variable. For example, %LOC can be used in an assignment statement where a longword in a character string descriptor is assigned the address of the actual character string.

By default, VAX FORTRAN passes numeric data by reference and character string data by descriptor. The built-in functions override default argument passing mechanisms. Occasionally, an external procedure is encountered that passes data differently from the VAX FORTRAN default and, in such cases, the built-in functions can be used in VAX FORTRAN code.

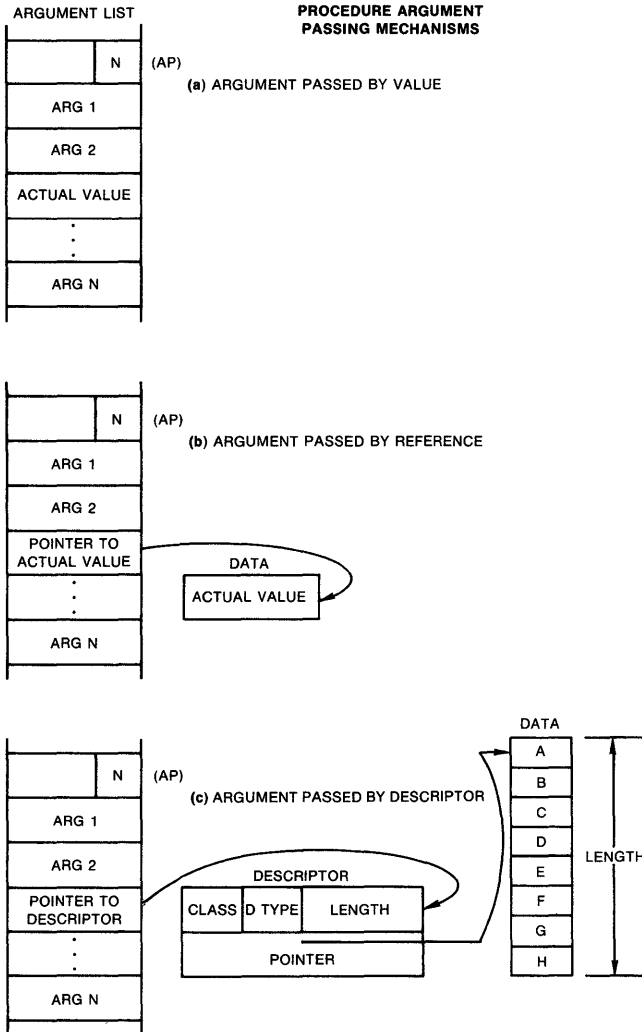
For specific information regarding similar procedure argument passing mechanisms for other high-level programming languages, refer to the appropriate language user's guide.

Figure 6-1 illustrates how arguments are placed on the stack and shows how arguments are passed to the called routine.

6-8 Programming Considerations

Figure 6-1 Passing Arguments

Procedure Argument Passing Mechanisms



Note. ARG 1, ARG 2, ARG N can be passed by value, by reference, or by descriptor in any of the above examples.

:(AP) = argument pointer

N = number of arguments

6.4 UIS Constants

UIS constants are symbolic names for values that can be passed to, or returned from, UIS routines. UIS constants are syntactically equivalent to literal integer constants and are used in the following ways:

- As arguments to UIS functions
- As indexes into array arguments that are passed to, or received from, the UIS subsystem
- As literals with which you can compare a returned value from an inquiry routine

Refer to Section 6.6 for information about UIS symbol definition files.

6.5 Condition Values Signaled

Occasionally hardware- or software-related events occur indicating errors that could jeopardize successful program execution. Instead of returning condition values to R0 (as in VAX MACRO) or to a status variable (as in high-level languages), the UIS routines signal a condition. In such cases, unless you have explicitly arranged to handle the signaled condition, program execution terminates.

6.6 Additional Program Components

In addition to the usual program entities, you should be aware of UIS-specific and language-specific program components that affect program execution.

Subroutines and Functions

VAX FORTRAN application programs must declare subroutines as external procedures with the EXTERNAL statement if the subroutine name is used as an actual argument to other subprograms. The subprogram can then use the corresponding dummy argument in a function reference or a CALL statement.

Entry Point and Symbol Definition Files

All UIS and UISDC routines are declared in an entry point file supplied with the graphics software. In addition, you may need to include a file of UIS symbol definitions depending on the language you are using. These files are also known as *data description* files. See your appropriate language user's manual to determine whether you must include these files in your program data declarations.

6-10 Programming Considerations

The following table contains a list of entry point files and symbol definition files for each VAX programming language. All files are located in SYS\$LIBRARY.

Table 6-2 Entry Point and Symbol Definition Files

VAX Language	Entry Point File	Symbol Definition File
BLISS	UIENTRY.R32	UISUSRDEF.R32
C	UIENTRY.H	UISUSRDEF.H
FORTRAN	UIENTRY.FOR	UISUSRDEF.FOR
MACRO		UISUSRDEF.MAR
PASCAL	UIENTRY.PAS	UISUSRDEF.PAS
PL/I	UIENTRY.PLI	UISUSRDEF.PLI

Message Definition File

A language-specific message definition file called UISMSG is included in the directory SYS\$LIBRARY. All possible UIS error codes are defined in this file. It is similar to the entry point file UIENTRY. For instance, to define message symbols in a VAX FORTRAN condition handler, you would add the following line to your program.

```
INCLUDE 'SYS$LIBRARY:UISMSG'
```

The appropriate language version of UISMSG is copied to your disk during the installation procedure depending on the programming language options you select.

All messages symbols use the prefix UIS\$....

6.7 Notes to Programmers

As a programmer, you should know about language-specific issues that might affect program execution. It is recommended that all application programmers read this section.

6.7.1 VAX C Programmers

Entry Point and Symbol Definition Files

The file UIENTRY.H defines all routine entry points in lowercase characters, while UISUSRDEF.H defines all constants in uppercase characters.

6.7.2 VAX PASCAL Programmers

Entry Point Files

Because VAX PASCAL references arguments as formal parameters, your calls to UIS must specify the same parameter names as those contained in the entry point file UISENTRY.PAS. Therefore, specify **obj_id** as the argument whenever the routine descriptions in Parts III and IV allow a choice between the **obj_id** and **seg_id** arguments. Refer to Tables A-1 and B-1 for summaries of UIS and UISDC calling sequences.

Creating Environment Files

Before running application programs written in VAX PASCAL, you must perform the following procedure.

1. Set your default directory to SYS\$LIBRARY.
`$ SET DEFAULT SYS$LIBRARY`
2. Produce an environment file of symbolic definitions and type declarations by invoking the VAX PASCAL compiler with the /ENVIRONMENT and /NOOBJECT qualifiers.

```
$ PASCAL/ENVIRONMENT/NOOBJECT UISENTRY
```

The result of the compilation is UISENTRY.PEN, an environment file.

3. Include the INHERIT attribute in the first line of the application program or program module specifying UISENTRY.PEN.
`[INHERIT('UISENTRY.PEN')]`
4. Repeat this procedure for the symbol definition file UISUSRDEF.PAS.

Refer to *Programming in VAX PASCAL* for more information about the /ENVIRONMENT and /NOOBJECT qualifiers and the INHERIT attribute.

Drawing Lines and Polygons

VAX PASCAL application programs should use UIS\$PLOT_ARRAY rather than UIS\$PLOT and UIS\$LINE_ARRAY instead of UIS\$LINE, when drawing lines and polygons.

6-12 Programming Considerations

6.7.3 VAX PL/I Programmers

Entry Point Files

Because VAX PL/I references arguments as formal parameters, your calls to UIS must specify the same parameter names as those contained in the entry point file UISENTRY.PLI. Therefore, specify **obj_id** as the argument whenever the routine descriptions in Parts III and IV allow a choice between the **obj_id** and **seg_id** arguments. Refer to Tables A-1 and B-1 for summaries of UIS and UISDC calling sequences.

6.8 Programming Examples

The programming examples in Parts II and III of this manual use VAX FORTRAN Version 4.4. In addition, some examples particularly in Part III include ellipses, the standard convention for indicating portions of code that have been left out. The ellipses are also included to point out places in the program where code could be added at the programmer's discretion.

Many of the examples include the VAX FORTRAN PAUSE statement. The PAUSE suspends program execution and returns the user to the DCL prompt (\$). A default message "FORTRAN PAUSE" is returned to the display screen. The graphic images that were created on the display screen will remain. You can respond to the DCL prompt (\$) by typing one of the following commands:

- CONTINUE—Program execution resumes at the next executable statement.
- EXIT—Program execution is terminated.
- DEBUG—Program execution resumes under the control of the VAX/VMS Symbolic Debugger.

NOTE: If your program is running in batch mode, program execution is not suspended. All messages are written to the system output file.

6.8.1 Structure of Programming Tutorial

Part II attempts to describe UIS graphics features and programming using a tutorial approach in each chapter. Within each chapter, after a discussion of the main topics, you are offered two types of information under the following headings:

- Programming options — Lists the features that you can use at a given point in time. The addition of each new group of programming options lets you progress in an orderly fashion from simple programming tasks to relatively complex ones.

- Program development — Lists the current programming objective and the tasks needed to successfully implement the objective.

Program — Contains the source module with embedded callouts. Each callout refers to a programming feature that should be noted.

Program output — Displays and explains the output from the program.

Each programming example uses some or all of the programming options listed. Not all routines are illustrated in the accompanying example.

6.9 Program Execution

Your program can run in batch mode with predefined data or it can run interactively accepting input from you when needed. However, in order to execute your application program successfully, you must first store it as a file using a text editor. Invoke the text editor on your workstation using the following command sequence. Please refer to appropriate sections of the user's manual for detailed information about MicroVMS text editors.

```
$ EDIT MYPROG.FOR
```

Please note in the previous example that you must supply a file name, for example, MYPROG. In addition, a VAX FORTRAN file type (FOR) is added to the file name to identify the file as a VAX FORTRAN source file. Enter your program according to the rules of the programming language you are using. Refer to the appropriate language reference manual for detailed information about the language.

6.9.1 Compiling Your Program

The newly created source file MYPROG.FOR must be compiled prior to execution. The language compiler, in our case the VAX FORTRAN compiler, checks for proper syntax and initiates code optimization where appropriate. Invoke the language compiler in the following manner.

```
$ FORTRAN/LIST MYPROG
```

Note that the file type need not be included. By default, the system searches for the latest version of the file, MYPROG, with a file type of FOR. If the application source file contains syntax errors, you will receive *compile-time* error messages called *diagnostics*. These diagnostic messages indicate the portion of code in error as well as an explanation. The /LIST qualifier specifies the creation of a listing file of accounting information and diagnostics (if present).

Some language compilers return a predetermined maximum number of diagnostics before terminating compilation. In any case, you must correct these errors and resubmit the source program for a successful compilation. Successful compilation produces an object module with file type of OBJ.

6-14 Programming Considerations

6.9.2 Linking the Object Module

The Linker resolves references to subroutines and allocates memory to variables within your program. Invoke the Linker in the following manner:

```
$ LINK MYPROG
```

You need not specify the file type of the program, MYPROG. By default, the system searches for the latest version of the file MYPROG with the file type OBJ.

In addition, you can link object modules of programs written in different source code.

6.9.3 Running the Executable Image

The Linker produces an executable image with a file type of EXE. At this point, you can run your program. However, you may receive run-time errors in which case you must correct errors in your source code and recompile the source module and relink the object modules. Run the executable image after receiving the \$ prompt in the following manner:

```
$ RUN MYPROG
```

Chapter 7

Creating Basic Graphic Objects

7.1 Overview

This chapter describes how to create basic graphic objects: lines, circles, ellipses, and text. To accomplish this task you will need to know about the following topics:

- Creating a virtual display
- Creating graphics and text
- Creating a display window

You will construct an interactive program that contains the necessary components for creating graphic objects. Later, you will manipulate these displays using other windowing routines.

Refer to Section 6.8 for more information about the programming examples that appear in this manual.

7.2 Step 1—Creating a Virtual Display

When an artist paints a picture, he is concerned with presenting a subject from a particular perspective. He then wonders how he will frame his subject and how much space he will need to accomplish this task successfully. These needs are fulfilled by the size of the canvas he chooses. All of the objects that we will create will use such a frame of reference or *virtual display* to establish the universe in which our graphic objects will exist.

While the artist simply chooses a spot on the canvas to paint, our calls to UIS routines must reference points within our virtual display. The UIS subsystem uses the coordinates you specify to generate a coordinate system with which we can create the virtual display and subsequent windows. This coordinate system, or grid, allows us to reference points as *world coordinates* along two perpendicular axes labelled *x* and *y*.

7-2 Creating Basic Graphic Objects

Unlike the artist's canvas which has finite dimensions, your virtual display is infinite and graphic objects may be drawn anywhere in it.

7.2.1 Specifying Coordinate Values

Many routines documented in this manual require specifying coordinates to define virtual displays, display windows, and extent rectangles. Table 7-1 lists information about coordinate values.

Table 7-1 Types of Coordinates

Coordinate	Units	Data Type	Origin
Absolute	cm	F_floating ¹	Lower-left corner of display screen or tablet
Normalized	Gutenbergs	F_floating ¹	Lower-left corner of virtual display
Viewport-relative	Pixels	Longword (unsigned)	Lower-left corner of display viewport
World	User-specified	F_floating ¹	Lower-left corner of virtual display

¹F_floating point numbers may be expressed with up to approximately seven decimal digits of precision.

7.2.2 Programming Options

The following options allow you to create the basic structures used to create graphic objects.

Creating a Virtual Display

You must use `UIS$CREATE_DISPLAY` to specify the world coordinate space in which you will draw graphic objects. The world coordinate values specified in `UIS$CREATE_DISPLAY` establish mapping and scaling factors that the system may later use in viewport creation. The coordinate values should not be thought of as the absolute boundaries of the virtual display.

You can create an unlimited number of virtual displays subject to system and process resources.

Deleting a Virtual Display

You may delete a virtual display at any time in your program using `UIS$DELETE_DISPLAY`. However, you should remember that when you delete a virtual display you are throwing out the canvas on which you have drawn graphic objects.

7.2.3 Program Development

Programming Objectives

To create an executable program using the VAX FORTRAN programming language.

Programming Tasks

1. Create a virtual display.
2. Delete the virtual display.

```

PROGRAM IMAGES_1
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY' ❶
INCLUDE 'SYS$LIBRARY:UISUSRDEF' ❷
.
.
.
VD_ID=UIS$CREATE_DISPLAY(+1.0,+1.0,+20.0,+20.0,10.0,10.0) ❸
.
.
.
PAUSE ❹
CALL UIS$DELETE_DISPLAY(VD_ID) ❺
END

```

At this point the program contains UIS entry points ❶ and definitions ❷. It also includes a call ❸ to UIS\$CREATE_DISPLAY. The plus sign (+) is optional for positive coordinates. The minus sign (-) is required for negative coordinates.

Because world coordinates are *f*-floating numbers, the decimal point is required when specifying world coordinate pairs.

See Section 6.8 for information about the VAX FORTRAN PAUSE statement ❹.

UIS\$DELETE_DISPLAY is called ❺ to remove the virtual display before the program ends. Terminating an application program with UIS\$DELETE_DISPLAY is not required.

Besides specifying the world coordinate range of the virtual display, UIS\$CREATE_DISPLAY returns the value of the virtual display identifier in *vd_id*. The virtual display ID uniquely identifies this newly created virtual display and is used in subsequent windowing routines. Typically, UIS\$CREATE_DISPLAY is the first UIS routine to be called in an application program.

If your application program were to invoke the UIS\$CREATE_DISPLAY only, you would not notice a change in your workstation display screen.

7-4 Creating Basic Graphic Objects

7.3 Step 2—Creating Graphics and Text

You are now at a point comparable to the artist preparing to draw on the canvas. The virtual display is an infinitely large canvas. You must choose the types of graphic objects to be drawn there. You can draw graphic objects anywhere in the virtual display. Three types of graphic objects can be drawn in the virtual display as shown in the following table.

Graphic Object	Example
Geometric shapes	Point, line, polygon, circle, and ellipse
Text	Characters
Raster images	Any object constructed with a bitmap of varying size

7.3.1 Graphics Drawing Operations

The following considerations apply to graphics operations:

- All line drawing operations are symmetrical and include both end points.
- All region specifications include the borders of the region specified. This applies in all cases to fill patterns, images, ellipses, moving windows, and so forth.

7.3.2 Programming Options

You can draw any of the graphic objects listed in this section. Read the routine description of each routine carefully.

Creating Points, Lines, and Polygons

Depending on the number of times you repeat coordinate pairs in `UIS$PLOT` or `UIS$PLOT_ARRAY`, you can draw a point, connected lines, or a polygon.

You can draw more than one unconnected line in single call to `UIS$LINE` or `UIS$LINE_ARRAY`. Each *pair* of world coordinate pairs specified represents the end points of a line.

NOTE: VAX PASCAL application programs should use `UIS$PLOT_ARRAY` or `UIS$LINE_ARRAY` to draw all lines, disconnected lines, and polygons.

Creating Circles

You can create circles or circular arcs with `UIS$CIRCLE`.

Creating Ellipses

You can create ellipses or elliptical arcs with `UIS$ELLIPSE`.

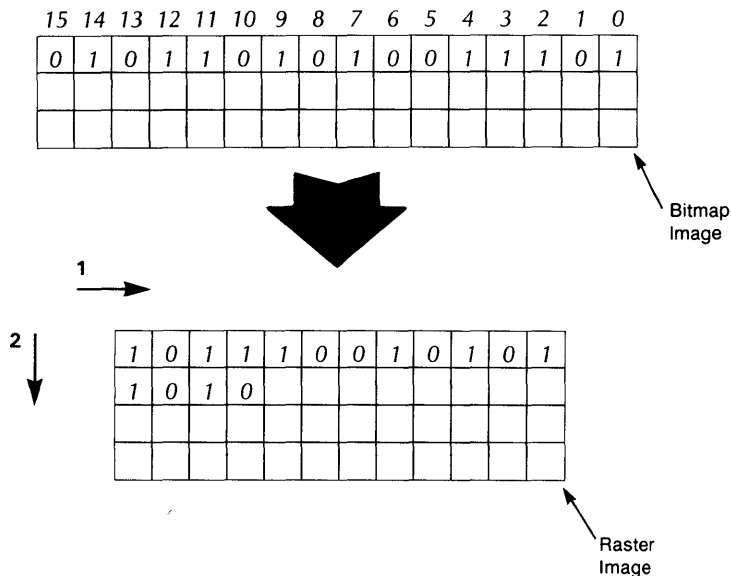
Drawing Images

You can create a bitmap image of a graphic object and then draw the raster to the display screen with UIS\$IMAGE using the following procedure:

1. Create a data structure in your program, such as an array or record, that defines the bitmap.
2. Set the bits in the structure to create the bitmap image by assigning values to the elements of the structure.
3. Specify width and height of the raster image in pixels in UIS\$IMAGE.
4. Specify the name of the data structure in UIS\$IMAGE.

Figure 7-1 illustrates how bitmap settings are mapped to raster images.

Figure 7-1 Mapping a Bitmap to a Raster



ZK 4627 85

Mapping the raster image occurs from left to right and from top to bottom. See the UIS\$IMAGE routine description for more information.

Text

You can set the current position and create text anywhere within a virtual display using UIS\$TEXT. The text within a virtual display could be used for labelling an accompanying graphic object within the window. Only UIS\$TEXT can write characters in a virtual display.

7-6 Creating Basic Graphic Objects

7.3.3 Program Development

Programming Objectives

To create an executable program using the VAX FORTRAN programming language.

Programming Tasks

1. Create a virtual display.
2. Draw four graphic objects in the virtual display.
3. Delete the virtual display.

```
PROGRAM IMAGES_2
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,10.0,10.0)

CALL UIS$CIRCLE(VD_ID,0,10.0,10.0,1.0) ①
CALL UIS$PLOT(VD_ID,0,4.0,3.0,5.0,7.0) ②
CALL UIS$ELLIPSE(VD_ID,0,15.0,15.0,1.0,2.0) ③
CALL UIS$TEXT(VD_ID,0,'This is a test.',1.0,12.0) ④
.
.
.

PAUSE
CALL UIS$DELETE_DISPLAY(VD_ID)
END
```

In the preceding example, world coordinate pairs are specified explicitly to the UIS graphics routines ① ② ③ ④ describing the exact locations of the graphic objects (circle, line, ellipse, and text) in the virtual display.

If you executed the program in its present form, the workstation display screen would show no objects. Your calls to the UIS graphics and text routines have been processed. However, you must create a window to view what has been drawn.

7.4 Step 3—Creating a Display Window

The next step is to create a display window. The display window defines the world coordinate range of the viewable portion of the virtual display. When you create a display window, you are also creating a display viewport, an area on the physical screen on which the display window is mapped.

7.4.1 Programming Options

All the programming options available to us at this point, are provided through `UIS$CREATE_WINDOW`. At this point, you do not need to know about its full capabilities, which are discussed in more detail in the next chapter.

Creating a Display Window and Viewport

You can create a display viewport and its associated viewport with `UIS$CREATE_WINDOW`.

7.4.2 Program Development

Programming Objectives

To create an executable program that draws and displays graphic objects on the VAXstation display screen.

Programming Tasks

1. Create a virtual display.
2. Draw four graphic objects in the virtual display.
3. Create a display window and viewport.
4. Delete the virtual display.

```

PROGRAM IMAGES_2A
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL*4 WIDTH,HEIGHT

TYPE *,'ENTER DESIRED VIEWPORT WIDTH AND HEIGHT'
ACCEPT *,WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,WIDTH,HEIGHT) ❶

CALL UIS$CIRCLE(VD_ID,0,10.0,10.0,1.0) ❷
CALL UIS$PLOT(VD_ID,0,4.0,3.0,5.0,7.0) ❸
CALL UIS$ELLIPSE(VD_ID,0,15.0,15.0,1.0,2.0) ❹
CALL UIS$TEXT(VD_ID,0,'This is a test.',1.0,12.0) ❺

WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION') ❻

PAUSE

CALL UIS$DELETE_DISPLAY(VD_ID)

END

```

7-8 Creating Basic Graphic Objects

The world coordinate range of the virtual display and the default dimensions of the display viewport are specified in a call to `UIS$CREATE_DISPLAY` ①.

NOTE: The display viewport will not be mapped until a display window is created.

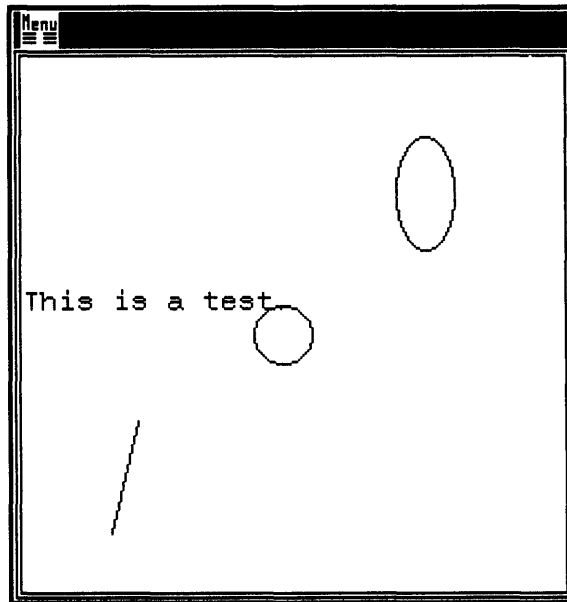
Next, the graphics and text routines are called ② ③ ④ ⑤ to draw the graphic objects.

A display window and viewport are created in a call to `UIS$CREATE_WINDOW` ⑥. The world coordinate range of the window and the viewport width and height are not specified. Therefore, the world coordinate space of the display window, that is, the viewable portion of the virtual display defaults to the entire virtual display. You will see all objects drawn in the virtual display.

7.4.3 Calling `UIS$CIRCLE`, `UIS$ELLIPSE`, `UIS$PLOT`, `UIS$TEXT`, and `UIS$CREATE_WINDOW`

When you run the program `IMAGES_2A`, you should get a single display viewport without a title, containing text, a circle, a line, and an ellipse as shown in Figure 7-2.

Figure 7-2 Display Viewport and Graphic Objects



Chapter 8

Display Windows and Viewports

8.1 Overview

Before you begin to manipulate graphic objects, you need to know more about display windows and viewports. After all, display windows and viewports allow you to see graphic objects drawn in the virtual display. This chapter discusses the following topics:

- Creating display windows and viewports
- Moving display windows
- Manipulating display viewports
- Deleting display windows
- Erasing the virtual display
- Creating transformations

These tasks are accomplished by the UIS *windowing* routines.

8.2 Windowing Routines

Windowing routines are responsible for the creation and deletion of virtual displays, display windows, and display viewports. Table 8–1 provides a list of window routines and their functions.

8-2 Display Windows and Viewports

Table 8-1 UIS Windowing Routines

Routine	Description
UIS\$CREATE__DISPLAY	Creates a virtual display and defines default viewport dimensions
UIS\$CREATE__WINDOW	Creates display window and viewport
UIS\$EXPAND__ICON	Substitutes an associated viewport for an icon
UIS\$MOVE__AREA	Moves a specified rectangle and its contents in the virtual display to another part of the virtual display
UIS\$MOVE__WINDOW	Pans the display window across the virtual display
UIS\$POP__VIEWPORT	Allows an occluded viewport to be fully displayed
UIS\$PUSH__VIEWPORT	Places a viewport behind another viewport
UIS\$SHRINK__TO__ICON	Substitutes an icon for a display viewport
UIS\$CREATE__TRANSFORMATION	Alters the world coordinate space of the virtual display
UIS\$ERASE	Erases objects that lie completely within a specified rectangle in the virtual display
UIS\$DELETE__DISPLAY	Deletes a virtual display
UIS\$DELETE__WINDOW	Deletes a display window and viewport

These routines allow you to create and manage the display screen environment and to perform certain housekeeping functions such as erasing and deleting virtual displays and windows.

8.3 Step 1—Creating Many Display Windows

For every display window that you create, you are also creating a display viewport. A one-to-one relationship exists between each display window and its associated viewport. An application program can create an unlimited number of display windows and viewports subject to system and process resources.

8.3.1 Programming Options

Each display window can be unique with regard to world coordinate range. Therefore, you can create display viewports that are also unique with respect to dimensions and position in the display screen.

Display Window Size

By default, a newly created display window displays the full world coordinate space specified when creating the virtual display. You can specify world coordinates pairs in `UIS$CREATE_WINDOW` as you see fit to produce display windows of different proportions within the virtual display.

Display Viewport Size

Similarly, the default display viewport dimensions are equal to the values specified in the **width** and **height** arguments in the `UIS$CREATE_DISPLAY` call. However, you may specify different dimensions to scale the contents of the window. Maximum display viewport size depends on the dimensions of the display screen. If you specify viewport dimensions that exceed the size of the display screen, `UIS` scales the viewport to the size of the display screen.

Graphic Object Magnification

The world coordinate range of the display window or the dimensions of the display viewport can be manipulated to increase or decrease magnification of the object displayed in the viewport. This occurs when the display window area is decreased or increased while the viewport size remains the same, or when the viewport is decreased or increased while dimensions of the window remain the same.

Distortion

Distortion occurs whenever the aspect ratios of the display viewport and display window are not equal. The aspect ratio of the display window is the absolute value of the difference between y world coordinates of the upper-right and the lower-right corners of the window divided by the absolute value of the difference between the x world coordinates of the lower-right and lower-left corners. Figure 8-1 illustrates how to calculate the aspect ratios of the display window and viewport.

Figure 8-1 Aspect Ratios of the Display Window and Display Viewport

$$\frac{|y_1 - y_0|}{|x_1 - x_0|} = \frac{\text{viewport height}}{\text{viewport width}}$$

ZK-4579-85

Number of Windows and Viewports

You can create an unlimited number of display windows and, as a result, an unlimited number of display viewports subject to system and process resources. In addition, you can specify the dimensions of each display viewport.

8-4 Display Windows and Viewports

Display Banner

The display banner appears along the top border of the display viewport and contains the menu and keyboard icons as well as the viewport title. The maximum length of the viewport title is 63 characters.

You can suppress the generation of the display banner with the **attributes** argument in `UIS$CREATE_WINDOW`. When the display banner is suppressed, only the viewport border is displayed.

Display Viewport Placement

You can either explicitly place a display viewport on the workstation display screen or you can allow UIS to choose a location for you. By default, display viewport placement is random.

8.3.2 Program Development

Programming Objectives

To create four display windows and display viewports.

Programming Tasks

1. Create a virtual display.
2. Draw four graphic objects in the virtual display.
3. Create four display windows and viewports omitting the display window coordinates in the calls to `UIS$CREATE_WINDOW`.
4. Delete the virtual display.

```
PROGRAM IMAGES_3
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,10.0,10.0)

CALL UIS$CIRCLE(VD_ID,0,10.0,10.0,1.0)
CALL UIS$PLOT(VD_ID,0,4.0,3.0,5.0,7.0)
CALL UIS$ELLIPSE(VD_ID,0,15.0,15.0,1.0,2.0)
CALL UIS$TEXT(VD_ID,0,'This is a test.',1.0,12.0)

WD_ID1=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION') ❶
PAUSE
WD_ID2=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION') ❷
WD_ID3=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION') ❸
WD_ID4=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION') ❹
```

```
PAUSE  
CALL UIS$DELETE_DISPLAY(VD_ID)  
END
```

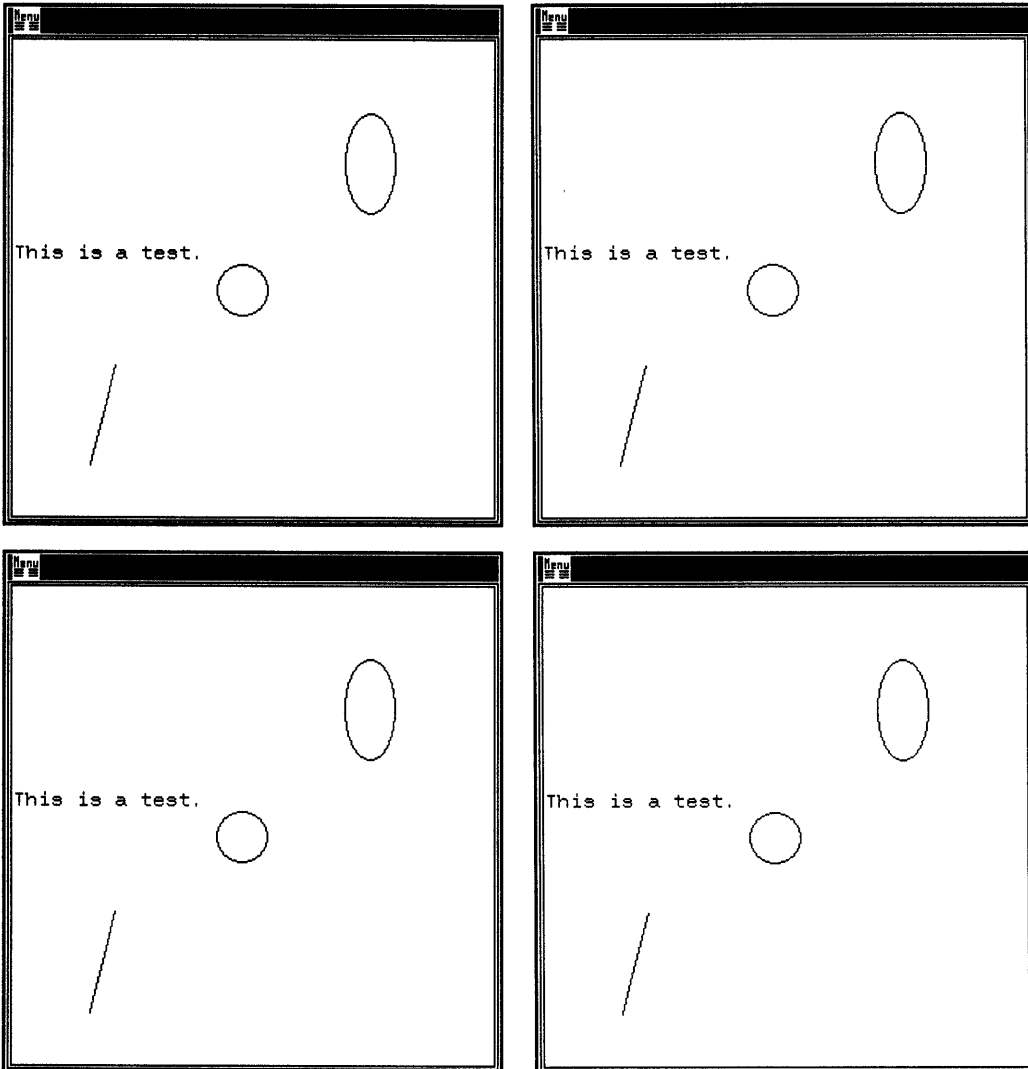
Four calls to `UIS$CREATE_WINDOW` ❶ ❷ ❸ ❹ have been inserted to create four windows. The world coordinate range of each window defaults to the world coordinate range of the entire virtual display.

8.3.3 Calling `UIS$CREATE_WINDOW`

If you were to run this program now, your workstation screen would display the graphic objects as shown in Figure 8-2.

8-6 Display Windows and Viewports

Figure 8-2 Four Display Viewports



ZK-4534-85

As you can see, four display windows have been created and mapped to the display screen as four viewports. Each of the viewports contains four objects. Because display window world coordinate pairs were not explicitly specified in `UIS$CREATE_WINDOW`, the viewports allow you to see the entire area of the

virtual display by default. In addition, because the display viewport width and height in centimeters were not explicitly specified in the `UIS$CREATE_WINDOW` call, each display viewport is, by default, 10 cm square as specified in the `width` and `height` arguments of the `UIS$CREATE_DISPLAY` call.

8.4 Step 2—Deleting and Erasing Display Windows

Some windowing routines perform housekeeping functions, that is, they delete unused display windows or erase graphic objects from the virtual displays. Such routines are important in managing display environment, when you run complicated applications.

8.4.1 Programming Options

You may want your application program to delete unwanted windows, viewports, and virtual displays. This can be done by calling UIS routines for deleting and erasing display windows and virtual displays.

Display Window Deletion

Any display window can be deleted without interfering with other windows or viewports. Deletion of the display window does not affect the graphic objects in the virtual display. If you delete a display window, you are also deleting the associated display viewport. Delete any display window and its associated viewport by specifying the appropriate display window identifier in `UIS$DELETE_WINDOW`.

Erasing the Virtual Display

Graphic objects that lie completely within a specified rectangle in the virtual display can be deleted at any time using `UIS$ERASE`. If no rectangle is specified, the entire virtual display is used.

8.4.2 Program Development

Programming Objectives

To enclose each graphic object in its own display window and then to delete a window and its viewport.

8-8 Display Windows and Viewports

Programming Tasks

1. Create a virtual display.
2. Draw four graphic objects in the virtual display.
3. Create four display windows and viewports specifying display window regions that enclose each of the graphic objects.
 - Specify display window regions that enclose each of the graphic objects.
 - Specify a viewport title identifying the graphic object.
4. Delete one of the display windows and its viewport.

```
PROGRAM IMAGES_4
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL WIDTH,HEIGHT

TYPE *,'ENTER DISPLAY SIZE' ❶
ACCEPT *,WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,WIDTH,HEIGHT)

CALL UIS$CIRCLE(VD_ID,0,12.0,12.0,1.0)
CALL UIS$PLOT(VD_ID,0,4.0,3.0,5.0,7.0)
CALL UIS$ELLIPSE(VD_ID,0,15.0,15.0,1.0,2.0)
CALL UIS$TEXT(VD_ID,0,'This is a test.',1.0,12.0)

WD_ID1=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','CIRCLE',
2      10.0,10.0,14.0,14.0,WIDTH,HEIGHT) ❷
WD_ID2=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','LINE',
2      3.0,2.0,6.0,8.0,WIDTH,HEIGHT) ❸
WD_ID3=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','TEXT',
2      1.0,12.0,10.0,10.0,WIDTH,HEIGHT) ❹
WD_ID4=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','ELLIPSE',
2      13.0,13.0,17.0,18.0,WIDTH,HEIGHT) ❺

PAUSE

CALL UIS$DELETE_WINDOW(WD_ID2) ❻

PAUSE

END
```

The program now accepts input for the display viewport dimensions interactively ❶.

The world coordinate space that defines each display window is specified explicitly in the UIS\$CREATE_WINDOW calls ❷ ❸ ❹ ❺.

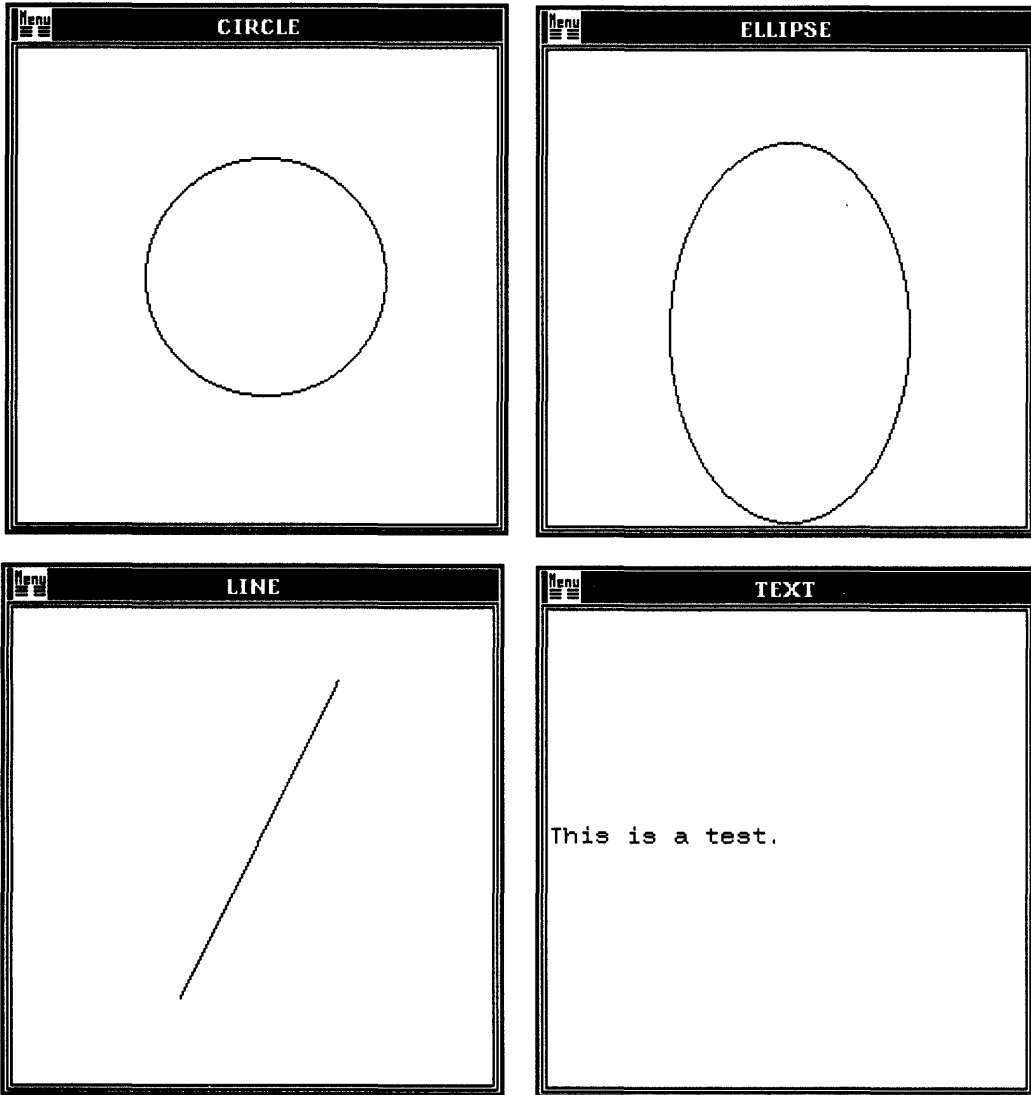
UIS\$CREATE_WINDOW returns the variable *wd_id2*, the display window identifier ❸ to uniquely identify the LINE window. Note that the call to delete the LINE window ❹ references this variable.

8.4.3 Calling UIS\$DELETE_WINDOW

If we ran this program until the first PAUSE statement, the workstation screen would display the graphic objects shown in Figure 8-3.

8-10 Display Windows and Viewports

Figure 8-3 Objects Within Different Windows

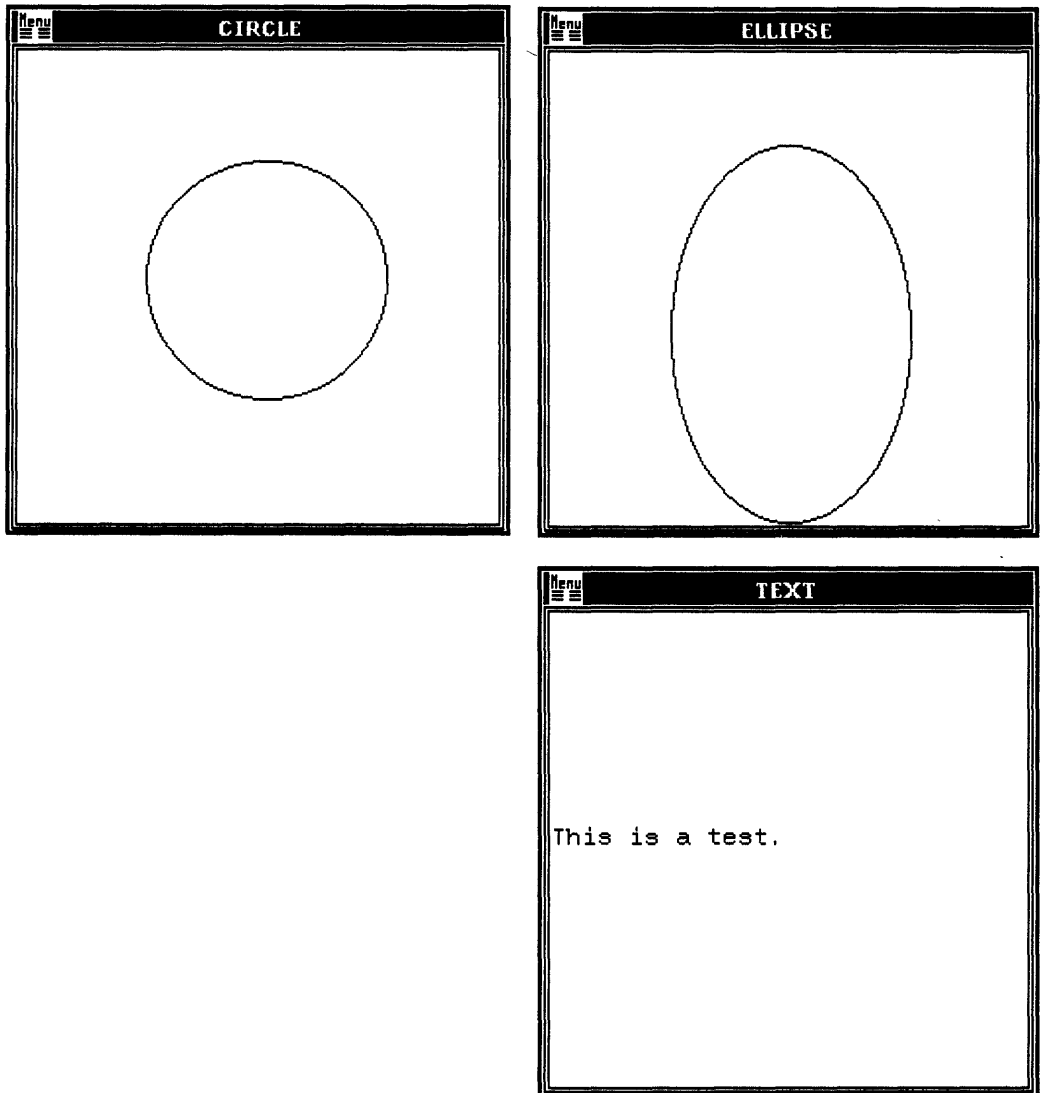


ZK-4535-85

By specifying explicitly a particular world coordinate range within the virtual display for each display window, each graphic object lies within a separate window that maps to the physical display screen as a separate display viewport.

To continue program execution, type `CONTINUE` at DCL prompt (`$`). The program continues to execute and the screen changes as shown in Figure 8-4.

Figure 8-4 Display Window Deletion



ZK-4536-85

The viewport `LINE` and its window are deleted. However, the actual graphic object still exists. You have simply deleted the display window that allowed you

8-12 Display Windows and Viewports

to view the portion of the virtual display that contained the line. If you called `UIS$CREATE_WINDOW` again, specifying the appropriate world coordinate space in the virtual display, the object would reappear.

8.5 Step 3—Manipulating Display Windows and Viewports

Display viewports and windows do not have to remain as static objects on your screen. You can manipulate the newly created display windows and viewports in many ways.

8.5.1 Programming Options

Viewport placement features and window attributes are implemented using the optional **attributes** argument of `UIS$CREATE_WINDOW`.

NOTE: When you include the **attributes** argument in `UIS$CREATE_WINDOW`, you are not modifying attribute block 0.

Attributes and attribute block 0 are discussed in detail in the next chapter.

General and Exact Placement of Viewports

Unless you specify otherwise, your display viewports are placed randomly throughout the screen. You can move any display viewport to any position on the screen. When you create the window, you can specify general viewport placement, that is, within a certain vicinity on the screen—top, left, right, or bottom.

Exact placement positions your display viewport where you want on the screen and allows you to occlude other viewports to save space.

Panning and Zooming the Virtual Display

You can pan across the virtual display to include either the entire virtual display or any discrete area within it.

Pushing and Popping Display Viewports

Pushing and popping display viewports is useful when you have created display windows with the exact placement attribute. In such a case, your application may have created two windows and purposely occluded one of the viewports. In this instance, you know which viewport will be occluded and the use of `UIS$POP_VIEWPORT` is clearly indicated.

Otherwise, the UIS subsystem places newly created windows randomly on the display screen by default. As a result, you will not know where the viewports will be placed. Therefore, use of `UIS$POP_VIEWPORT` or `UIS$PUSH_VIEWPORT` in this instance, would be unnecessary and confusing.

Moving a Display Viewport

You can move an existing display viewport anywhere on the display screen using `UIS$MOVE_VIEWPORT`.

Moving a Portion of the Virtual Display

You can draw a graphic object in a portion of the virtual display, then move that coordinate space to another part of the same virtual display with `UIS$MOVE_AREA`. The vacated source area is filled with the current background color.

8.5.2 Program Development I**Programming Objectives**

To delete three display windows and viewports and then to pan the virtual display using the remaining display window.

Programming Tasks

1. Create a virtual display.
2. Draw four graphic objects in the virtual display.
3. Create four display windows and viewports each containing a graphic object.
4. Specify a title for each viewport.
5. Delete three of the four display windows.
6. Pan the virtual display with the remaining display window using `UIS$MOVE_WINDOW`.

```

PROGRAM IMAGES_5
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL WIDTH,HEIGHT

TYPE *,'ENTER VIEWPORT WIDTH AND HEIGHT'
ACCEPT *,WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,10.0,10.0)

CALL UIS$CIRCLE(VD_ID,0,12.0,12.0,1.0) ①
CALL UIS$PLOT(VD_ID,0,4.0,3.0,5.0,7.0) ②
CALL UIS$ELLIPSE(VD_ID,0,15.0,15.0,1.0,2.0) ③
CALL UIS$TEXT(VD_ID,0,'This is a test.',1.0,12.0) ④

```

8-14 Display Windows and Viewports

```
WD_ID1=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'CIRCLE',
2      10.0,10.0,14.0,14.0,WIDTH,HEIGHT) ⑤
WD_ID2=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'LINE',
2      3.0,2.0,6.0,8.0,WIDTH,HEIGHT) ⑥
WD_ID3=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'TEXT',
2      1.0,12.0,10.0,10.0,WIDTH,HEIGHT) ⑦
WD_ID4=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'ELLIPSE',
2      13.0,13.0,17.0,18.0,WIDTH,HEIGHT) ⑧

PAUSE ⑨

CALL UIS$DELETE_WINDOW(WD_ID1) ⑩
CALL UIS$DELETE_WINDOW(WD_ID3) ⑪
CALL UIS$DELETE_WINDOW(WD_ID4) ⑫

PAUSE ⑬

CALL UIS$MOVE_WINDOW(VD_ID,WD_ID2,6.0,8.0,18.0,18.0) ⑭

PAUSE ⑮

CALL UIS$DELETE_DISPLAY(VD_ID)

END
```

The program IMAGE_5 creates four graphic objects ① ② ③ ④ in the virtual display.

The program prompts for the viewport width and height which overrides the values specified in UIS\$CREATE_DISPLAY.

Each graphic object is contained within a newly created display window ⑤ ⑥ ⑦ ⑧. Each display window is mapped to the physical screen as a display viewport with an appropriate title describing the graphic object within the window.

Program execution is suspended ⑨. The display screen contains four viewports as previously described.

Three calls to UIS\$DELETE_WINDOW ⑩ ⑪ ⑫ remove the windows and their viewports CIRCLE, ELLIPSE, and TEXT from the display screen.

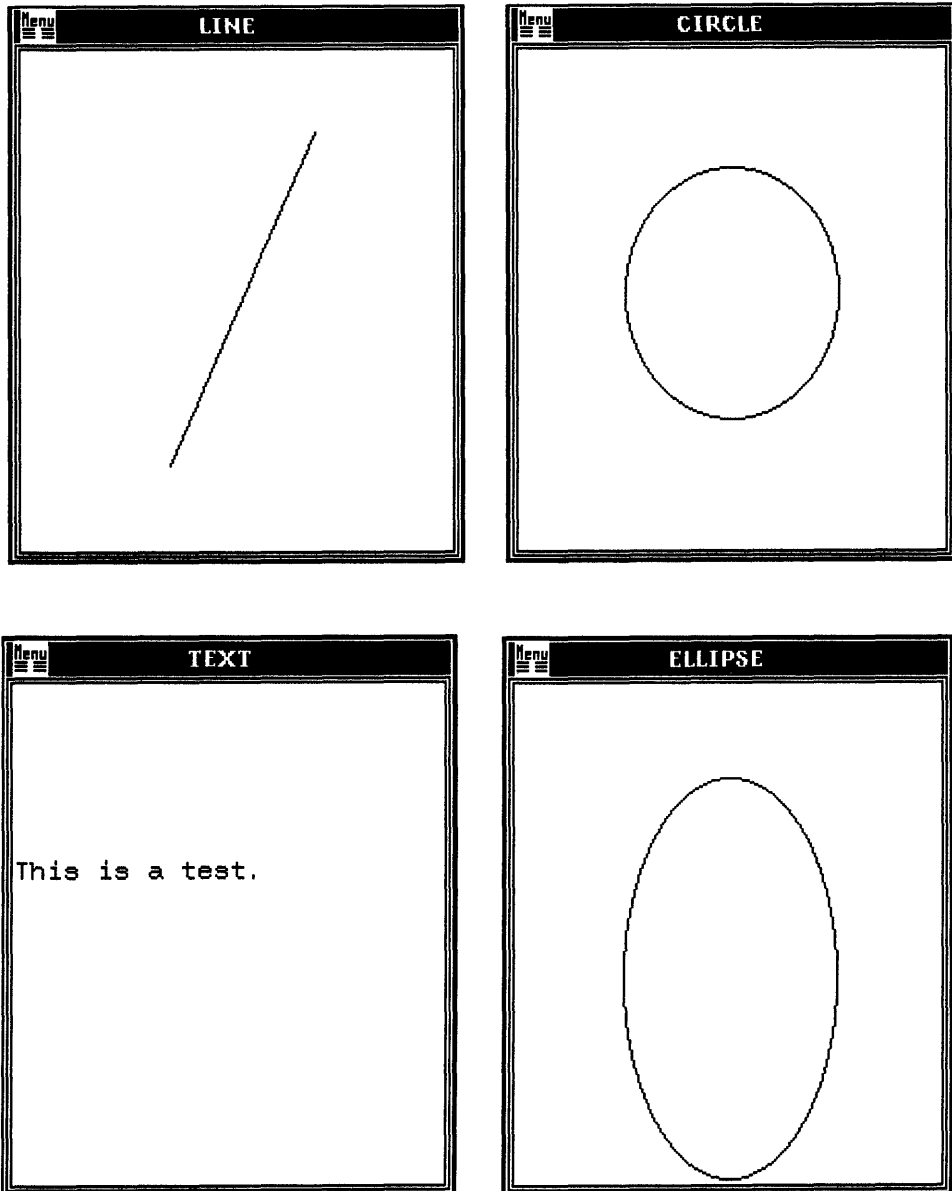
Program is suspended ⑬. The display screen contains one display viewport LINE.

A call to UIS\$MOVE_WINDOW ⑭ has been inserted. Thus, the display window LINE pans the virtual display.

8.5.3 Calling UIS\$MOVE_WINDOW

The display screen initially contains all four windows as shown in Figure 8-5.

Figure 8-5 Before Panning the Virtual Display

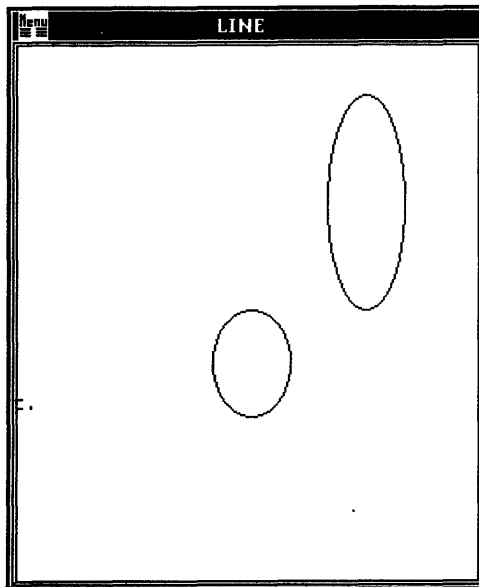
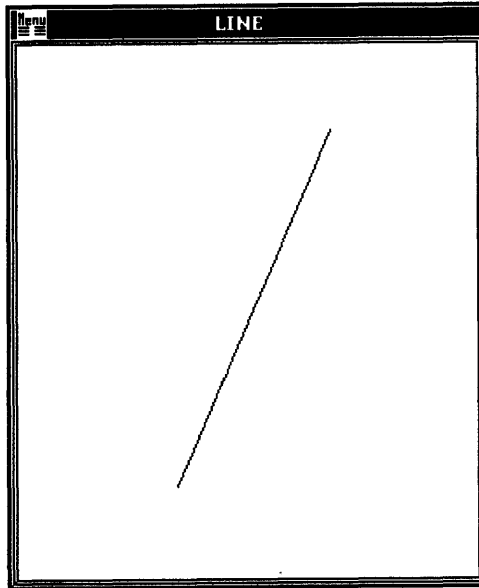


8-16 Display Windows and Viewports

Three of the display windows and viewports are deleted.

The display viewport LINE remains. Originally, the viewport contained a line; now it contains the circle and the ellipse. The display window will go to the location in the virtual display you have specified. You may include as many calls to `UIS$MOVE_WINDOW` as you see fit. Your workstation screen will display the objects shown in Figure 8-6.

Figure 8-6 Panning the Virtual Display



8-18 Display Windows and Viewports

The circle and the ellipse still exist in the virtual display.

8.5.4 Program Development II

Programming Objectives

To demonstrate exact placement of the display viewport on the display screen in order to pop and push viewports.

Programming Tasks

1. Create a viewport attributes data structure specifying viewport placement data.
2. Create a virtual display.
3. Draw two graphic objects in the virtual display in separate viewports.
4. One viewport will occlude the other initially.

```
PROGRAM IMAGES_6
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL WIDTH,HEIGHT

STRUCTURE/PLACE/ ①
  INTEGER*4   CODE_1
  REAL*4      ABS_POS_X
  INTEGER*4   CODE_2
  REAL*4      ABS_POS_Y
  INTEGER*4   END_OF_LIST
END STRUCTURE

RECORD /PLACE/PLACE_LIST,ON_TOP ②

PLACE_LIST.CODE_1=WDPL$C_ABS_POS_X
PLACE_LIST.ABS_POS_X=8 ③
PLACE_LIST.CODE_2=WDPL$C_ABS_POS_Y
PLACE_LIST.ABS_POS_Y=8 ④
PLACE_LIST.END_OF_LIST=WDPL$C_END_OF_LIST

ON_TOP.CODE_1=WDPL$C_ABS_POS_X
ON_TOP.ABS_POS_X=8.5 ⑤
ON_TOP.CODE_2=WDPL$C_ABS_POS_Y
ON_TOP.ABS_POS_Y=8.5 ⑥
ON_TOP.END_OF_LIST=WDPL$C_END_OF_LIST

TYPE *, 'ENTER DISPLAY SIZE'
ACCEPT *,WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,10.0,10.0)
```

```

CALL UIS$CIRCLE(VD_ID,0,10.0,10.0,1.0)
CALL UIS$PLOT(VD_ID,0,4.0,3.0,5.0,7.0)

WD_ID1=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','CIRCLE',
2      8.0,8.0,12.0,12.0,WIDTH,HEIGHT,PLACE_LIST) ⑦
WD_ID2=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','LINE',
2      3.0,2.0,6.0,8.0,,,ON_TOP) ⑧

PAUSE ⑨

CALL UIS$POP_VIEWPORT(WD_ID1) ⑩
.
.
.

PAUSE

CALL UIS$PUSH_VIEWPORT(WD_ID1) ⑪

PAUSE

CALL UIS$DELETE_DISPLAY(VD_ID)

END

```

A data structure argument ① is created and given the symbolic name PLACE using the STRUCTURE statement. The symbolic names for the fields were chosen arbitrarily.

Two variables, PLACE_LIST and ON_TOP, of type PLACE are created ② and contain five longwords.

Actual values are assigned to the different fields of the record PLACE_LIST. In this case, the absolute coordinates of the lower-left corner ③ ④ of the display viewport LINE are assigned to the fields ON_TOP.ABS_POS_X and ON_TOP.ABS_POS_Y ⑤ ⑥. The absolute coordinates of the display viewport CIRCLE, are assigned to the fields PLACEMENT.ABS_POS_X and PLACEMENT.ABS_POS_Y as well.

Also, the position of your calls to UIS\$CREATE_WINDOW ⑦ ⑧ within your program is important. The call to create the display viewport CIRCLE must be executed prior to LINE.

At the first PAUSE statement ⑨, viewport LINE occludes viewport CIRCLE.

UIS\$POP_VIEWPORT is called ⑩. The display viewport CIRCLE is placed over the viewport LINE.

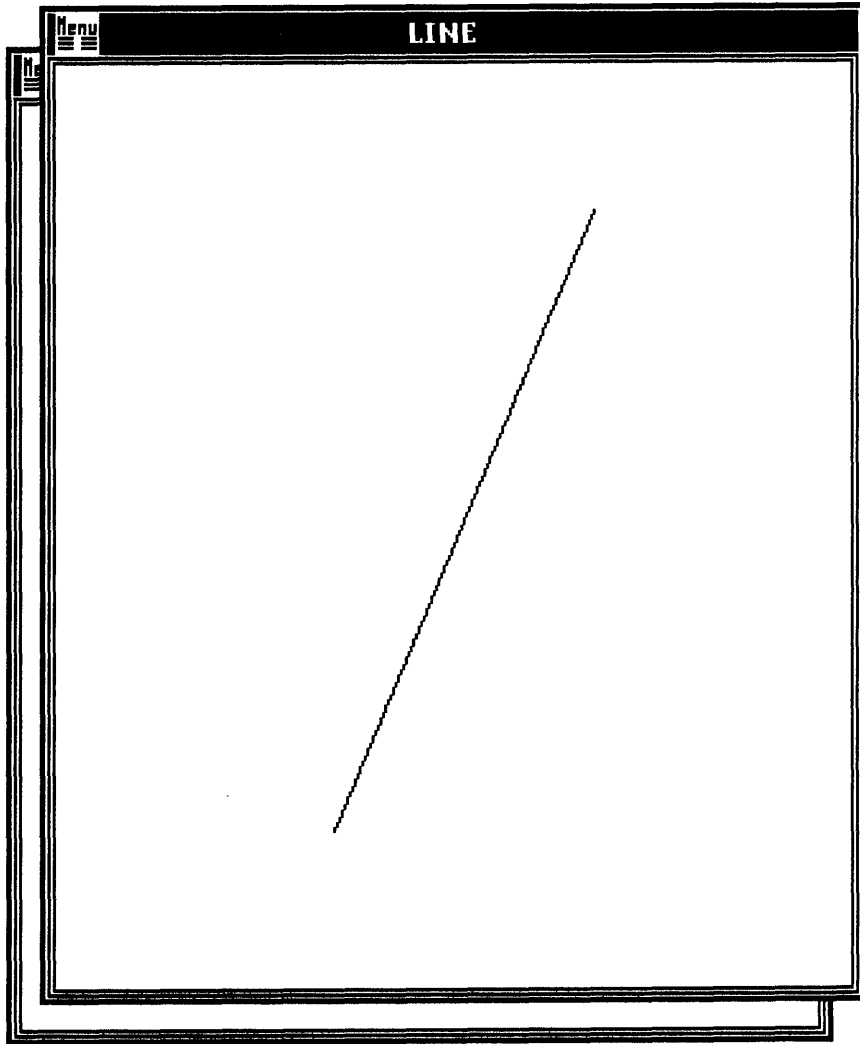
A call to UIS\$PUSH_VIEWPORT ⑪ returns the viewports to their original position.

8.5.5 Calling UIS\$POP_VIEWPORT and UIS\$PUSH_VIEWPORT

Initially, the viewport LINE is placed over CIRCLE. Note that display viewports are placed on the physical display screen with absolute coordinates. The lower-left corner of any viewport is the origin of the viewport rectangle. When you request exact placement of a viewport, you are specifying where on display screen the origin of the viewport rectangle is to be placed relative to the lower-left corner of the display screen.

The program execution is suspended at the first PAUSE statement. The display screen contains the graphic objects shown in Figure 8-7.

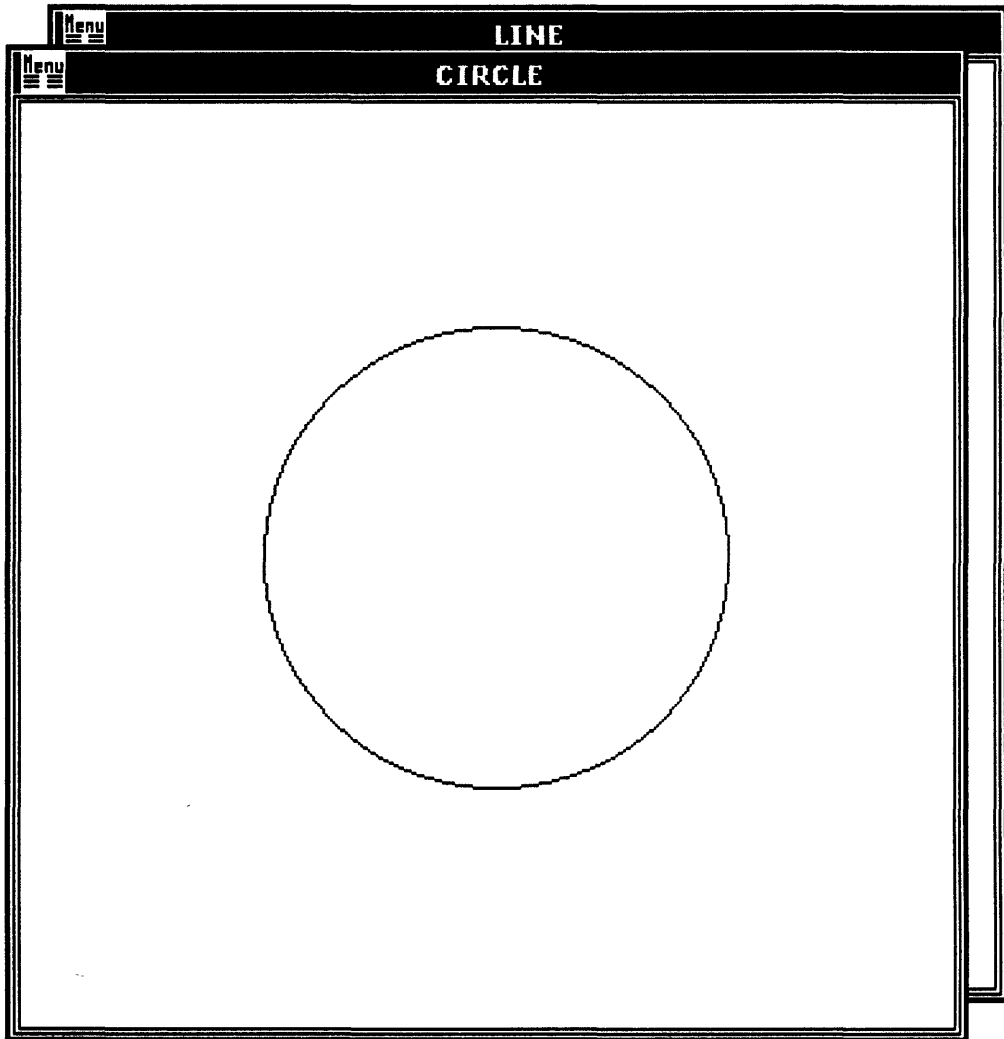
Figure 8-7 Occluding a Display Viewport



8-22 Display Windows and Viewports

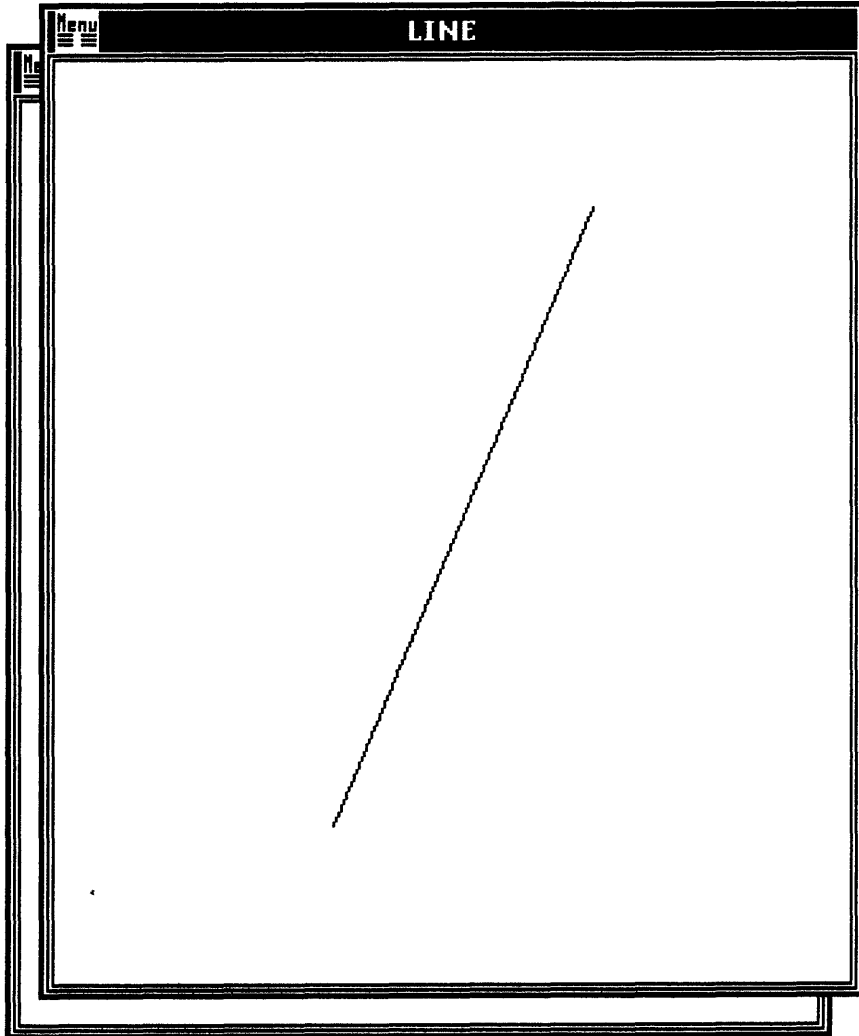
The display viewports LINE and CIRCLE exchange positions when the call to `UIS$POP_VIEWPORT` is executed. The viewport CIRCLE now occludes LINE as shown in Figure 8-8.

Figure 8-8 Popping a Display Viewport



In order to return the viewports to their original positions, a call to `UIS$PUSH_VIEWPORT` pushes viewport CIRCLE behind viewport LINE as shown in Figure 8-9.

Figure 8-9 Pushing a Display Viewport



8-24 Display Windows and Viewports

8.5.6 Program Development III

Programming Objectives

To place a viewport in a general vicinity on the display screen and to create a display viewport with no border.

Programming Tasks

1. Create a viewport attributes list to hold the appropriate viewport placement and attributes data.
2. Create a virtual display.
3. Draw two graphic objects in the virtual display.
4. Create two display windows and associated viewports each containing a graphic object.
5. Delete the virtual display.

```
PROGRAM IMAGES_7
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL WIDTH,HEIGHT

STRUCTURE/PLACE/ ①
INTEGER*4 CODE_5
INTEGER*4 REL_POS
INTEGER*4 CODE_6
INTEGER*4 ATTR
INTEGER*4 END_OF_LIST
END STRUCTURE

RECORD /PLACE/LOCATION(2) ②

LOCATION(1).CODE_5=WDPL$C_PLACEMENT
LOCATION(1).REL_POS=WDPL$M_TOP .OR. WDPL$M_LEFT ③
LOCATION(1).CODE_6=WDPL$C_ATTRIBUTES
LOCATION(1).ATTR=WDPL$M_NOMENU_ICON
LOCATION(1).END_OF_LIST=WDPL$C_END_OF_LIST

LOCATION(2).CODE_5=WDPL$C_PLACEMENT
LOCATION(2).REL_POS=WDPL$M_RIGHT .OR. WDPL$M_BOTTOM ④
LOCATION(2).CODE_6=WDPL$C_ATTRIBUTES
LOCATION(2).ATTR=WDPL$M_NOBORDER
LOCATION(2).END_OF_LIST=WDPL$C_END_OF_LIST

TYPE *,'ENTER VIEWPORT WIDTH AND HEIGHT'
ACCEPT *,WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,20.0,20.0,10.0,10.0)
```

```

CALL UIS$CIRCLE(VD_ID,0,12.0,12.0,1.0)
CALL UIS$ELLIPSE(VD_ID,0,15.0,15.0,1.0,2.0)

WD_ID1=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'CIRCLE',
2      10.0,10.0,14.0,14.0,WIDTH,HEIGHT,LOCATION(1))
WD_ID4=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'ELLIPSE',
2      13.0,13.0,17.0,18.0,WIDTH,HEIGHT,LOCATION(2))

PAUSE

CALL UIS$DELETE_DISPLAY(VD_ID)

PAUSE

END

```

The name of the data structure argument PLACE is defined using the STRUCTURE statement ❶. An array LOCATION is defined to have two elements that are records with a structure defined by the structure PLACE ❷. Each record LOCATION(1) and LOCATION(2) consists of two pairs of longwords terminated by a longword equaling the constant WDPL\$C_END_OF_LIST.

We prefer to have the display viewport CIRCLE placed in the upper-left corner of the display screen and the borderless viewport ELLIPSE in the lower-right corner. Therefore, we must specify in each assignment two preference masks for each viewport ❸ ❹.

NOTE: Note that you must use the logical operator .OR. when specifying more than one preference mask.

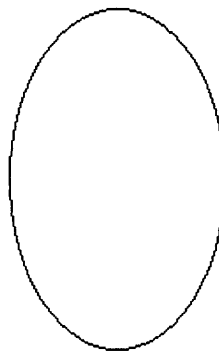
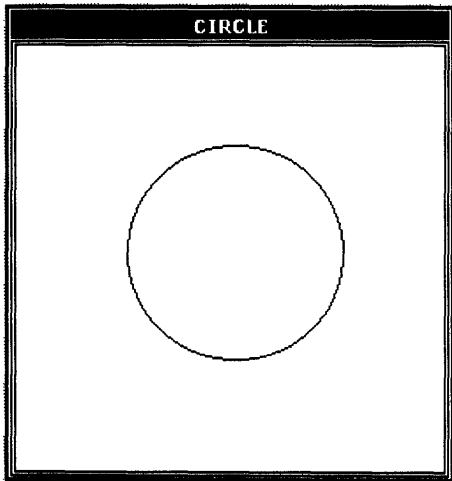
The array name LOCATION is added to the argument lists of the viewport CIRCLE and ELLIPSE to invoke the optional attribute list.

8.5.7 Requesting General Placement and No Border

General display viewport placement works best on an uncluttered display screen. Your workstation screen will display the objects shown in Figure 8-10.

8-26 Display Windows and Viewports

Figure 8-10 General Placement and No Border



8.5.8 Program Development IV

Programming Objectives

To move graphic objects within the virtual display.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Draw two graphic objects in the virtual display.
4. Move the coordinate space containing each graphic object to another portion of the virtual display using `UIS$MOVE_AREA`.

```

PROGRAM AREA
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LI

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,50.0,50.0,15.0,15.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','MOVE AREA')

CALL UIS$PLOT(VD_ID,0,1.0,25.0,16.0,25.0,9.0,42.0,1.0,25.0) ①
CALL UIS$CIRCLE(VD_ID,0,35.0,35.0,10.0) ②

PAUSE
CALL UIS$MOVE_AREA(VD_ID,0.0,22.0,20.0,42.0,30.0,1.0) ③
CALL UIS$MOVE_AREA(VD_ID,25.0,25.0,50.0,50.0,1.0,1.0) ④

PAUSE
END

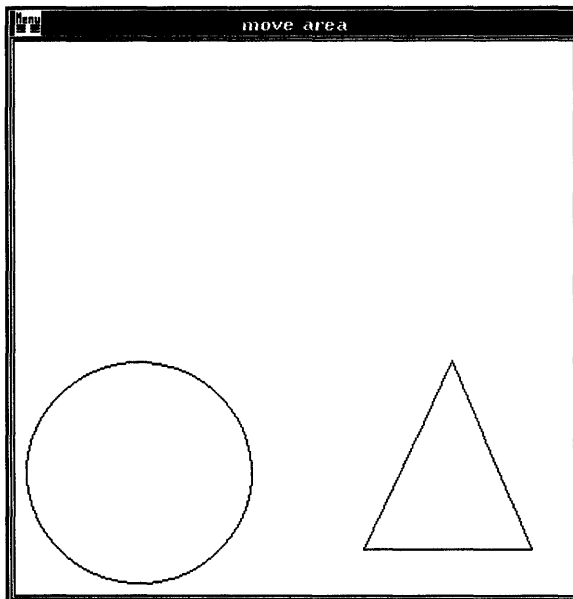
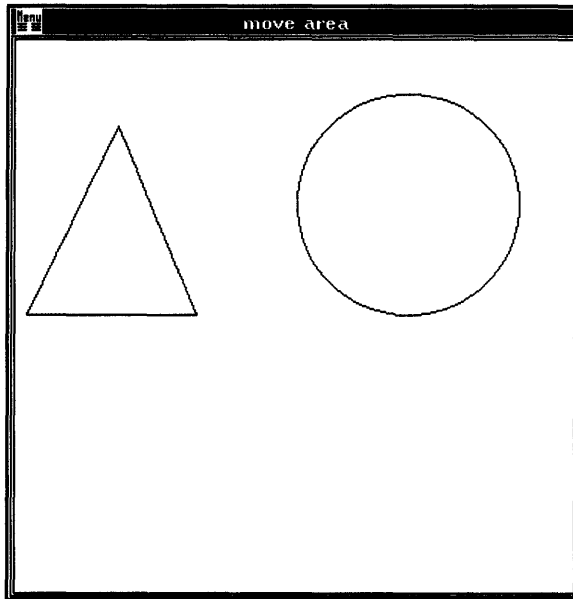
```

A triangle and a circle are drawn in the upper half of the virtual display using `UIS$PLOT` and `UIS$CIRCLE` ① ②.

A rectangular area containing the triangle is moved to the lower-right area of the virtual display ③. A rectangular area containing the circle is moved to the lower-left region in the virtual display ④.

8-28 Display Windows and Viewports

Figure 8-11 Moving Graphic Objects Within the Virtual Display



8.5.9 Calling UIS\$MOVE_AREA

Figure 8-11 shows how areas within the virtual display containing graphic objects can be moved to other parts of the same virtual display.

8.6 World Coordinate Transformations

Certain applications may require that you create more than one virtual display, or *world coordinate space*. Depending on the requirements of the program, you might have to map graphic objects in one virtual display to another virtual display.

8.6.1 Programming Options

To illustrate the advantages of world coordinate transformations, we will construct a program that creates a virtual display. We will then create a circle in a virtual display. The circle will be written to new world coordinate space or transformation space.

Two-Dimensional Transformation and Scaling

Depending on the values supplied to UIS\$CREATE_TRANSFORMATION, graphic objects mapped to other coordinate spaces may be scaled. If the coordinates of the new transformation space are the same as those of the original virtual display, no scaling occurs.

8.6.2 Program Development

Programming Objectives

To transform a world coordinate space by altering its mapping and scaling factors.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Draw a graphic object in the virtual display.
4. Create a new coordinate space using UIS\$CREATE_TRANSFORMATION.
5. Redraw the graphic object substituting the transformation identifier of the new coordinate space returned by UIS\$CREATE_TRANSFORMATION for the virtual display identifier of the old coordinate space.

8-30 Display Windows and Viewports

```
PROGRAM TRANS
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(-5.0,-5.0,25.0,25.0,10.0,10.0) ❶
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','TRANSFORMATION')
CALL UIS$CIRCLE(VD_ID,0,6.0,6.0,7.0) ❷

TR_ID=UIS$CREATE_TRANSFORMATION(VD_ID,-5.0,-5.0,
2      17.5,17.5) ❸
CALL UIS$CIRCLE(TR_ID,0,6.0,6.0,7.0) ❹

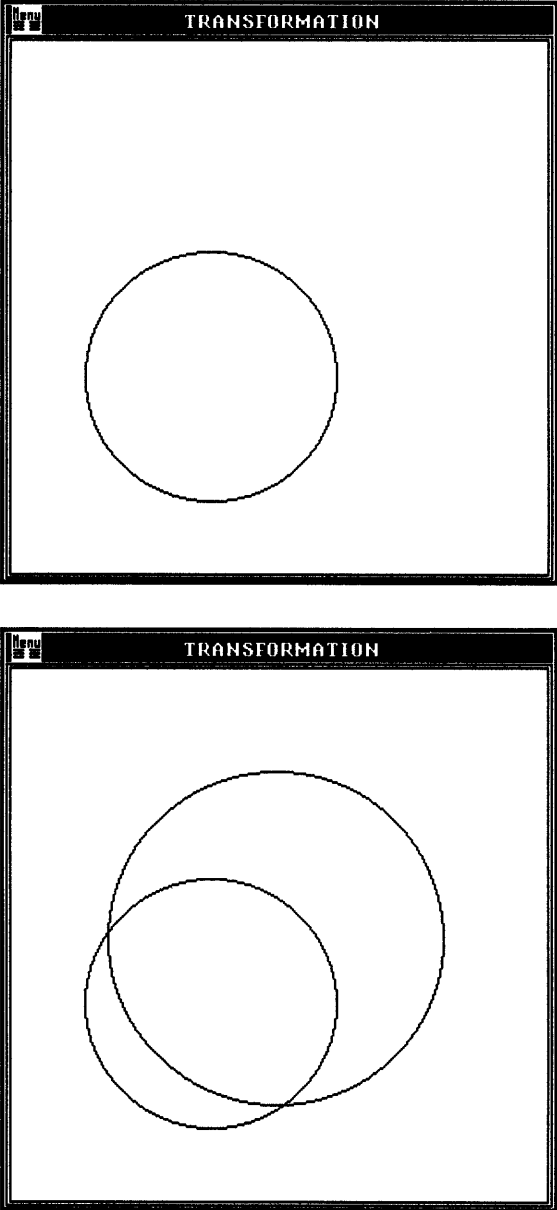
PAUSE
END
```

The virtual display ❶ and the new transformation space ❸ specify different coordinate ranges. The circles are created in calls to `UIS$CIRCLE` ❷ ❹ where the `tr_id` argument is substituted for `vd_id` in the second call. The same circle is redrawn with the same world coordinates in the new transformation space.

8.6.3 Calling `UIS$CREATE_TRANSFORMATION`

The graphic objects appear to be superimposed one over the other. If the `vd_x1` and `vd_y1` arguments are manipulated, the size of the arc can increase or decrease relative to the size of the first circle. In any case, the arc is mapped to the transformation space eliminating the need for additional computation and coding on the part of the programmer.

Figure 8-12 World Coordinate Transformations



Chapter 9

General Attributes

9.1 Overview

Until the information presented in this manual has been concerned with UIS output routines that create the basic structures needed to produce graphic objects. However, there are other types of routines. This chapter discusses the following topics:

- Understanding general attributes
- Using general attributes

The attribute routines place a great deal of control over the quality of graphic objects and text in the hands of the programmer.

9.2 Attributes—How to Use Them

As the canvas gradually fills with various shapes and figures, the artist is concerned not only with the shapes of the subjects—a line, a circle, an ellipse, and text but also with whether their appearance conveys the intended meaning. What our artist would regard as an aesthetic consideration, we will call an *attribute*. Attributes control the appearance of graphic objects and text. You will use attributes whenever you need to enhance some element on the display screen. Attributes can be modified at any time within your program.

9-2 General Attributes

9.2.1 Attribute Blocks

All UIS attributes are grouped in a data structure called an *attribute block*. One or more attributes may be modified within a given attribute block. Default attribute settings reside in *attribute block 0*. Table 9-1 lists the categories of attributes within attribute block 0.

Table 9-1 Attribute Block 0

Type	Attribute
General	Writing mode
	Writing color index
	Background color
Text	Character rotation
	Character spacing
	Character slant
	Character size
	Text path
	Text slope
	Text formatting
	Left margin
	Right margin
	Font
Graphics	Line width
	Line style
	Fill pattern
	Arc type
Windowing	Clipping rectangle

The default attribute settings in attribute block 0 can never be modified.

9.2.2 Modifying General Attributes

When you modify general attributes, you do not change the default attribute settings within attribute block 0 itself. You should think of attribute block 0 as a template of default settings and you are modifying a copy of this attribute block for use within your program. Attribute modification routines contain two arguments—the input attribute block number (**iatb**) and the output attribute block number (**oatb**). Table 9-2 lists the default settings of general attributes.

Table 9-2 Default Settings of General Attributes

General Attribute	Default Setting	Modification Routine
Background index ¹	Index 0	UIS\$SET_BACKGROUND_INDEX
Writing index ²	Index 1	UIS\$SET_WRITING_INDEX
Writing mode	Overlay	UIS\$SET_WRITING_MODE

¹Index of the background color in the virtual color map.

²Index of the foreground color in the virtual color map.

Perform attribute modification using the following procedure:

1. Choose an appropriate attribute modification routine to modify the attribute.
2. Specify 0 as the **iatb** argument to obtain a copy of attribute block 0.
3. Specify a number from 1 to 255 as the **oatb** argument. The attribute block can then be referenced in subsequent UIS graphics and text routines or in any other attribute modification routine.

Graphics and text routines as well as UIS\$MEASURE_TEXT, UIS\$NEW_TEXT_LINE, and UIS\$SET_ALIGNED_POSITION reference attribute blocks in the **atb** argument.

9.3 Structure of Graphic Objects

There are three types of graphic objects: (1) geometric shapes such as circles, ellipses, points, lines, and polygons, (2) text, and (3) raster images. Graphic objects consist of a *pattern*. In memory, the pattern represents one or more bit settings to 0 or 1 that comprise the actual graphic object.

When these entities are written in the virtual display, the UIS writing modes interpret the bit settings that comprise these objects in different ways.

9-4 General Attributes

Text

In the case of text, a standard character within the default font displayed on the workstation screen represents the bitmap image of a *cell* in memory. The size of the cell varies and depends on the type of font. UIS draws *monospaced* and *proportionally spaced* text. Monospaced fonts use a standard cell size for all letters within the font. However, the standard cell size varies depending on the monospaced font you are using.

Proportionally spaced fonts use character cells that vary in width according to the letter used. The height of the character cell remains constant for all characters within the font.

The character cell contains the pattern. The remaining bits in the cell are set to 0. All bits within the character cell are significant to UIS writing modes.

Geometric Shapes

In the case of geometric shapes, only the bit settings that actually comprise the pattern are significant. Bit settings in the pattern may be 0 or 1. For example, a dotted line represents bit settings of 0 and 1 in a pattern. All bit settings both 0 and 1 within this pattern are significant to UIS writing modes.

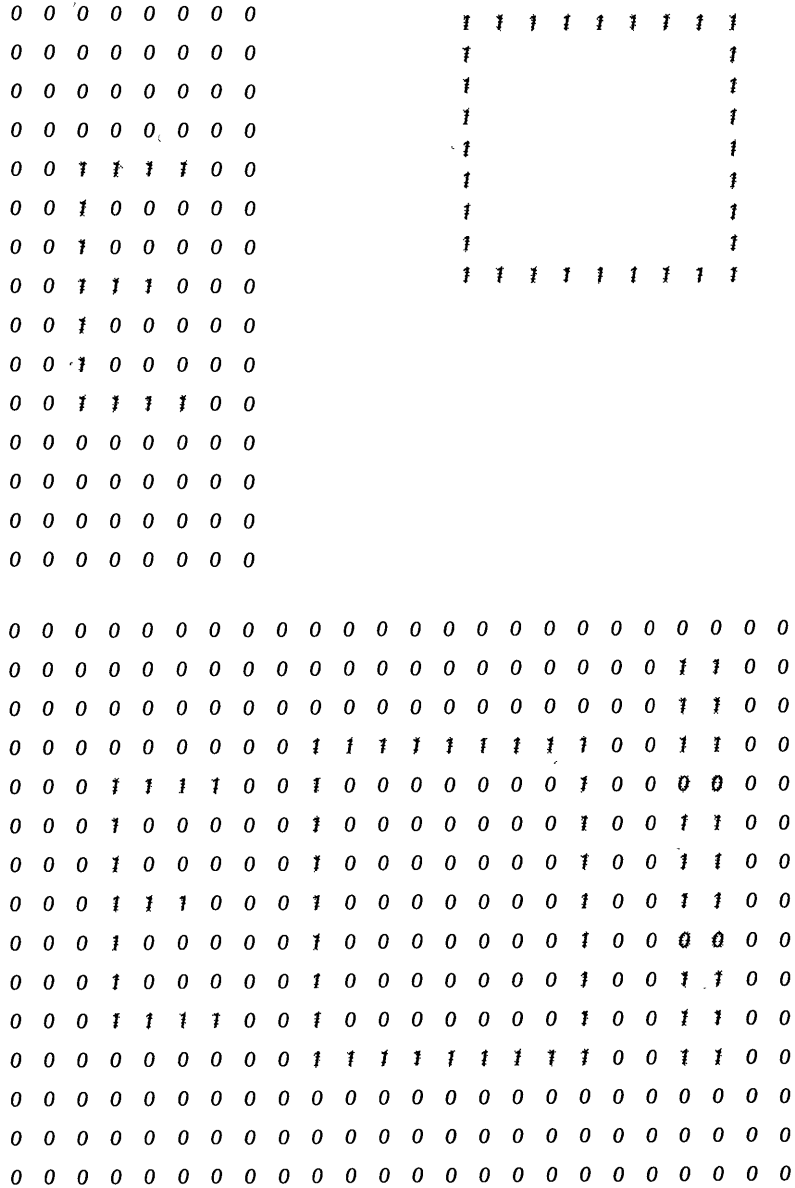
Raster Images

When you draw a raster image, you set bits in a bitmap to create text characters or geometric shapes. For example, `UIS$IMAGE` and `UIS$SET_POINTER_PATTERN` use bitmaps to map rasters to the display screen. All bits in the bitmap are significant to the UIS writing modes. The following table shows the underlying structures from which graphic objects are created.

Graphic Object	Structure
Text	Character cell
Geometric shapes	Pattern
Raster Image	Bitmap image of varying size

For a given graphic object, the current writing mode determines how the bit settings in the appropriate structure are displayed. All bit settings of a particular structure are significant to UIS writing modes. Figure 9-1 shows graphic objects as structures that UIS writing modes recognize: (1) the letter E within a character cell, (2) a square as a pattern, and (3) a bitmap containing the letter E, a square, and a vertical dashed line of double thickness.

Figure 9-1 Structure of Graphic Objects



9.4 UIS Writing Modes

There are 14 UIS writing modes: transparent, complement, copy, copy negate, overlay, overlay negate, erase, erase negate, replace, replace negate, bit set, bit set negate, bit clear, and bit clear negate. The writing mode controls how graphics and text routines use foreground and background colors to display graphic objects. The default writing mode is overlay.

Table 9-3 lists how each writing mode functions.

Table 9-3 UIS Writing Modes

UIS Writing Modes	Function
Device-Independent	
UIS\$C_MODE_ERAS	Displays the current background color for each bit position no matter what the bit settings are in the character cell, pattern, or bitmap image.
UIS\$C_MODE_ERASN	Displays the current writing color for each bit position no matter what the bit settings are in the character cell, pattern, or bitmap image.
UIS\$C_MODE_OVER	Displays the current writing color for bits set to 1 in the character cell, pattern, or bitmap image. All bits set to 0 have no effect on the existing graphic object. This is the default writing mode attribute setting.
UIS\$C_MODE_OVERN	Bitwise complements the character cell, pattern, or bitmap image that is, bits originally set to 0 are now set to 1 and vice versa. The bits now set to 1 in the character cell, pattern, or bitmap image display the current writing color. The bits that are now set to 0 in the character cell have no effect on any existing graphic object.
UIS\$C_MODE_REPL	Displays the current writing color for bits set to 1 in the character cell, pattern, or bitmap image. Bits set to 0 in the character cell, pattern, or bitmap image display the current background color.

Table 9-3 (Cont.) UIS Writing Modes

UIS Writing Modes	Function
Device-Independent	
UIS\$C_MODE_REPLN	Bitwise complements the character cell, pattern, or bitmap image. The bits now set to 1 in the character cell, pattern, or bitmap image now display the current writing color. Bits now set to 0 in the character cell, pattern, or bitmap image now display the current background color.
UIS\$C_MODE_COMP	Where the two graphic objects intersect, the bits in the character cell, pattern, or bitmap image are exclusive .OR.ed with the existing graphic object.
UIS\$C_MODE_TRAN	Does not alter the display screen.
Device-Dependent¹	
UIS\$C_MODE_BIC	The bitwise complement of the character cell, pattern, or bitmap image is logically .AND.ed with the existing graphic object and background. On mapped color systems, where the two graphic objects intersect, the bitwise complement of the writing index of the character cell, pattern, or bitmap image is logically .AND.ed with the pixel values of the existing graphic object and background.
UIS\$C_MODE_BICN	On monochrome systems, the bits in the character cell, pattern, or bitmap image are logically .AND.ed with the existing graphic object and background. On mapped color systems, the writing index of the character cell, pattern, or bitmap image is logically .AND.ed with the pixel values of the existing graphic object and background.
UIS\$C_MODE_BIS	The bits in the character cell, pattern, or bitmap image are logically .OR.ed with the existing graphic object and background. On mapped color systems, the writing index of the character cell, pattern, or bitmap image is logically .OR.ed with the pixel values of the existing graphic object and background.

¹These UIS writing modes produce device-dependent results. Depending on the specific operation, graphic objects drawn using these writing modes may appear differently on VAXstation monochrome and color systems.

9-8 General Attributes

Table 9-3 (Cont.) UIS Writing Modes

UIS Writing Modes	Function
Device-Dependent¹	
UIS\$C_MODE_BISN	On monochrome systems, the bitwise complement of the character cell, pattern, or bitmap image is logically .OR.ed with the existing graphic object and background. On color systems, the bitwise complement of the writing index of the character cell, pattern, or bitmap image is logically .OR.ed with the pixel values of the existing graphic object and background.
UIS\$C_MODE_COPY	Displays the character cell, pattern, or bitmap image without regard to current background and writing color. On a VAXstation monochrome system, bits set to 0 are black, and bits set to 1 are white. On mapped color systems, the writing index of the character cell, pattern, or bitmap is used directly as an index.
UIS\$C_MODE_COPYN	Displays the character cell, pattern, or bitmap image without regard to current background and writing color. On monochrome systems, bits set to 0 are white and bits set to 1 are black. On mapped color systems, the bitwise complement of the writing index of the character cell, pattern, or bitmap image is used directly as an index.

¹These UIS writing modes produce device-dependent results. Depending on the specific operation, graphic objects drawn using these writing modes may appear differently on VAXstation monochrome and color systems.

9.4.1 Using General Attributes

General attributes affect all graphics images displayed on the screen. These attributes are background color, writing color (foreground), and writing mode.

9.4.1.1 Programming Options

A program can set different background and writing colors for different display viewports for application-specific reasons or, simply, for variety.

Setting the Background Color

Modifying the background color attribute sets the value of an index into the color map. Modifying the background color affects how the current writing mode interprets the bits that comprise background color of the graphic object. You can set the background color attribute with UIS\$SET_BACKGROUND_INDEX.

Setting the Writing Color

Modifying the writing color attribute sets the value of an index into the color map. Writing color affects the color of the graphic object. You can set the writing color with `UIS$SET_WRITING_INDEX`.

Setting the Writing Mode

The writing mode controls how background and foreground colors are used to draw graphic objects in the virtual display. You can specify the writing mode using `UIS$SET_WRITING_MODE`.

9.4.1.2 Program Development I**Programming Objective**

To draw a graphic object in each of the UIS device-independent writing modes using the default background and writing color attribute settings.

Programming Tasks

1. Create a virtual display.
2. Create a display window and associated viewport.
3. Draw a line using the default overlay writing mode in the virtual display.
4. Draw a character at same location in each of the UIS writing modes.
5. Erase graphic objects in the virtual display using `UIS$ERASE` and delete the window using `UIS$DELETE_WINDOW`.
6. Repeat steps 3 through 5.

The font name `MY_FONT_5` is a logical name.

```

PROGRAM MODE
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,3.0,3.0,6.0,5.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')

CALL UIS$PLOT(VD_ID,0,0.5,1.0,2.0,2.5)

PAUSE

C      Erase the object in the virtual display and delete the window
C      Display window is deleted in order to change viewport title

CALL UIS$ERASE(VD_ID,0.0,0.0,3.0,3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

```

9-10 General Attributes

```
WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'OVERLAY')
CALL UIS$SET_FONT(VD_ID, 0, 1, 'MY_FONT_5')
CALL UIS$PLOT(VD_ID, 0, 0.5, 1.0, 2.0, 2.5)
CALL UIS$TEXT(VD_ID, 1, 'D', 1.0, 2.0)

PAUSE

CALL UIS$ERASE(VD_ID, 0.0, 0.0, 3.0, 3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'OVERLAY NEGATE')
CALL UIS$SET_WRITING_MODE(VD_ID, 1, 2, UIS$C_MODE_OVERN)
CALL UIS$PLOT(VD_ID, 0, 0.5, 1.0, 2.0, 2.5)
CALL UIS$TEXT(VD_ID, 2, 'D', 1.0, 2.0)

PAUSE

CALL UIS$ERASE(VD_ID, 0.0, 0.0, 3.0, 3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'REPLACE')
CALL UIS$SET_WRITING_MODE(VD_ID, 2, 3, UIS$C_MODE_REPL)
CALL UIS$PLOT(VD_ID, 0, 0.5, 1.0, 2.0, 2.5)
CALL UIS$TEXT(VD_ID, 3, 'D', 1.0, 2.0)

PAUSE

CALL UIS$ERASE(VD_ID, 0.0, 0.0, 3.0, 3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'REPLACE NEGATE')
CALL UIS$SET_WRITING_MODE(VD_ID, 3, 4, UIS$C_MODE_REPLN)
CALL UIS$PLOT(VD_ID, 0, 0.5, 1.0, 2.0, 2.5)
CALL UIS$TEXT(VD_ID, 4, 'D', 1.0, 2.0)

PAUSE

CALL UIS$ERASE(VD_ID, 0.0, 0.0, 3.0, 3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'ERASE')
CALL UIS$SET_WRITING_MODE(VD_ID, 4, 5, UIS$C_MODE_ERAS)
CALL UIS$PLOT(VD_ID, 0, 0.5, 1.0, 2.0, 2.5)
CALL UIS$TEXT(VD_ID, 5, 'D', 1.0, 2.0)

PAUSE
```

```

CALL UIS$ERASE(VD_ID,0.0,0.0,3.0,3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'ERASE NEGATE')
CALL UIS$SET_WRITING_MODE(VD_ID,5,6,UIS$C_MODE_ERASN)

CALL UIS$PLOT(VD_ID,0,0.5,1.0,2.0,2.5)
CALL UIS$TEXT(VD_ID,6, 'D',1.0,2.0)

PAUSE

CALL UIS$ERASE(VD_ID,0.0,0.0,3.0,3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'TRANSPARENT')
CALL UIS$SET_WRITING_MODE(VD_ID,6,7,UIS$C_MODE_TRAN)

CALL UIS$PLOT(VD_ID,0,0.5,1.0,2.0,2.5)
CALL UIS$TEXT(VD_ID,7, 'D',1.0,2.0)

PAUSE

CALL UIS$ERASE(VD_ID,0.0,0.0,3.0,3.0)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'COMPLEMENT')
CALL UIS$SET_WRITING_MODE(VD_ID,7,8,UIS$C_MODE_COMP)

CALL UIS$PLOT(VD_ID,0,0.5,1.0,2.0,2.5)
CALL UIS$TEXT(VD_ID,8, 'D',1.0,2.0)

PAUSE

END

```

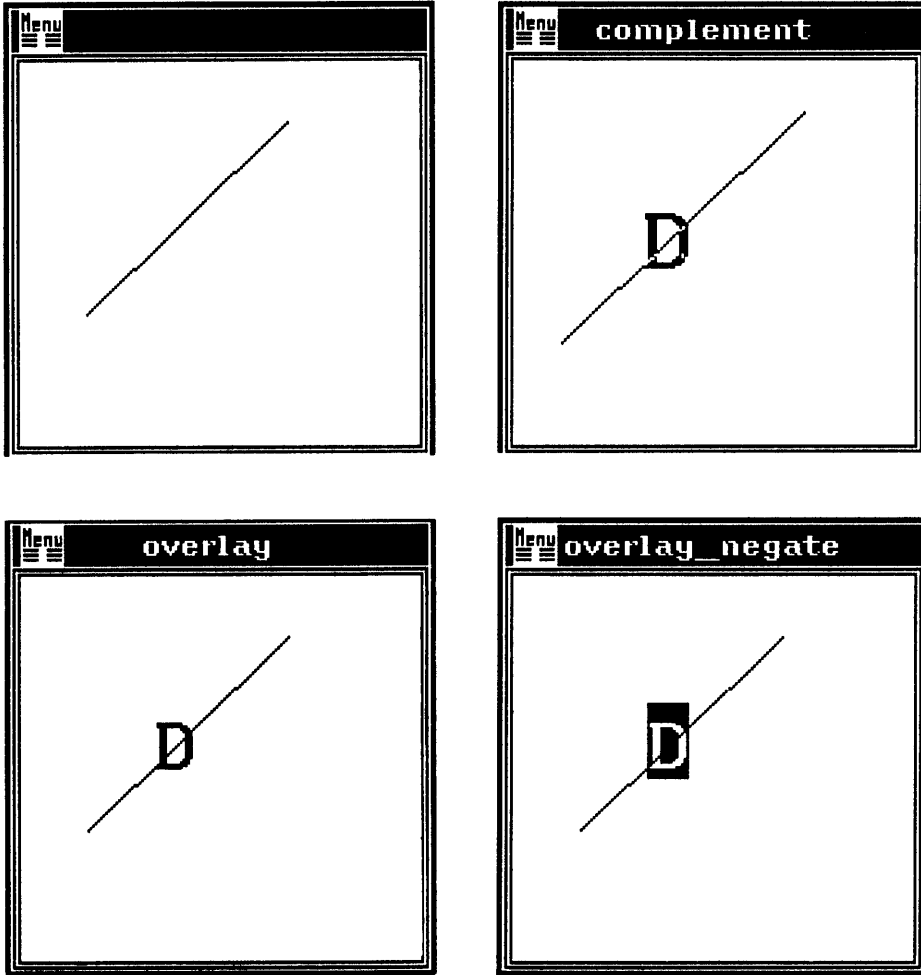
The program MODE sets the writing mode attribute ten times. The letter D is placed over the line. Table 9-3 describes the behavior of the UIS writing modes when text or geometric shapes such as circles are placed on top of an existing graphic object. Remember character cells refer to text, while patterns refer to geometric shapes.

9.4.1.3 Calling UIS\$SET_BACKGROUND_INDEX, UIS\$SET_WRITING_INDEX, and UIS\$SET_WRITING_MODE

To illustrate the effects of the writing modes, imagine that the character cell is slowly lowered onto the virtual display containing an existing graphic object drawn in OVERLAY mode—a line. As it approaches the plane of the virtual display, the writing mode of the character cell determines the final appearance of the graphic object. See Table 9-3 for a description of each writing mode.

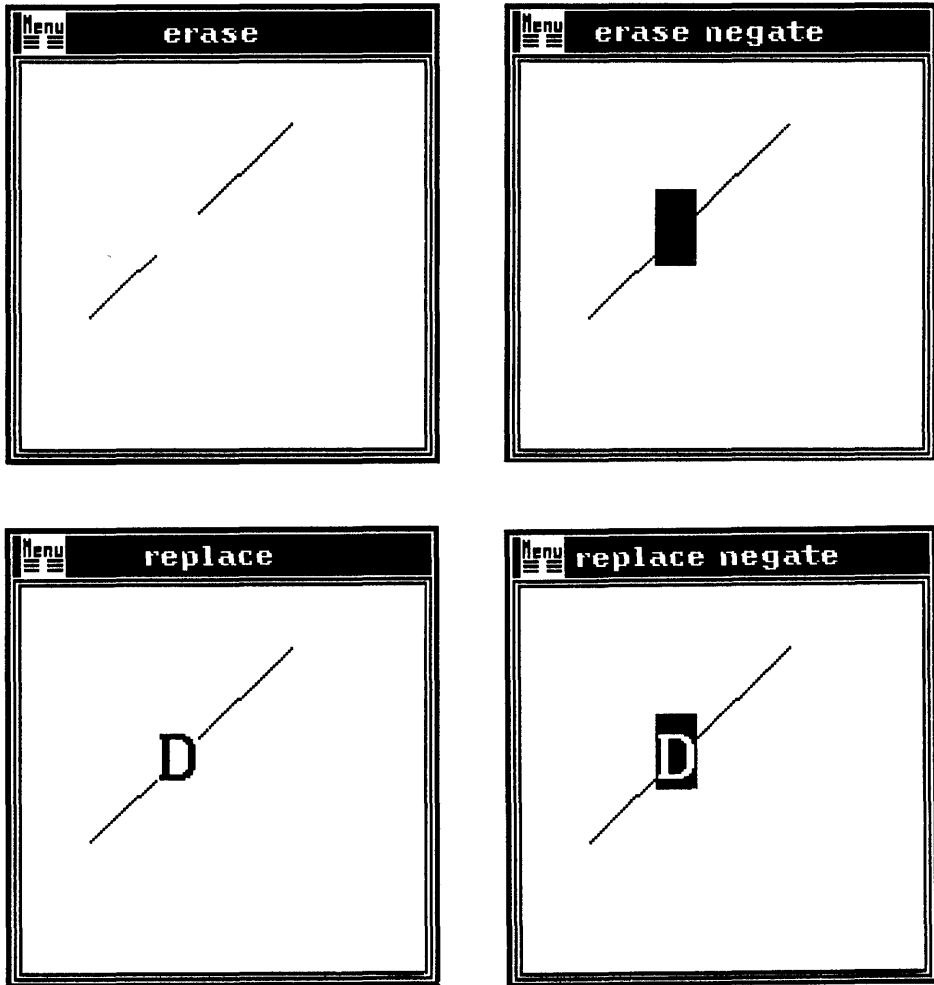
The default background and writing color are in effect as shown in Figure 9-2.

Figure 9-2 UIS Device-Independent Writing Modes



ZK-4543-85

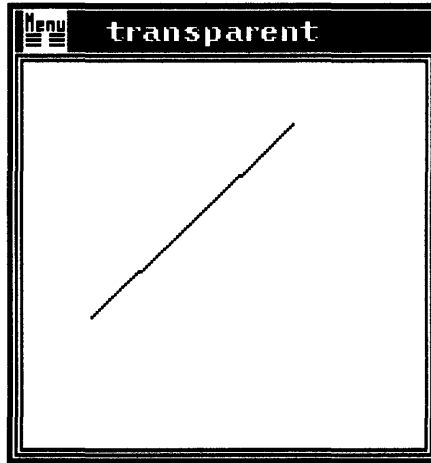
Figure 9-2 (Cont.) UIS Device-Independent Writing Modes



ZK-4544-85

9-14 General Attributes

Figure 9-2 (Cont.) UIS Device-Independent Writing Modes



ZK-4545-85

9.4.1.4 Program Development II

Programming Objective

To illustrate the behavior of the device-dependent writing modes.

Programming Tasks

1. Create an eight-entry virtual color map containing intensity values.
2. Draw three overlapping circles—one in overlay mode and two in bit set mode.
3. Redraw the same circles—one in overlay mode, one in bit clear mode, and one in bit set mode.
4. Redraw two of the circles in the remaining device-dependent writing modes. One circle is always drawn in OVERLAY mode. Both are drawn using the same writing index.

```
PROGRAM PLANE_MODES
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
INCLUDE 'SYS$LIBRARY:UISENTRY'
REAL*4 I_VECTOR(8)
DATA I_VECTOR/0.0,0.125,0.25,0.375,0.50,0.625,0.75,1.0/
DATA VCM_SIZE/8/
DATA INDEX2/2/
DATA INDEX4/4/
```

①
②
③
④
⑤


```

VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE)
VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,40.0,40.0,15.0,15.0,VCM_ID)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
CALL UIS$SET_INTENSITIES(VD_ID,0,8,I_VECTOR)

CALL UIS$SET_FONT(VD_ID,0,1,'UIS$FILL_PATTERNS')
CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_FOREGROUND)

CALL UIS$SET_FONT(VD_ID,0,2,'UIS$FILL_PATTERNS')
CALL UIS$SET_WRITING_INDEX(VD_ID,2,2,INDEX2) ⑥

CALL UIS$SET_WRITING_MODE(VD_ID,2,2,UIS$C_MODE_BIS)
CALL UIS$SET_FILL_PATTERN(VD_ID,2,2,PATT$C_FOREGROUND)
CALL UIS$SET_WRITING_INDEX(VD_ID,2,4,INDEX4) ⑦

CALL UIS$CIRCLE(VD_ID,1,15.0,20.0,10.0) ⑧
CALL UIS$CIRCLE(VD_ID,2,25.0,20.0,10.0) ⑨
CALL UIS$CIRCLE(VD_ID,4,20.0,30.0,10.0) ⑩

PAUSE

CALL UIS$SET_WRITING_MODE(VD_ID,4,4,UIS$C_MODE_BIC) ⑪
CALL UIS$CIRCLE(VD_ID,4,20.0,30.0,10.0)

PAUSE

CALL UIS$ERASE(VD_ID)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
CALL UIS$SET_WRITING_MODE(VD_ID,2,2,UIS$C_MODE_BICN)
CALL UIS$CIRCLE(VD_ID,1,15.0,25.0,10.0)
CALL UIS$CIRCLE(VD_ID,2,25.0,25.0,10.0) ⑬ ⑫

PAUSE

CALL UIS$ERASE(VD_ID)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
CALL UIS$SET_WRITING_MODE(VD_ID,2,2,UIS$C_MODE_BISN)
CALL UIS$CIRCLE(VD_ID,1,15.0,25.0,10.0)
CALL UIS$CIRCLE(VD_ID,2,25.0,25.0,10.0) ⑮ ⑭

PAUSE

CALL UIS$ERASE(VD_ID)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

```

9-16 General Attributes

```

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION')
CALL UIS$SET_WRITING_MODE(VD_ID, 2, 2, UIS$C_MODE_COPY)
CALL UIS$CIRCLE(VD_ID, 1, 15.0, 20.0, 10.0)
CALL UIS$CIRCLE(VD_ID, 2, 25.0, 20.0, 10.0)

PAUSE

CALL UIS$ERASE(VD_ID)
CALL UIS$DELETE_WINDOW(WD_ID)

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION')
CALL UIS$SET_WRITING_MODE(VD_ID, 2, 2, UIS$C_MODE_COPYN)
CALL UIS$CIRCLE(VD_ID, 1, 15.0, 20.0, 10.0)
CALL UIS$CIRCLE(VD_ID, 2, 25.0, 20.0, 10.0)

PAUSE
END

```

An array `I_VECTOR` is declared to hold the intensity values ❶. Each location in the array element is initialized with an intensity value ❷. The color map size variable is initialized to the number of color map entries ❸. Color index variables *index2* and *index4* are initialized ❹ ❺.

Three circles are drawn ❸ ❹ ❷ using three different indices in the virtual color map—index 1 (the default), index 2, and index 4 ❻ ❼. The circles are filled with the current foreground color. The following table lists the circles, their writing modes and indices and corresponding intensity values.

Circle	Writing Mode	Writing Index	Intensity Value
1	Overlay	1	0.0
2	Bit Set	2	0.125
3	Bit Set	4	0.375

The three circles are redrawn with circle 3 drawn in Bit Clear mode ❶.

In subsequent drawings, only overlapping circles 1 and 2 are redrawn. Circle one is always drawn in overlay mode ❸ ❷ ❹ ❶ ❸ while circle 2 is drawn in the remaining writing modes ❹ ❺ ❻ ❼ ❷.

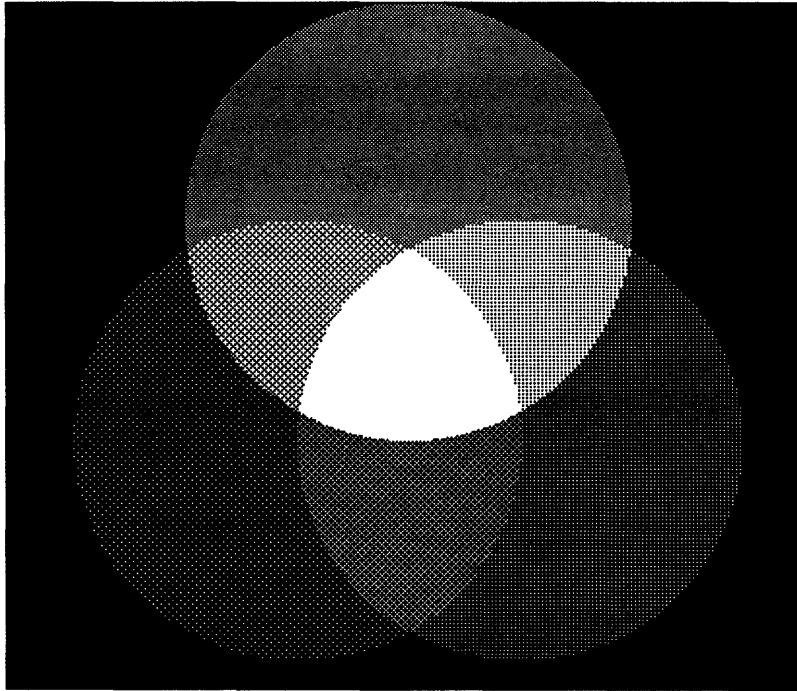
9.4.1.5 Using Device-Dependent Writing Modes

The preceding program PLANE_MODES produced Figures 9-3 through 9-8. In each of the figures, the circle on the left (circle 1) was drawn in overlay mode and writing index 1. The circle on the right (circle 2) was drawn in a different writing mode with a writing index 2. The circle on top (circle 3) was drawn with writing index 4 and is drawn in Figures 9-3 and 9-4 only. The following table lists the writing indices, their binary value and binary bitwise complements.

Object	Writing Index	Binary Value	Bitwise Complement
Background	0	000 ₂	000 ₂
Circle 1	1	001 ₂	110 ₂
Circle 2	2	010 ₂	101 ₂
Circle 3	4	100 ₂	011 ₂

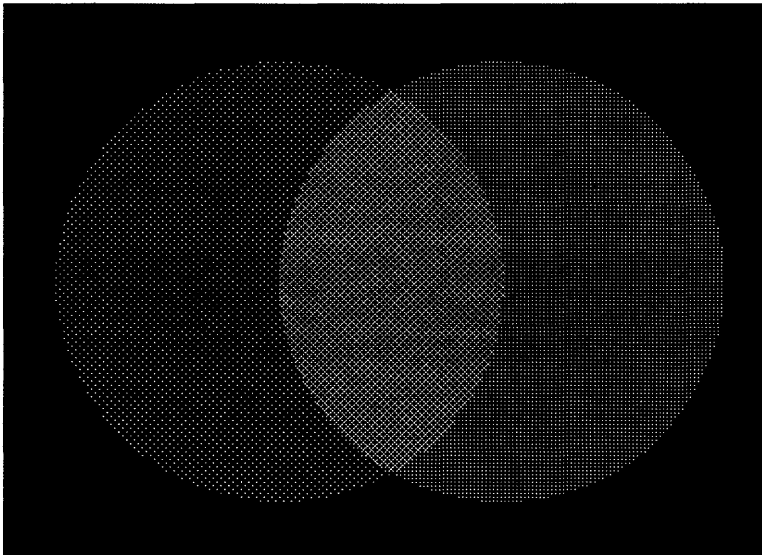
In Figure 9-3, whenever the circles 1, 2, and 3 intersect, their writing indices 001₂, 010₂, and 100₂, are logically .OR.ed with the pixel values of the existing graphic objects and the background. The bit set writing mode has the effect of combining the value of the bit plane settings of each object. Therefore, the intersections of the circles are lighter than the rest of the circles.

Figure 9-3 Bit Set Mode



In Figure 9-4, circle 3 is drawn in bit clear mode with a writing index of 4 or 100_2 . Circle 2 is drawn in bit set mode in writing index 2 or 010_2 . The binary bitwise complement of the writing index of circle 3 is 101_2 . It is logically .AND.ed with the pixel values of the existing graphic objects—circle 1, circle 2, and the background. In bit clear mode the appropriate bit plane settings are now changed such that, circle 3 appears to blend into the background and circles 1 and 2.

Figure 9-4 Bit Clear Mode

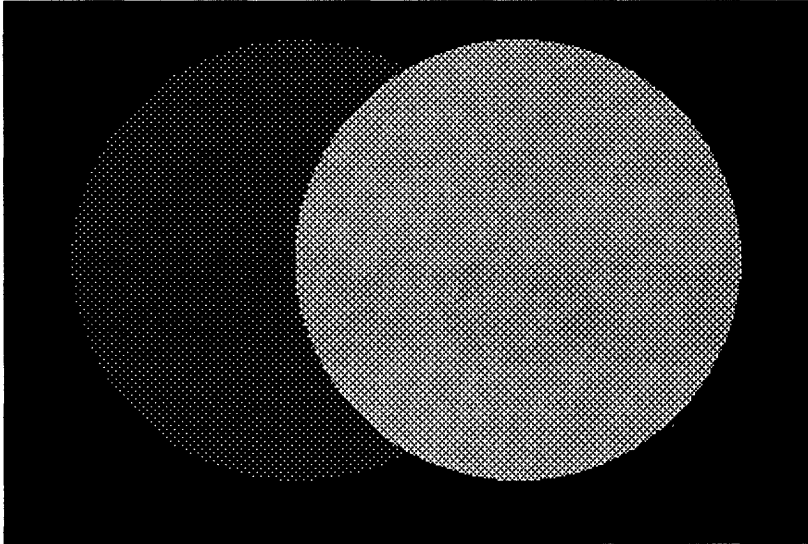


ZK 5486 86

9-20 General Attributes

In Figure 9-5 the binary bitwise complement of the writing index of the circle 2 is 101_2 . It is logically .OR.ed with the pixel values of the existing graphic object and background which are 001_2 and 000_2 . In bit set negate mode the appropriate bit plane settings are now changed such that all of circle 2 is drawn in writing index 5.

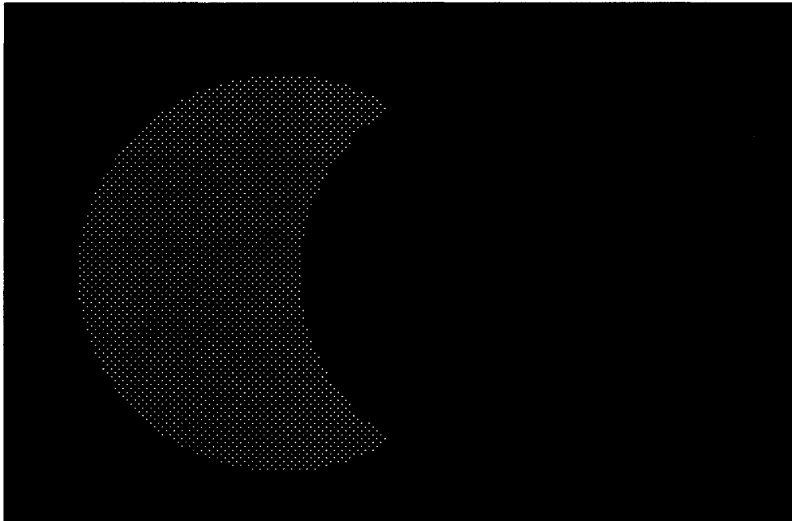
Figure 9-5 Bit Set Negate Mode



ZK 5487 86

In Figure 9-6, the writing index of the circle $2\ 010_2$ is logically .AND.ed with the pixel values of the existing circle 001_2 and the background 000_2 to produce the pixel value 000_2 . The appropriate bit plane settings are now changed such that all of circle 2 including the area of intersection with circle 1 match the background.

Figure 9-6 Bit Clear Negate Mode

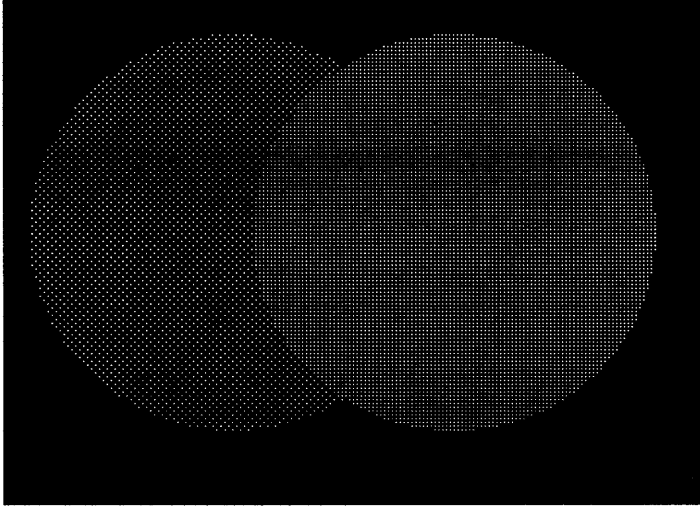


ZK 5488 86

9-22 General Attributes

In figure 9-7 the writing index of circle 2 is used as the index in the virtual color map to draw the circle regardless of existing graphic objects or background.

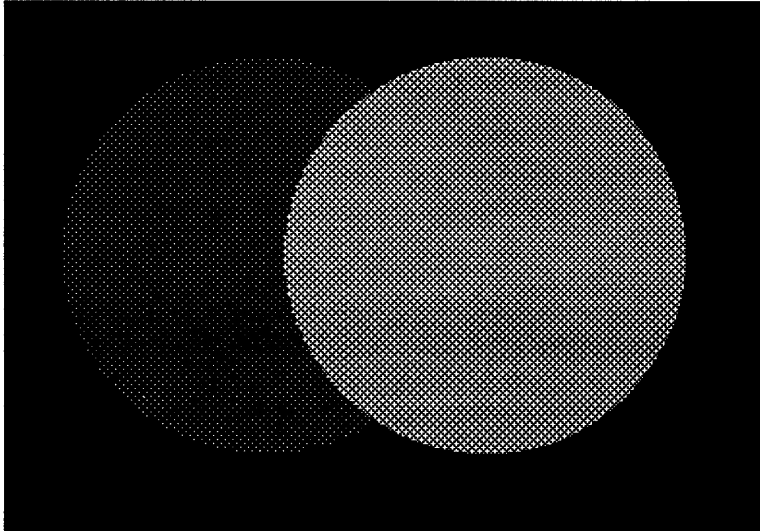
Figure 9-7 Copy Mode



ZK 5489 86

In Figure 9-8, the binary bitwise complement of the writing index of circle 2 101_2 was used as the index into the virtual color map to draw the circle regardless of existing graphic objects or background.

Figure 9-8 Copy Negate Mode



ZK 5490 86

Chapter 10

Text Attributes

10.1 Overview

UIS draws characters in the virtual display according to the specifications of the particular font. The appearance or shape of characters remains unaltered unless an appropriate text attribute is changed. Likewise, UIS draws characters and character strings at user-specified locations within the coordinate space. This orientation within the coordinate space does not change unless an appropriate attribute modification routine is executed.

The orientation and shape of characters and character strings defines spatially how UIS draws these objects on the display screen. Text attribute modification routines allow you to alter the appearance of characters and character strings and redefine the spatial relationship of a character to other characters in significant ways. This chapter discusses the following topics:

- Structure of text
- Using text attributes

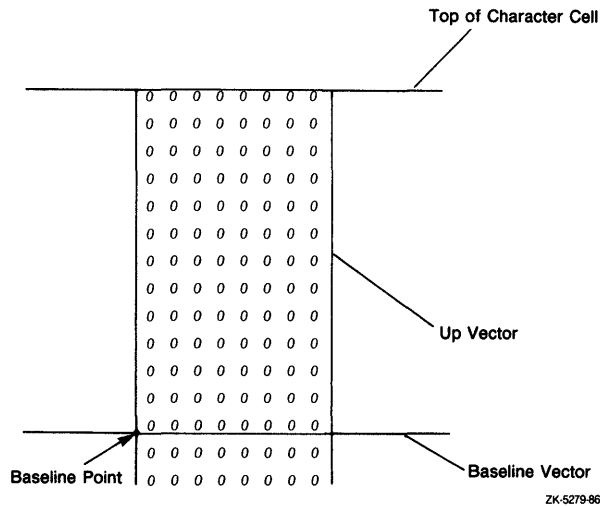
Refer to Section 10.3.1 for information about how to modify the default text attribute settings of attribute block 0.

10.2 Structure of Text

The underlying structure of a single character is a character cell. Every character drawn on the display screen is contained within a character cell. Figure 10–1 describes a character cell and its reference points.

10-2 Text Attributes

Figure 10-1 Character Cell



10.2.1 Monospaced and Proportionally Spaced Fonts

For text drawing purposes fonts are either monospaced or proportionally spaced. Monospaced fonts use a standard character cell size for each character within the font. The character cells of proportionally spaced fonts vary in width for each character within the font, although the height of each cell is the same for each character in the font. Figure 10-2 shows the two types of fonts.

The character cell is a bitmap whose settings are mapped to the display screen as a character.

10.2.2 Lines of Text

Lines of text share a spatial relationship with other lines of text—for example, a line of text within a paragraph. Ordinarily, lines of English text are read from left to right. Your eyes trace an imaginary path across the page from the left margin to the right margin. By default UIS draws lines of text in this left-to-right direction known as the *default major path*. Normally, when you reach the end of the line, you would start reading the next line below this one. When UIS finishes drawing a line of text, the secondary downward movement to begin a new line of text drawing is known as the *default minor path* of text drawing. This is the normal relationship between lines of English text and the direction in which they are drawn. Figure 10-3 illustrates the two default paths that UIS uses to draw text.

Figure 10-2 Monospaced and Proportionally Spaced Characters

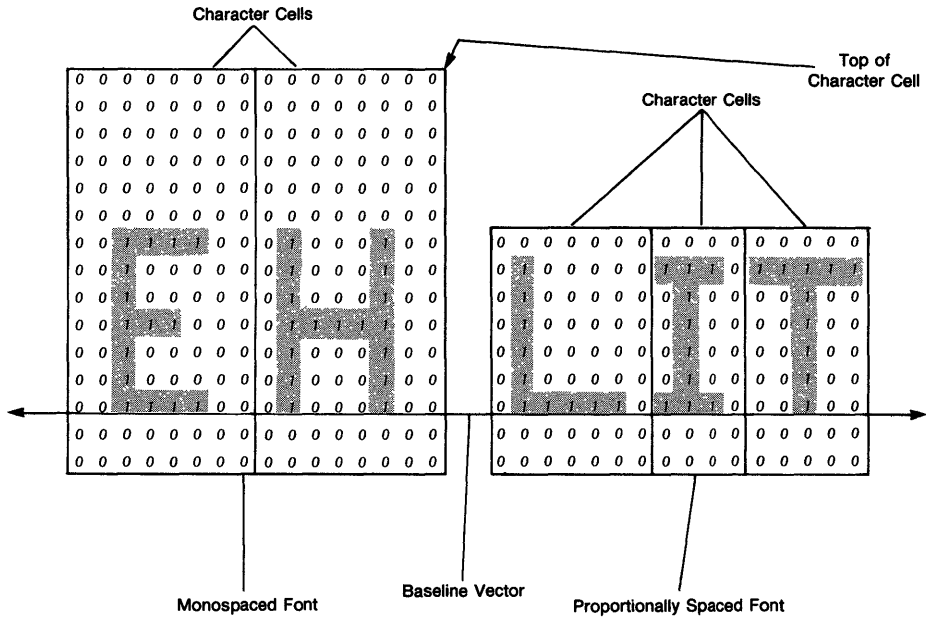
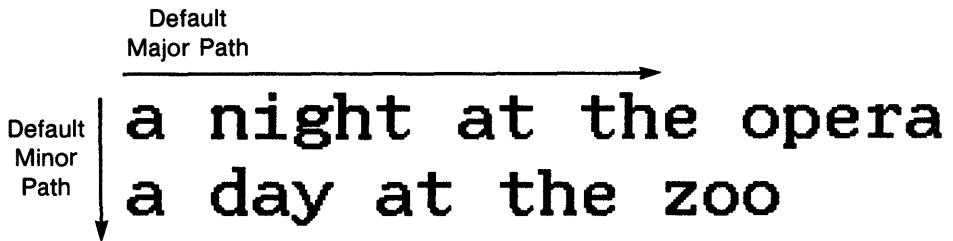


Figure 10-3 Text Path



10-4 Text Attributes

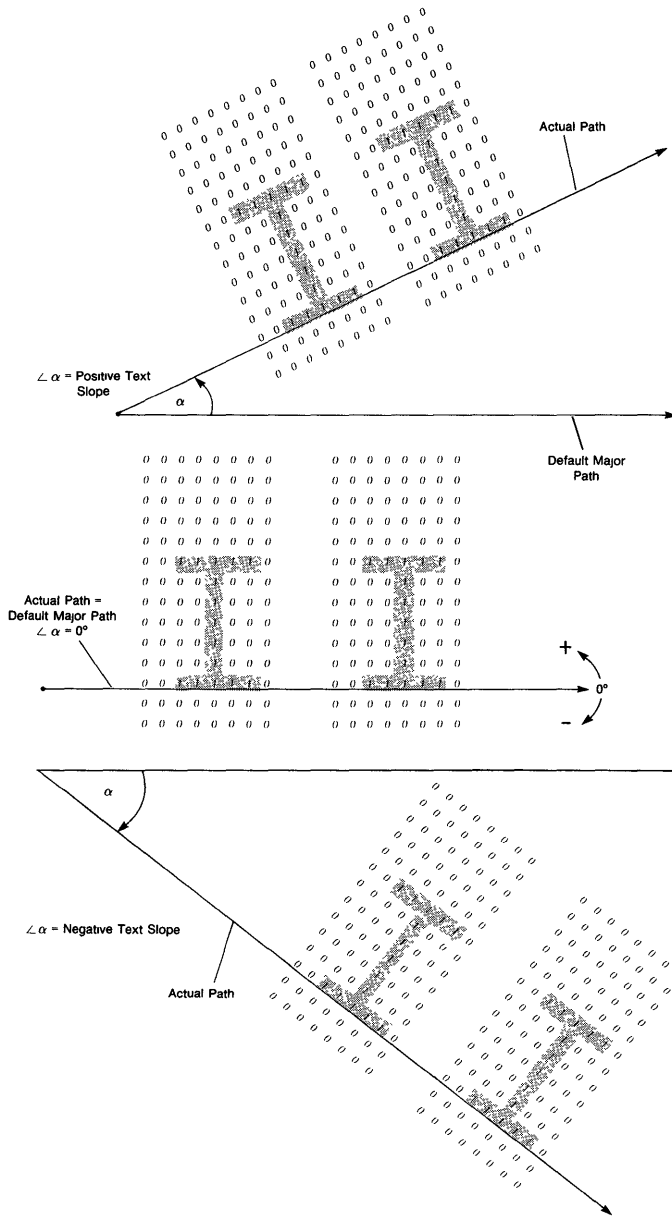
10.2.3 Character Strings

Characters within character strings also share a spatial relationship with other members of the string.

Text Slope

UIS draws all characters of a character string at the same angle with the respect to the major path. The *actual path* of text drawing is a line containing the baseline points of all the character cells in a character string. The angle between the actual path and the major path, measured counterclockwise is called the angle of *text slope*. UIS can draw text at any angle from 0 to 360 degrees. Figure 10-4 describes how text slope can be manipulated.

Figure 10-4 Text Slope



10-6 Text Attributes

Text Margins

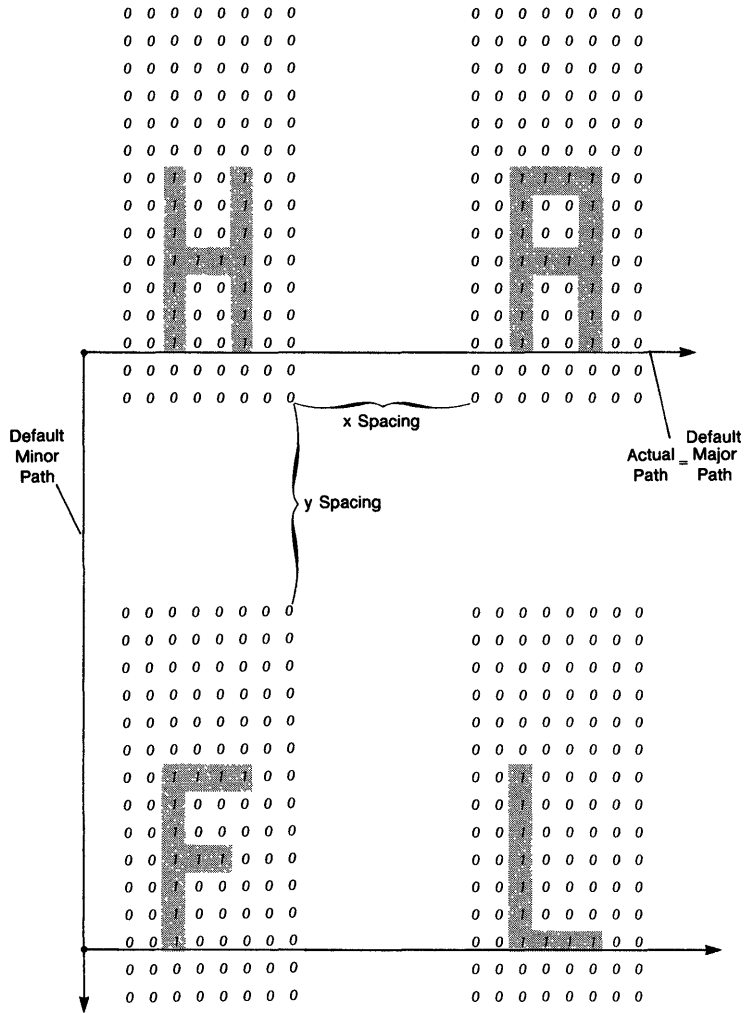
Character strings are drawn along the actual path of text drawing within certain explicit or implicit boundaries called *margins*. The implied text margin for all text output is the minor text path when the angle of text slope is 0 degrees. The programming interface lets you set explicit text margins that are always parallel to the implied margins.

Character Spacing

Spacing between characters and lines can be increased uniformly throughout the character string through the use of x and y spacing factors. The size of the characters remains constant space between them diminishes or increases.

Figure 10-5 shows how text path affects character spacing.

Figure 10-5 Character Spacing

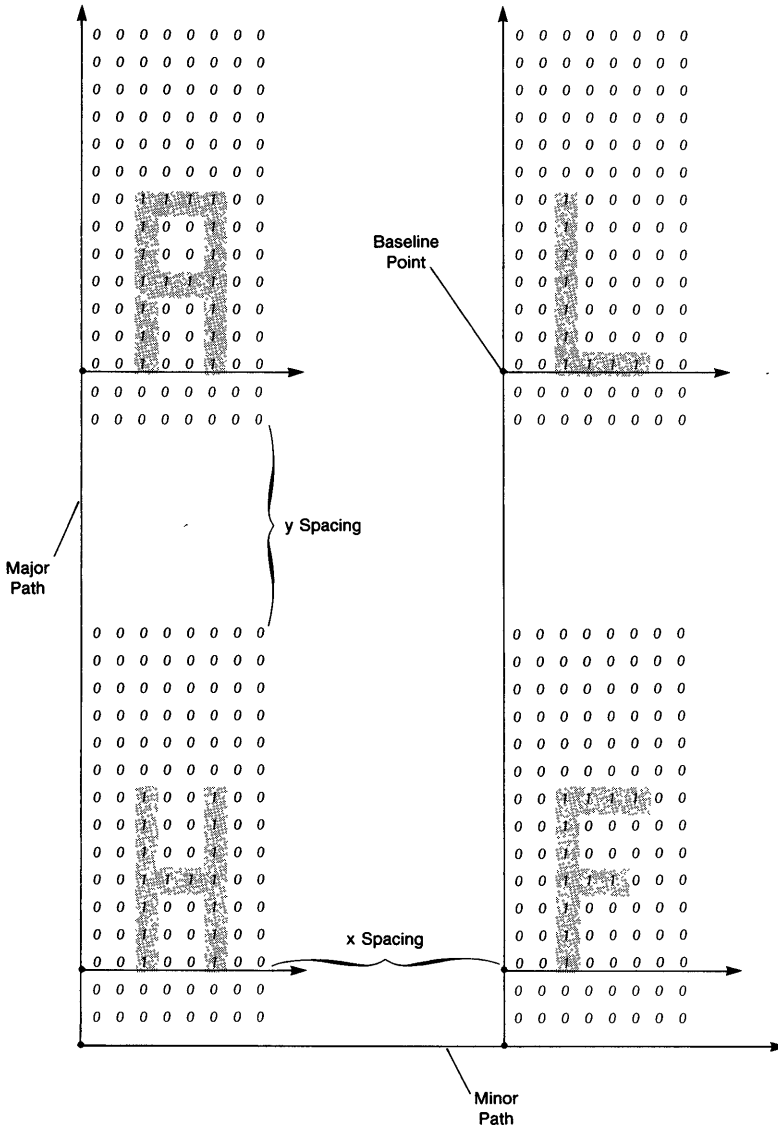


ZK-5356-86

(Continued on next page)

10-8 Text Attributes

Figure 10-5 (Cont.) Character Spacing



Text Formatting

Character strings can be arranged on a line in many ways through *justification*. Formatted character strings are drawn as follows: (1) flush against the left margin, (2) flush against the right margin, (3) centered between the margins, and (4) both right and left justified or *fully justified*.

10.2.4 Character Cell

The components of a character cell share a spatial relationship with each other. The orientation and shape of a single character cell in the virtual display can be altered through character rotation, slanting, and scaling. These attributes when modified alter the character cell with respect to its baseline vector. For example a scaled character may have its height modified changing the height-relationship. The resulting letter may appear “squat” or vertically elongated.

Rotating Characters

An individual character is rotated about its baseline point. The angle of character rotation is the angle between the baseline vector and the actual path of text drawing measured counterclockwise. Figure 10-6 describes simple character cell rotation about the baseline point.

10-10 Text Attributes

Figure 10-6 Simple Character Rotation

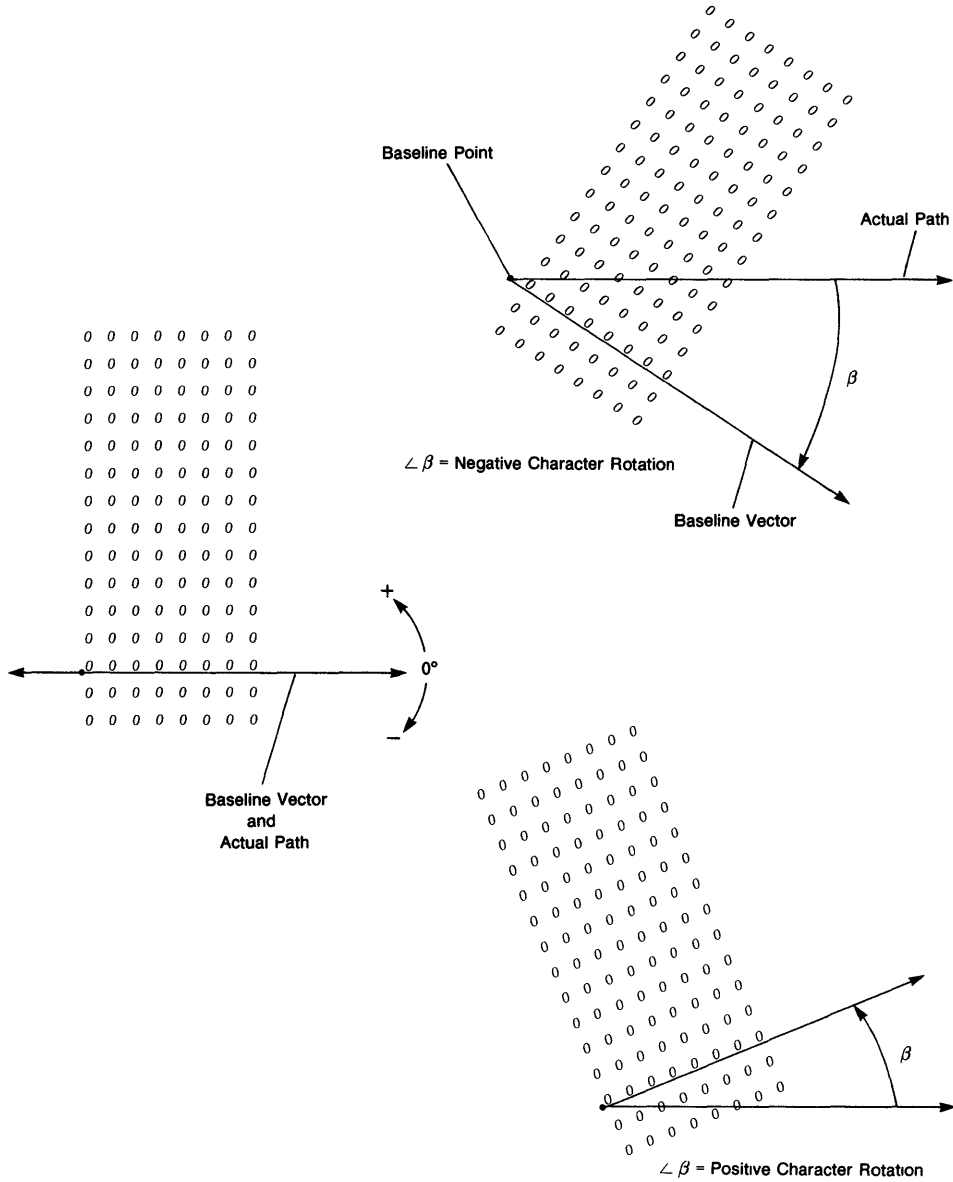
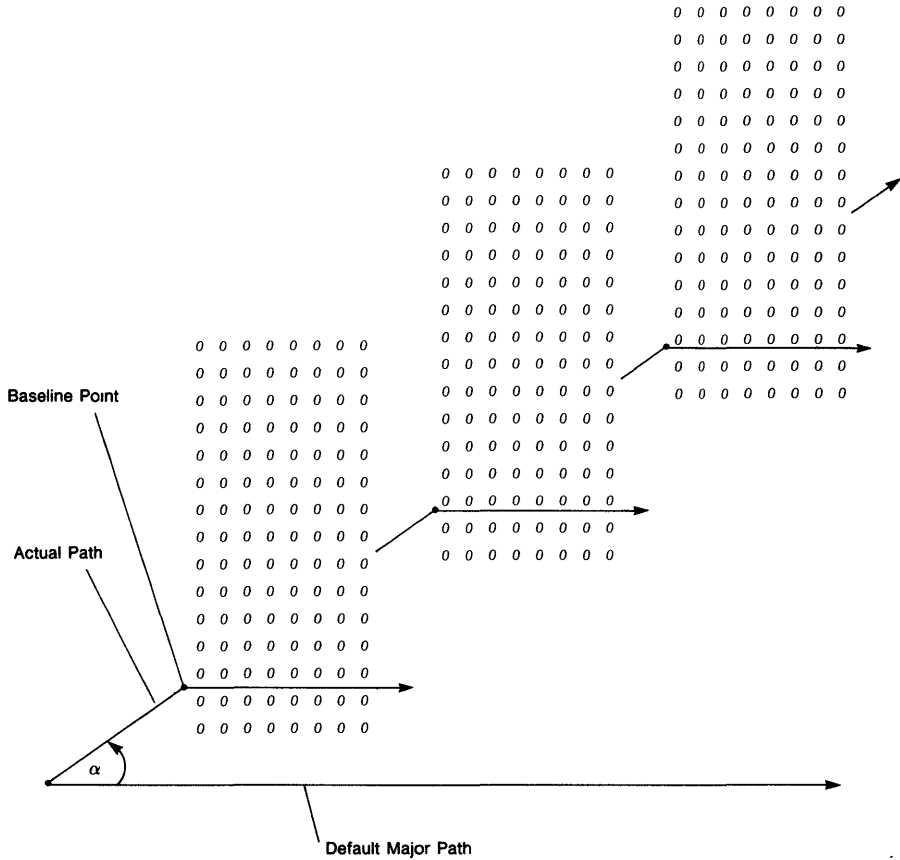


Figure 10-7 describes character rotation and text slope manipulation performed simultaneously.

Figure 10-7 Character Rotation with Slope Manipulation

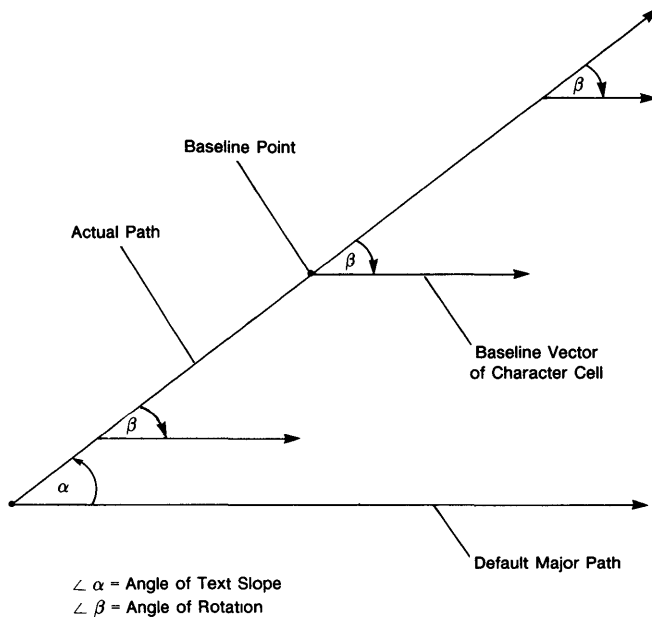


ZK-5276-86

(Continued on next page)

10-12 Text Attributes

Figure 10-7 (Cont.) Character Rotation with Slope Manipulation



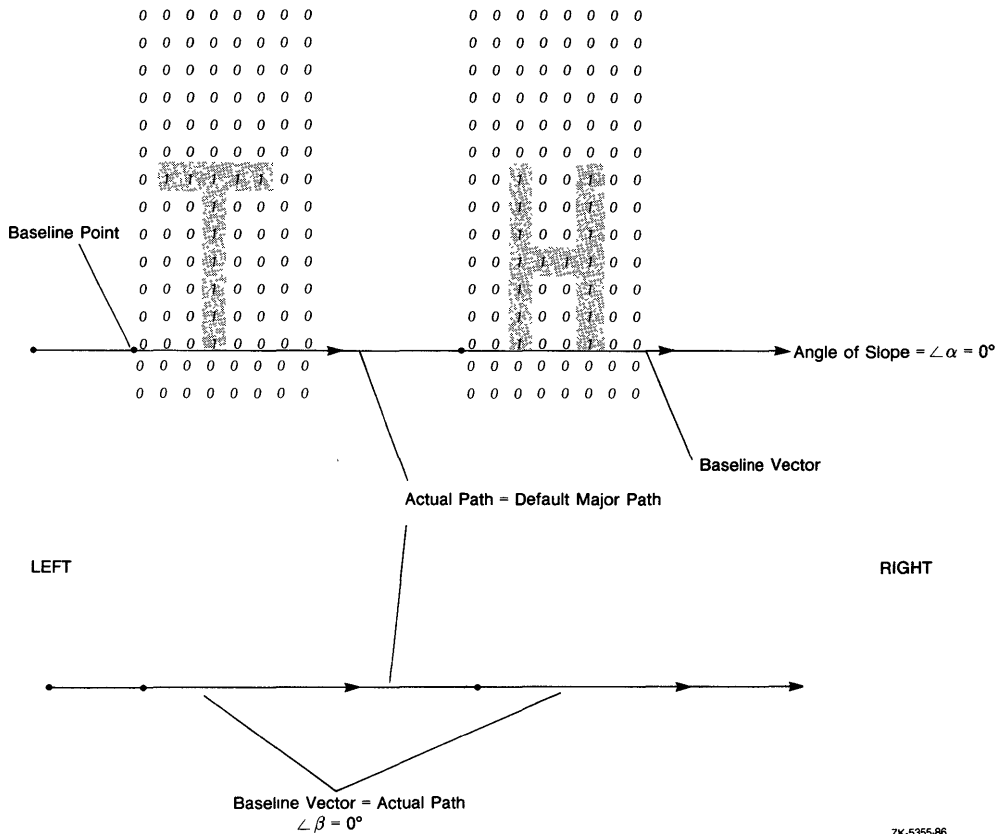
ZK 5273-86

When the character rotation attribute is set to 0 and text slope is 0 degrees, the angle of character rotation behaves in the following manner:

Slope (degrees)	Major Path	Rotation (degrees)
0	Left to right (default)	0
0	Bottom to top	-90
0	Right to left	-180
0	Top to bottom	-270

Figure 10-8 describes the appearance of the angle of rotation after text path modification when default character rotation is in effect.

Figure 10-8 Text Path Manipulation Without Character Rotation

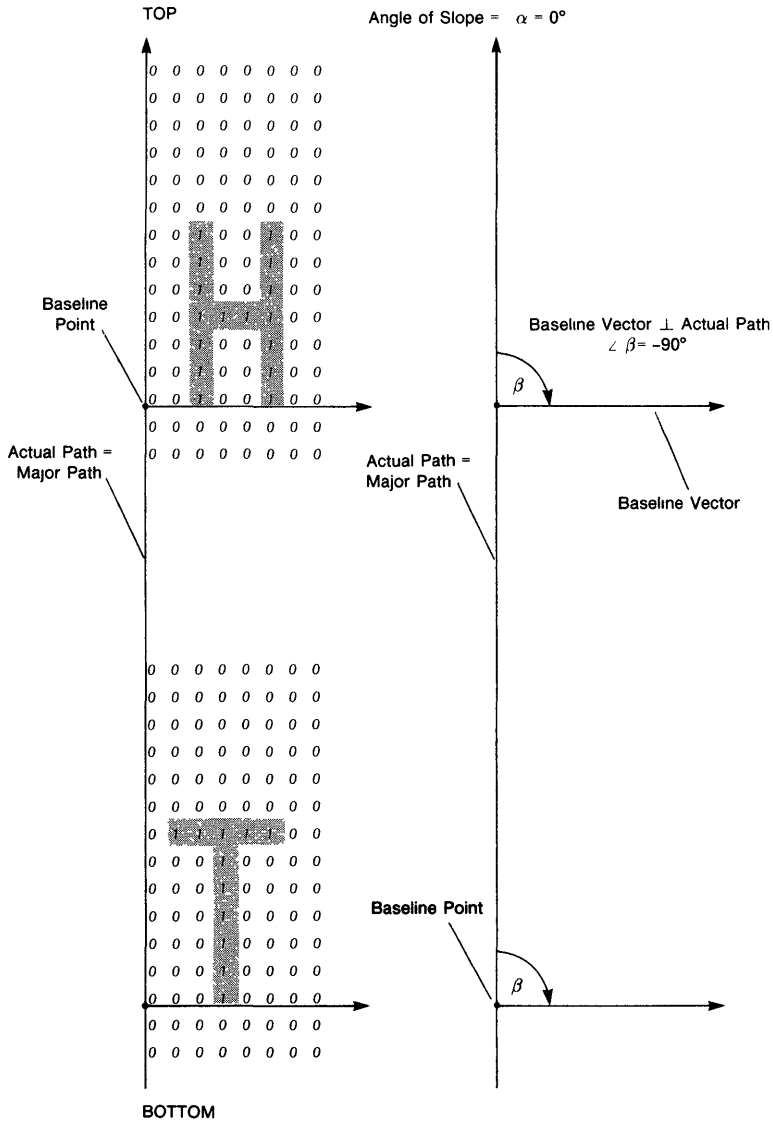


ZK-5355-86

(Continued on next page)

10-14 Text Attributes

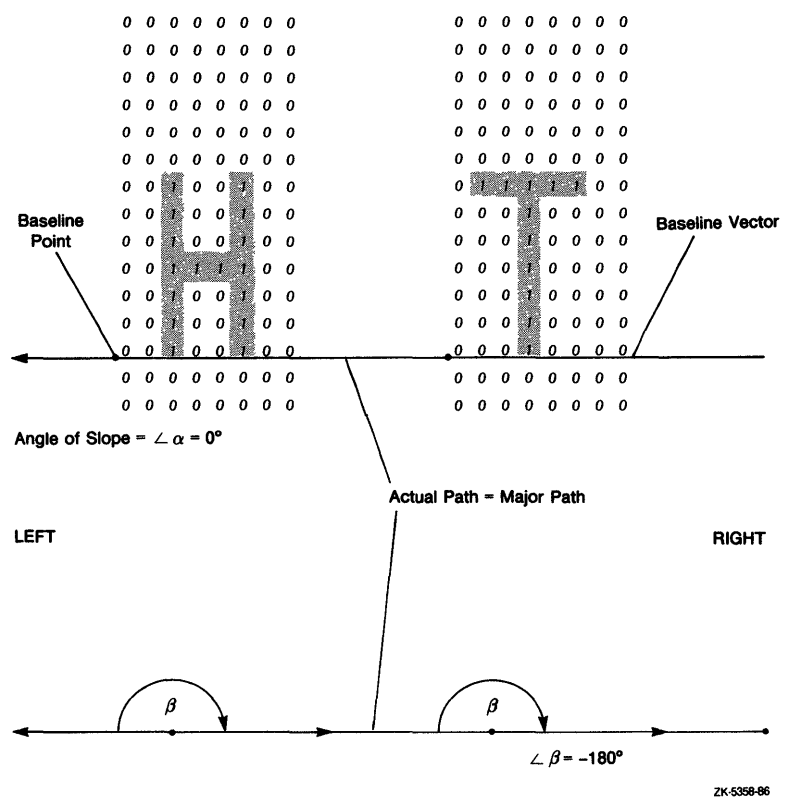
Figure 10-8 (Cont.) Text Path Manipulation Without Character Rotation



ZK 5361 86

(Continued on next page)

Figure 10-8 (Cont.) Text Path Manipulation Without Character Rotation

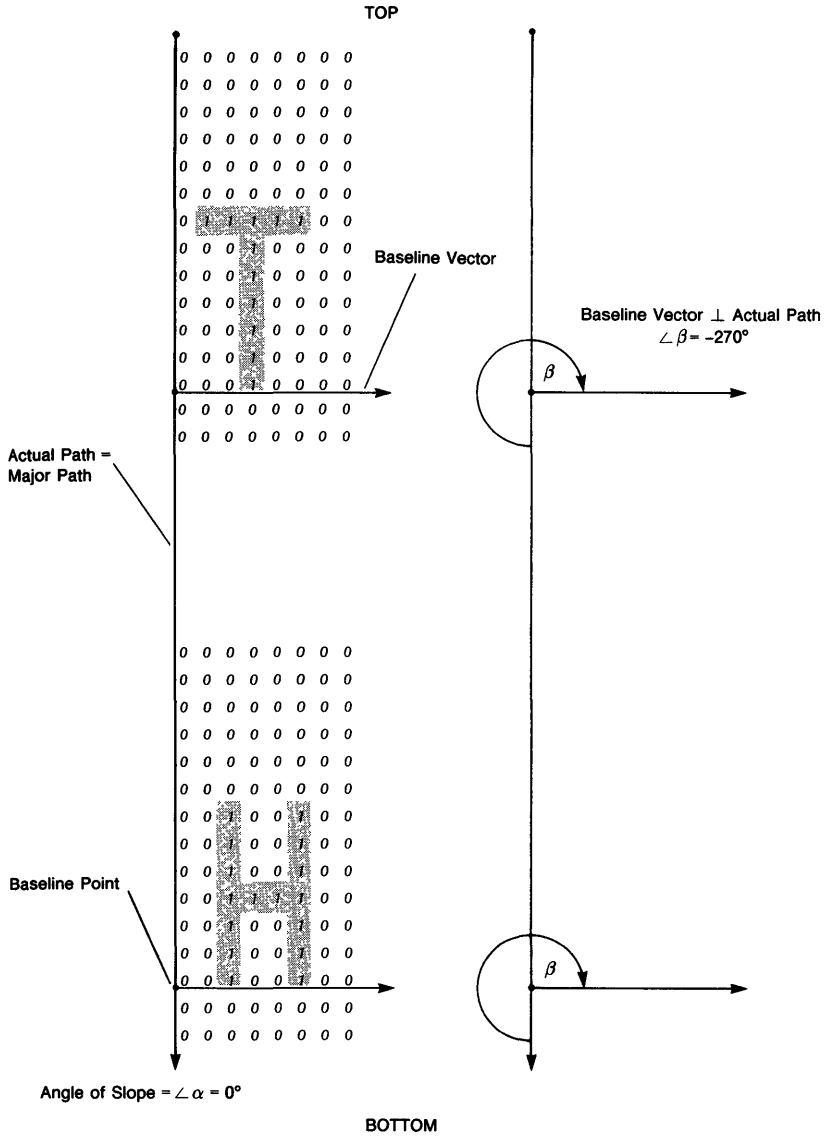


ZK-5358-86

(Continued on next page)

10-16 Text Attributes

Figure 10-8 (Cont.) Text Path Manipulation Without Character Rotation



Slanting Characters

Character slant is a measure of the angle between the up vector of the character cell and baseline vector. Character slant is 0.0 when this angle is 90 degrees. As slant increases, the up vector rotates clockwise toward the baseline vector, until at a slant of 90 degrees, the two vectors coincide. Figure 10-9 show a slanted character cell where the actual path and the default major path form an angle of 0 degrees.

10-18 Text Attributes

Figure 10-9 Character Slanting

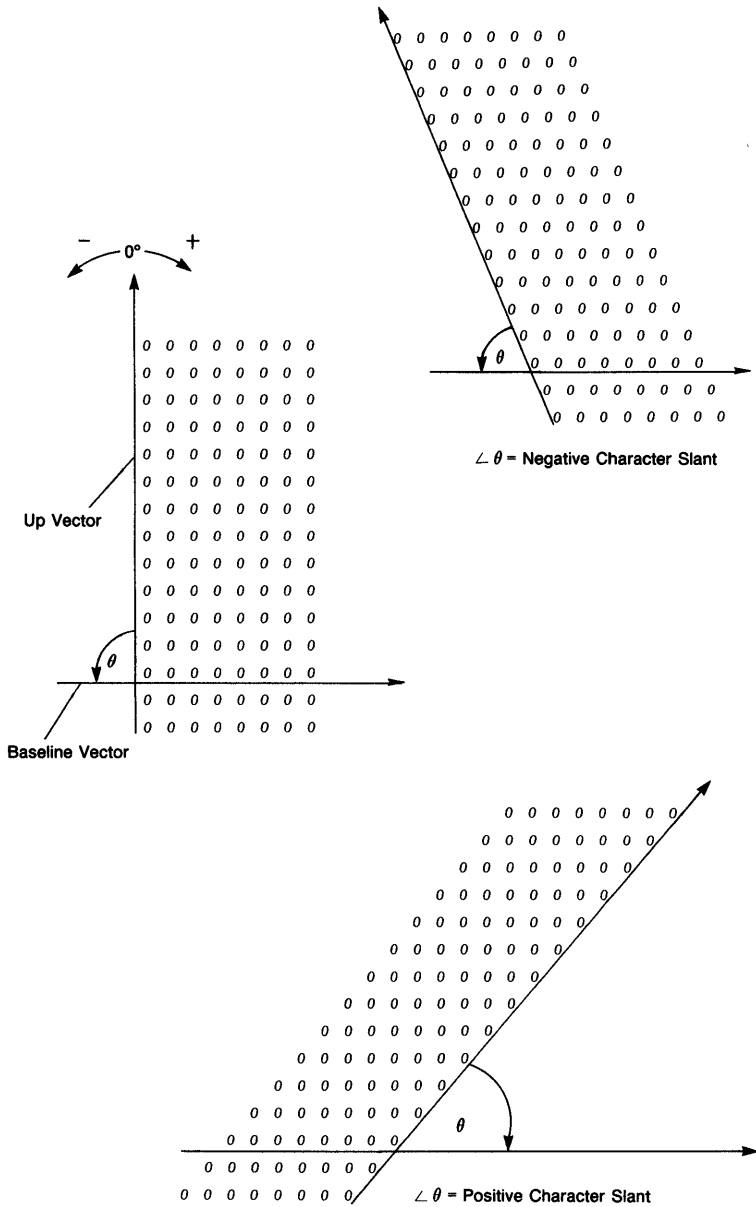
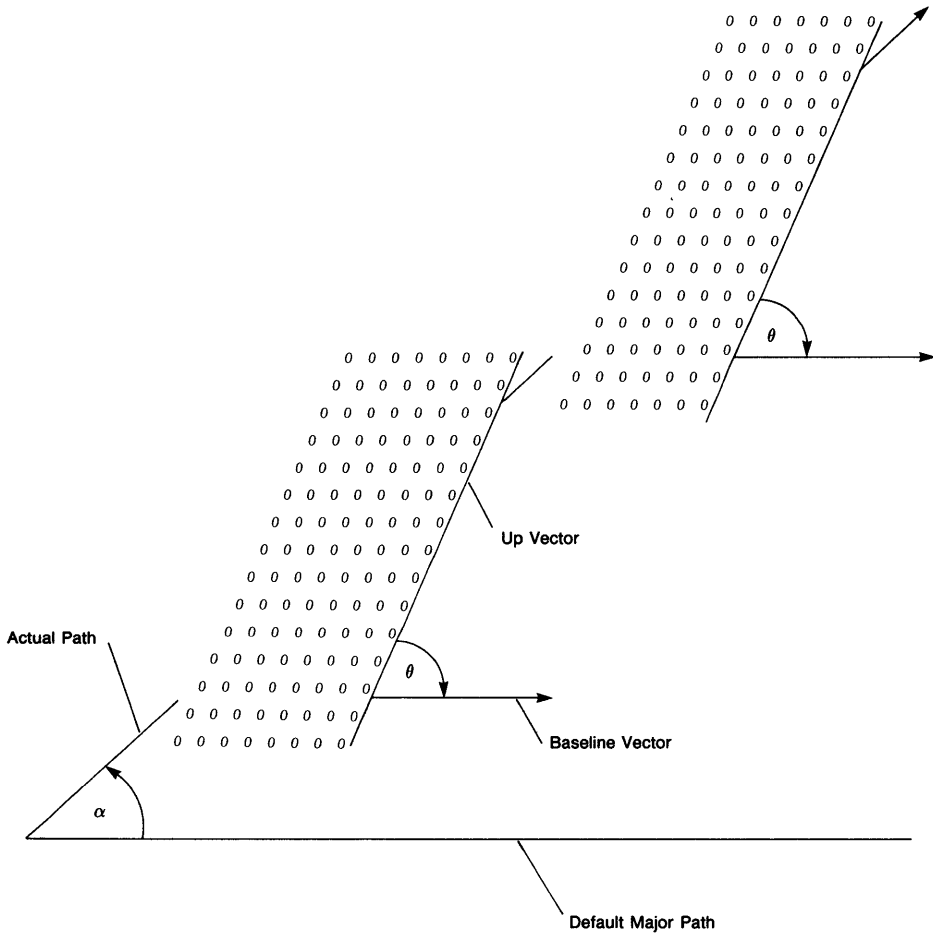


Figure 10-10 shows character slanting, character rotation, and text slope operations performed simultaneously on two character cells.

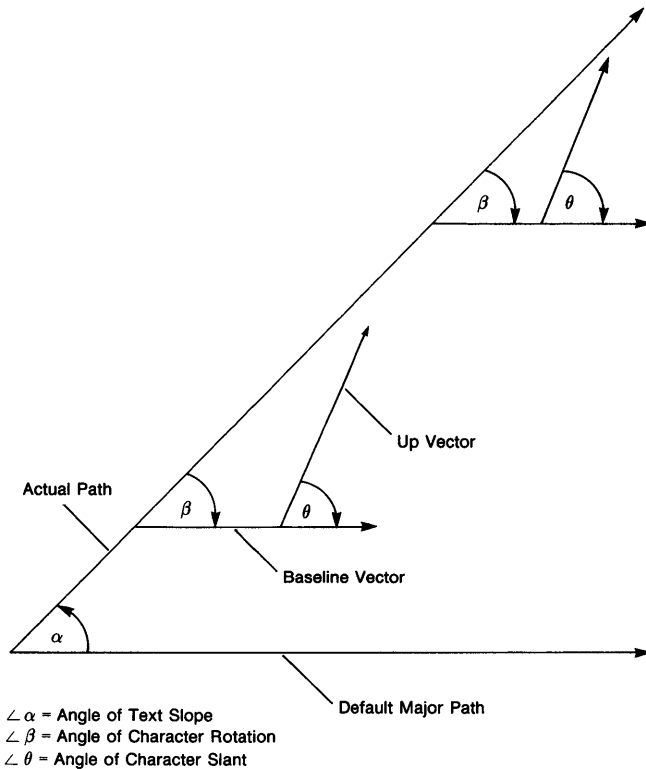
Figure 10-10 Character Slanting and Rotation with Slope Manipulation



ZK-5272-86

(Continued on next page)

Figure 10-10 (Cont.) Character Slanting and Rotation with Slope Manipulation

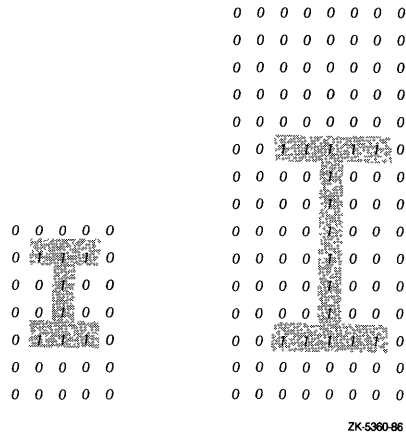


ZK-5274-86

Scaling Characters

Character scaling involves increasing or diminishing the size of the character cell. Scaling factors specify the world coordinate space in which the scaled character is drawn. The character cell is expanded or contracted to fill the specified space.

Figure 10-11 Character Scaling



ZK-5360-86

10.3 Using Text Attributes

As you can see, there are several attributes associated with text output. You are not limited to simply choosing from a library of fonts. For example, you can modify the appearance of any font through scaling and slanting and also alter the way in which the system draws the text in the virtual display using formatting modes and paths.

The following routines are not attribute modification routines but included here to illustrate other types of text manipulation.

Routine	Function
UIS\$NEW_TEXT_LINE	Moves the current text position along the minor text path
UIS\$SET_ALIGNED_POSITION	Sets the current text position at the upper-left corner of the character cell
UIS\$SET_POSITION	Sets the current text position at the baseline point of the character cell

These routines contain an **atb** argument which indicates that appropriate text attribute settings can modify their behavior.

10.3.1 Modifying Text Attributes

When you modify text attributes, you do not change the default attribute settings within attribute block 0 itself. You should think of attribute block 0 as a template of default settings and you are modifying a copy of this attribute block for use within your program. Attribute modification routines contain two arguments—the input attribute block number (**iatb**) and the output attribute block number (**oatb**). Table 10-1 lists all text attributes and their default settings.

Table 10-1 Default Settings of Text Attributes in Attribute Block 0

Text Attribute	Default Setting	Modification Routine
Character rotation	0.0	UIS\$SET_CHAR_ROTATION
Character size	Specified by the font	UIS\$SET_CHAR_SIZE
Character slant	0.0	UIS\$SET_CHAR_SLANT
Character spacing	0.0,0.0	UIS\$SET_CHAR_SPACING
Text formatting	Normal	UIS\$SET_TEXT_FORMATTING
Text margins	0.0,0.0	UIS\$SET_TEXT_MARGINS
Text path	Left to right (default major path) top to bottom (default minor path)	UIS\$SET_TEXT_PATH
Text slope	0.0	UIS\$SET_TEXT_SLOPE
Font	Multinational ASCII, 14-point, fixed pitch	UIS\$SET_FONT

Perform attribute modification using the following procedure:

1. Choose an appropriate attribute routine to modify the attribute.
2. Specify 0 as the **iatb** argument to obtain a copy of attribute block 0.
3. Specify a number from 1 to 255 as the **oatb** argument. The attribute block can then be referenced in subsequent UIS graphics and text routines or in any other attribute modification routine.

Graphics and text routines as well as UIS\$MEASURE_TEXT, UIS\$NEW_TEXT_LINE, and UIS\$SET_ALIGNED_POSITION reference modified attribute blocks in the **atb** argument. These routines are discussed later in this chapter.

10.4 Programming Options

You can modify text attributes within your application to change the font type, margin settings, and character spacing.

Fonts

You can change the font type of a line of text using `UIS$SET_FONT`. You must specify the desired font file name in the `font_id` argument. Font files reside in the directory `SYS$FONT`. The directory contains one file of fill patterns (`UIS$FILL_PATTERNS`) and 26 font files. You can choose between two types of fonts.

- Multinational character fonts — Contain international alphanumeric characters, including characters with diacritical marks.
- Technical fonts — Include scientific and mathematical symbols.

Font File Names

A standard 31-character file name identifies each font file as follows:

DTERMINM06K0OPG0001UZZZZO2A000

The first 16 bytes of this sample file name (representing unique font specifications) are explained in the following table.

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type family ID	TERMIN	Terminal
8	Spacing	M ₃₆	13 pitch (monospaced)
9-11	Type size	06O ₃₆	24 points (240 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Refer to Appendix C for more information about UIS fonts.

NOTE: You can define logical names to represent font file names.

10-24 Text Attributes

Font File Types

The following table lists sample font file names and their device-dependent font file types.

System	Font File Name
Mutinalional Character Set Fonts	
Monochrome	DTERMINM06OK00PG0001UZZZZ02A000.VWS\$FONT
Intensity or color	DTERMINM06OK00PG0001UZZZZ02A000.VWS\$VAFONT
Technical Character Set Fonts	
Monochrome	DVWSVT0G03CK00GG0001QZZZZ02A000.VWS\$FONT
Intensity or color	DVWSVT0G03CK00GG0001QZZZZ02A000.VWS\$VAFONT

NOTE: Whenever you reference a font file name as in `UIS$SET_FONT`, you should **not** specify the directory `SYS$FONT` or the file type.

Setting the Text Margins

You can set the left and right margins with `UIS$SET_TEXT_MARGINS`.

Setting the Text Formatting Mode

There are four text formatting modes—left justification, right justification, center justification, and full justification. The text formatting modes are set using `UIS$SET_TEXT_FORMATTING`.

NOTE: `UIS$SET_TEXT_FORMATTING` does not automatically wrap long lines of text.

Setting the Character Spacing

You can alter the spacing between character, or *kerning*, or the spacing between lines, also known as *leading*, with `UIS$SET_CHAR_SPACING`.

New Text Lines

When you are writing text and you need to move to a new line, use `UIS$NEW_TEXT_LINE`. When you create a new line of text, the current position becomes the beginning of the new line. When used in conjunction with `UIS$SET_CHAR_SPACING`, you can manipulate the spacing between lines, or *leading*.

Character Rotation

You can rotate characters about a pivotal point called the baseline point from 0 to 360 degrees using `UIS$SET_CHAR_ROTATION`.

Aligning Text Along the Baseline and Top of Character Cell

You can align text along the baseline vector using `UIS$SET_POSITION` or along the upper-left corner of the character cell using `UIS$SET_ALIGNED_POSITION`.

Specifying Character Slant

You can specify the angle relative to the text baseline vector by which text is to be slanted using `UIS$SET_CHAR_SLANT`.

Specifying Character Scaling

You can specify the width and height for characters in a font using `UIS$SET_CHAR_SIZE`.

Specifying Slope of the Text Baseline

You can specify the angle of the actual path of text drawing relative to the major path using `UIS$SET_TEXT_SLOPE`.

Specifying the Text Path

You can specify the direction of text drawing with `UIS$SET_TEXT_PATH`. There are four directions in which text can be drawn: (1) left to right, (2) right to left, (3) bottom to top, and (4) top to bottom. You must use these direction in the context of a major text drawing path and a minor text drawing path. The major path of text drawing is the relationship between letters; the minor path is the relationship between lines.

10.4.1 Program Development I

Programming Objective

To draw the multinational character set fonts available in the directory `SYS$FONT` and to show how to move to a new text line.

Programming Task

1. Create a virtual display.
2. Create a display window and viewport.
3. Modify the font attribute in attribute block 0.
4. Move to the beginning of a new line using `UIS$NEW_TEXT_LINE` and the appropriate attribute setting.
5. Draw a line of text.
6. Repeat steps 3 through 5.

10-26 Text Attributes

Note that font file names used in the program TEXT_1 are logical names. Some examples of a font occupy two lines.

```
PROGRAM TEXT_1
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,30.0,30.0,20.0,10.0)
WD_ID1=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','FONTS')

CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_1') ❶
CALL UIS$TEXT(VD_ID,1,'The quality of mercy is not strained',
2      1.0,30.0) ❷

CALL UIS$SET_FONT(VD_ID,0,2,'MY_FONT_2')
CALL UIS$NEW_TEXT_LINE(VD_ID,2) ❸
CALL UIS$TEXT(VD_ID,2,'Long visits bring short compliments')

CALL UIS$SET_FONT(VD_ID,0,3,'MY_FONT_3')
CALL UIS$NEW_TEXT_LINE(VD_ID,3)
CALL UIS$TEXT(VD_ID,3,'Wise men make proverbs and fools')
CALL UIS$NEW_TEXT_LINE(VD_ID,3)
CALL UIS$TEXT(VD_ID,3,'repeat them')

CALL UIS$SET_FONT(VD_ID,0,4,'MY_FONT_4')
CALL UIS$NEW_TEXT_LINE(VD_ID,4)
CALL UIS$TEXT(VD_ID,4,'Je pense donc je suis')

CALL UIS$SET_FONT(VD_ID,0,5,'MY_FONT_5')
CALL UIS$NEW_TEXT_LINE(VD_ID,5)
CALL UIS$TEXT(VD_ID,5,'Do well and have well')

CALL UIS$SET_FONT(VD_ID,0,6,'MY_FONT_6')
CALL UIS$NEW_TEXT_LINE(VD_ID,6)
CALL UIS$TEXT(VD_ID,6,'You cannot make a crab walk straight')

CALL UIS$SET_FONT(VD_ID,0,7,'MY_FONT_7')
CALL UIS$NEW_TEXT_LINE(VD_ID,7)
CALL UIS$TEXT(VD_ID,7,'Great minds think alike')

CALL UIS$SET_FONT(VD_ID,0,8,'MY_FONT_8')
CALL UIS$NEW_TEXT_LINE(VD_ID,8)
CALL UIS$TEXT(VD_ID,8,'One today is worth two tomorrows')

CALL UIS$SET_FONT(VD_ID,0,9,'MY_FONT_9')
CALL UIS$NEW_TEXT_LINE(VD_ID,9)
CALL UIS$TEXT(VD_ID,9,'With Latin, a horse, and money, you may')
CALL UIS$NEW_TEXT_LINE(VD_ID,9)
CALL UIS$TEXT(VD_ID,9,'travel the world')
```

```

CALL UIS$SET_FONT(VD_ID,0,10,'MY_FONT_10')
CALL UIS$NEW_TEXT_LINE(VD_ID,10)
CALL UIS$TEXT(VD_ID,10,'Whispered words are heard afar')

CALL UIS$SET_FONT(VD_ID,0,11,'MY_FONT_11')
CALL UIS$NEW_TEXT_LINE(VD_ID,11)
CALL UIS$TEXT(VD_ID,11,'Et tu, Brute?')
CALL UIS$NEW_TEXT_LINE(VD_ID,11)
CALL UIS$TEXT(VD_ID,11,'Per ardua astra')

CALL UIS$SET_FONT(VD_ID,0,12,'MY_FONT_12')
CALL UIS$NEW_TEXT_LINE(VD_ID,12)
CALL UIS$TEXT(VD_ID,12,'Velut arbor aevo')

CALL UIS$SET_FONT(VD_ID,0,13,'MY_FONT_13')
CALL UIS$NEW_TEXT_LINE(VD_ID,13)
CALL UIS$TEXT(VD_ID,13,'One mule scrubs another')

CALL UIS$SET_FONT(VD_ID,0,14,'MY_FONT_14')
CALL UIS$NEW_TEXT_LINE(VD_ID,14)
CALL UIS$TEXT(VD_ID,14,'Life is just a bowl of cherries')

PAUSE

END

```

The font attribute in attribute block 0 is modified in fourteen calls to `UIS$SET_FONT` ❶. There now exists an attribute block containing a modified font attribute for each font in `SYS$FONT`. These attribute blocks are identified by their output attribute block number when they were created.

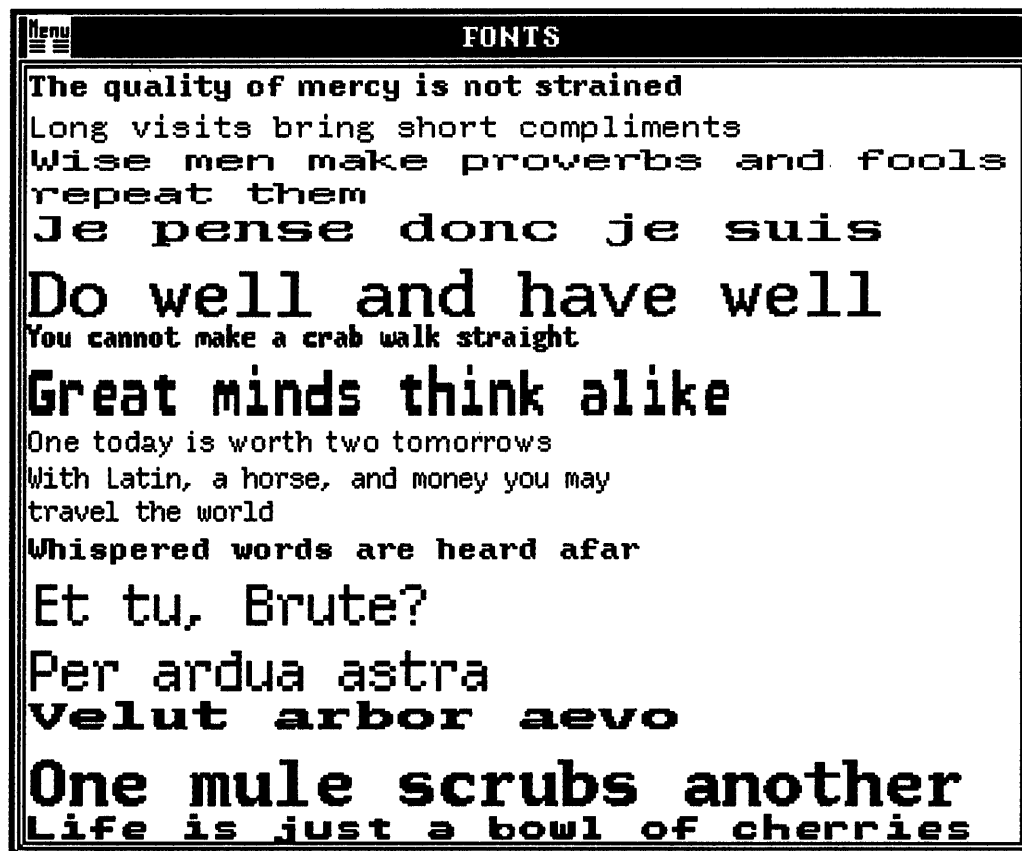
The `atb` argument of `UIS$TEXT` ❷ uses the appropriate attribute block number to generate text in the desired font.

A call to `UIS$NEW_TEXT_LINE` ❸ causes each new line of text to begin on a new line at the left margin.

10.4.2 Calling `UIS$SET_FONT` and `UIS$NEW_TEXT_LINE`

Once again, note the positional order of the attribute routines. Attribute routines modify the attribute block used by the routine creating the graphic object and, therefore, must precede that routine. The attribute routine and the output routine must reference the same attribute block. Figure 10-12 contains examples of each of the UIS fonts.

Figure 10-12 UIS Fonts



ZK-4546-85

Refer to Appendix C for a listing of UIS fonts.

10.4.3 Program Development II

Programming Objective

To increase character and line spacing in two lines of text.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with a title.
3. Draw a line of text using the default character spacing factor.
4. Modify the character and line spacing factors using `UIS$SET_CHAR_SPACING`.
5. Draw a line of text using the modified spacing attribute.
6. Move to the beginning of a new line using `UIS$NEW_TEXT_LINE` with the modified spacing attribute.
7. Repeat steps 3 through 5.

```

PROGRAM SPACE_1
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,40.0,40.0,18.0,6.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','KERNING AND LEADING')

CALL UIS$SET_FONT(VD_ID,0,15,'MY_FONT_1') ❶
CALL UIS$TEXT(VD_ID,15,'The best mirror is an old friend',0.0,40.0) ❷
CALL UIS$NEW_TEXT_LINE(VD_ID,15) ❸
CALL UIS$SET_CHAR_SPACING(VD_ID,15,16,3.0,3.0) ❹
CALL UIS$TEXT(VD_ID,16,'The best mirror is an old friend') ❺
CALL UIS$NEW_TEXT_LINE(VD_ID,16) ❻
CALL UIS$TEXT(VD_ID,15,'In the coldest flint there is hot fire')

CALL UIS$NEW_TEXT_LINE(VD_ID,15)
CALL UIS$TEXT(VD_ID,16,'In the coldest flint there is hot fire')

PAUSE

END

```

A call to `UIS$SET_FONT` ❶ sets the font attribute. The attribute block containing the newly modified font attribute is assigned the number 15. The logical name `MY_FONT_1` denotes a font that is used throughout the program.

The first line of text is drawn in the appropriate font ❷. The text is drawn at the location in the virtual display specified in `UIS$TEXT`.

10-30 Text Attributes

When the next line of text is written, `UIS$NEW_TEXT_LINE` references attribute block number 15 ③. `UIS$NEW_TEXT_LINE` uses the characteristics of the new font to determine proper line spacing. If you had used attribute block number 0, `UIS$NEW_TEXT_LINE` would use the characteristics of the default font. In that case, the descenders of letters in the previous line and the ascenders of the letters of the new line might crash into each other or obscure portions of letters in either line. Therefore, you should call `UIS$NEW_TEXT_LINE` using the appropriate attribute block number.

Attribute block 15 is further modified in a call to `UIS$SET_CHAR_SPACING` ④. Attribute block 15 containing the previously modified font attribute and now the newly modified character spacing attribute is assigned the number 16.

NOTE: Attribute block 15 still exists and can be referenced.

The character and line spacing attributes are set to a factor of 3. Characters are placed apart by a factor of 3 times their width. Lines of text are placed apart by a factor of 3 times the height of the character.

Text is drawn and spaced, character by character, according to the values specified in font attribute and the character spacing attribute in attribute block 16 ⑤. The character spacing component of the character spacing attribute, or *x factor* determines spacing between characters for left-to-right and right-to-left text paths.

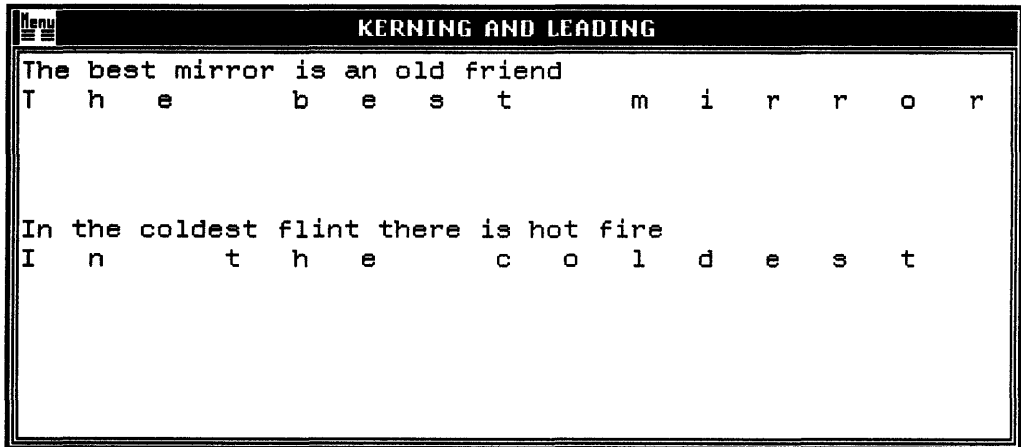
A call to `UIS$NEW_TEXT_LINE` ⑥ creates a new text line using attribute block number 16. `UIS$NEW_TEXT_LINE` uses the line spacing component of the character spacing attribute, or *y factor* to determine spacing between lines. The *y factor* is used for top-to-bottom and bottom-to-top text paths.

10.4.4 Calling `UIS$SET_CHAR_SPACING`

You can call character spacing in one line of the previous example by calling `UIS$SET_CHAR_SPACING` as shown here.

`UIS$SET_CHAR_SPACING` specified a spacing factor of 3. If you ran this program with the changes described above, your workstation screen would display the graphic objects shown in Figure 10-13.

Figure 10-13 Character and Line Spacing



ZK-4547-85

The line now extends beyond the right margin of the display viewport.

10.4.5 Program Development III

Programming Objective

To alignment along the top of the character cell and along the baseline vector.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with title.
3. Draw a horizontal line the width of the viewport.
4. Set the current position for text output at the leftmost point on the line using `UIS$SET_ALIGNED_POSITION`.
5. Choose a font and modify the font attribute block in attribute block 0.
6. Draw a line of text using the new font.

10-32 Text Attributes

- Repeat step 4 using `UIS$SET_POSITION`.
- Repeat steps 5 and 6.

```
PROGRAM SET_POS
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,40.0,40.0,18.0,5.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','TEXT ALIGNMENT')

CALL UIS$PLOT(VD_ID,0,0.0,35.0,40.0,35.0) ①
CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_7')
CALL UIS$SET_ALIGNED_POSITION(VD_ID,1,0.0,35.0) ②
CALL UIS$TEXT(VD_ID,1,'Never refuse a good offer') ③
CALL UIS$PLOT(VD_ID,0,0.0,20.0,40.0,20.0) ④
CALL UIS$SET_POSITION(VD_ID,0.0,20.0) ⑤
CALL UIS$SET_FONT(VD_ID,0,2,'MY_FONT_5')
CALL UIS$TEXT(VD_ID,2,'Weigh justly and sell dearly') ⑥

PAUSE
END
```

Two horizontal and parallel lines are drawn with `UIS$PLOT` ① ④.

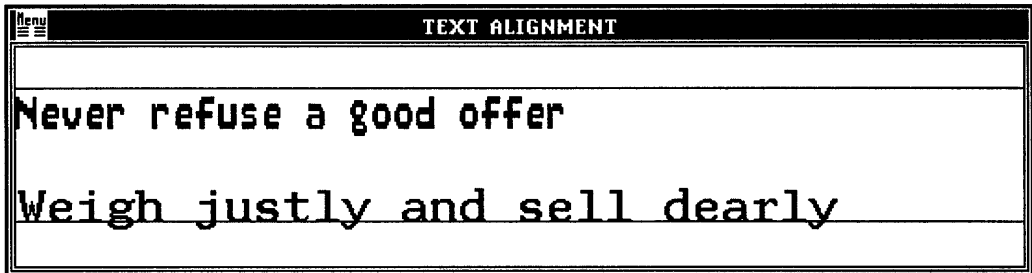
Both calls to `UIS$SET_ALIGNED_POSITION` and `UIS$SET_POSITION` ② ⑤ use the starting points of the respective lines to establish the current position for new text output unless the current position is specified in `UIS$TEXT`.

Text creation ③ ⑥ begins by default at the current position established in `UIS$SET_ALIGNED_POSITION` and `UIS$SET_POSITION`.

10.4.6 Calling `UIS$SET_POSITION` and `UIS$SET_ALIGNED_POSITION`

The first sentence shown in Figure 10-14 illustrates the alignment of text along the top of the character cell. The second sentence is aligned on the baseline vector.

Figure 10-14 Baseline and Top of Character Cell



ZK-4548-85

10.4.7 Program Development IV

Programming Objective

To draw characters at three different angles relative to the baseline vector.

Programming Tasks

1. Create a virtual display.
2. Create a display window and a viewport with a title.
3. Choose a font and modify the font attribute in attribute block 0.
4. Draw a character string at the default angle 0 degrees.
5. Modify the character slant attribute using `UIS$SET_CHAR_SLANT`.
6. Draw the character string again using the modified attribute block.
7. Repeat step 5 specify negative degrees.

The file name `MY_FONT_12` is a logical name for a font in `SY$FONT`.

```

PROGRAM SLANT
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,20.0,5.0,18.0,4.5)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','CHARACTER SLANTING')
CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_12') ❶
CALL UIS$TEXT(VD_ID,1,'Unslanted characters do not lean',0.1,5.0) ❷

```

10-34 Text Attributes

PAUSE

```
CALL UIS$SET_CHAR_SLANT(VD_ID,1,2,25.0) ③  
CALL UIS$TEXT(VD_ID,2,'Slanted characters lean forward',0.5,3.0)
```

PAUSE

```
CALL UIS$SET_CHAR_SLANT(VD_ID,1,3,-25.0) ④  
CALL UIS$TEXT(VD_ID,3,'Slanted characters lean backward',0.5,1.0)
```

PAUSE

END

A font is selected using `UIS$SET_FONT` ①. A text string is drawn using the default attribute setting in attribute block 0 ②.

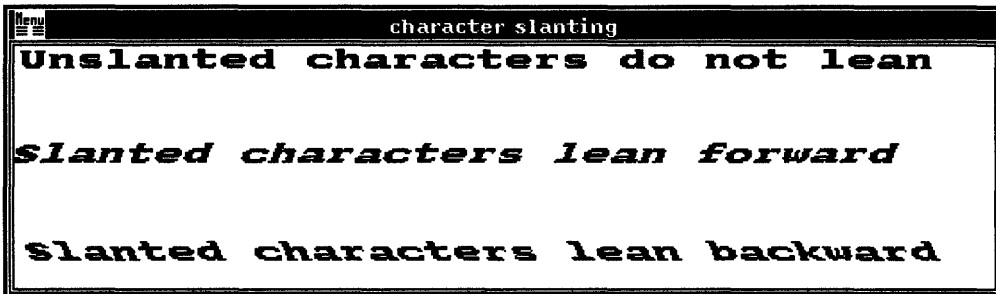
Next, the character slant attribute is modified ③ to specify a 25 degree shift to the right of a line perpendicular to the text baseline.

The character slant attribute is further modified ④ to specify a 25 degree shift to the left of a line perpendicular to the text baseline.

10.4.8 Calling `UIS$SET_CHAR_SLANT`

First, the character string is drawn at the default slant—0 degrees. Next, the character string is drawn twice slanting each character 25 degrees to the right of a line perpendicular to the text baseline and then 25 degrees to the left of that line.

Figure 10-15 Character Slanting



10.4.9 Program Development V

Programming Objective

To draw a character string whose actual path increases at 20-degree increments from 0 to 340 degrees.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Create DO loop that increases from 0 to 360 degrees by 20-degree increments.
 - Place the slope attribute modification routine `UIS$SET_TEXT_SLOPE` within the DO loop.
 - Place the text drawing routine `UIS$TEXT` within the DO loop.

The font file name `MY_FONT_13` is a logical name for a font in `SY$FONT`.

```

PROGRAM SLOPE
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,50.0,50.0,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','text slope')

CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_13') ①

DO I=0,340,20 ②
CALL UIS$SET_TEXT_SLOPE(VD_ID,1,2,FLOAT(I)) ③
CALL UIS$TEXT(VD_ID,2,' Slope!',25.0,25.0) ④
ENDDO ⑤

PAUSE
END

```

A font is selected and the default font attribute setting is modified using `UIS$SET_FONT` ①.

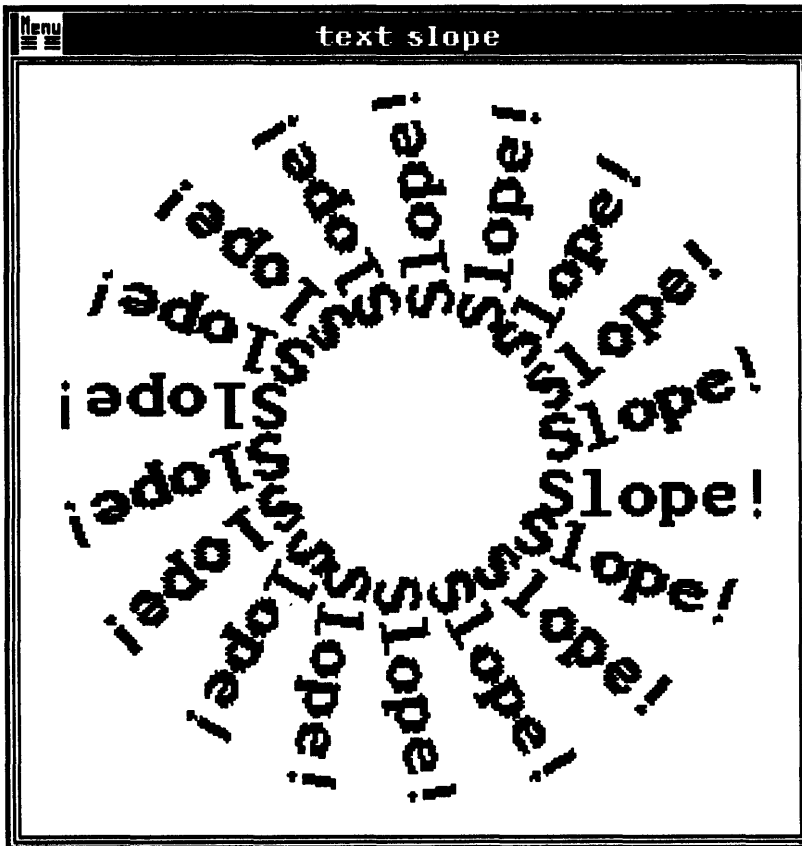
A DO loop is established ② ⑤. The counter *I* is initialized to 0 and will increase by increments of 20. The **angle** argument in `UIS$SET_TEXT_SLOPE` uses the value of *I* as the new text baseline attribute setting ③. The VAX FORTRAN function `FLOAT` changes the integer counter *I* to a real number ③.

Using `UIS$TEXT`, text strings are drawn from a central point (25.0,25.0) at 20-degree intervals ④.

10.4.10 Calling UIS\$SET_TEXT_SLOPE

Text strings are drawn at 20-degree intervals from 0 degrees to 360 degrees. The angle of each new text baseline increases by a multiple of 20. Text is drawn in a counterclockwise direction from the default horizontal baseline.

Figure 10-16 Manipulating the Text Baseline



ZK-5422-86

10.4.11 Program Development VI

Programming Objective

To rotate each character in order to offset text slope.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Create a DO loop.
4. Modify the attributes within the DO loop.

```

PROGRAM SLOPE_ROTATE
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,50.0,51.0,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION',
2      'TEXT SLOPE AND CHARACTER ROTATION')
CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_13')

DO I=0,340,20
CALL UIS$SET_TEXT_SLOPE(VD_ID,1,2,FLOAT(I))    ❶
CALL UIS$SET_CHAR_ROTATION(VD_ID,2,2,FLOAT(-I)) ❷
CALL UIS$TEXT(VD_ID,2,'  Rotate!',24.0,28.5)
ENDDO

PAUSE

END

```

This program is identical to the previous program SLOPE except that in addition to the text slope attribute we have modified the character rotation attribute.

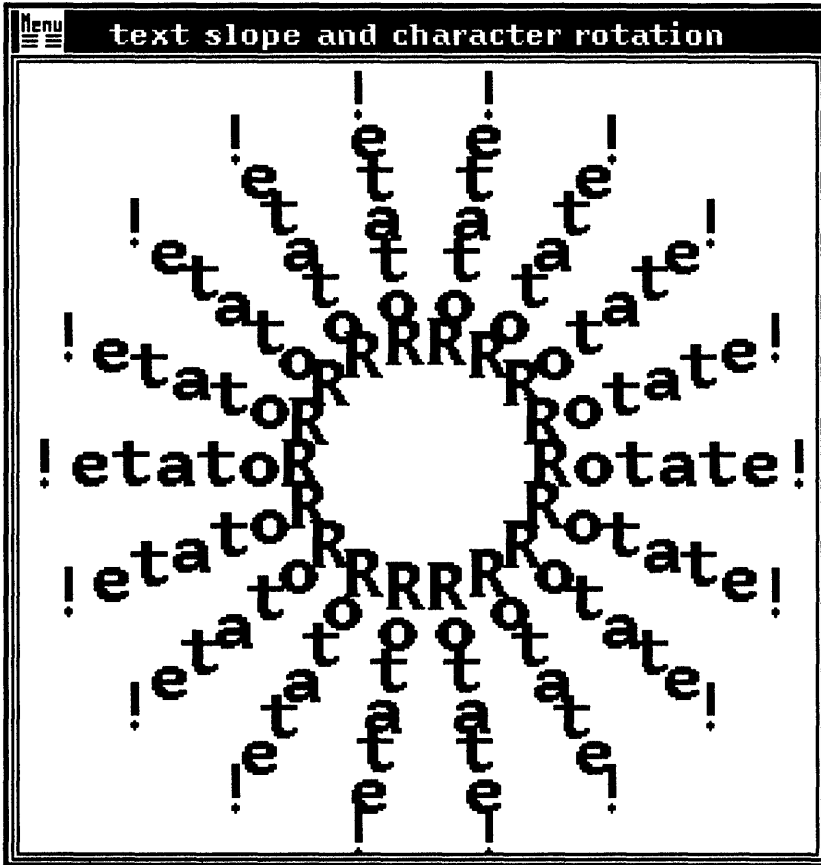
Within the DO loop, both attribute modification calls use the value of the counter *I* to increase the angles of text slope and character rotation for different purposes ❶ ❷.

For every 20-degree increase in the angle of text slope, the angle of character rotation of each character must be decremented by -20 degrees. Consequently, each character's baseline vector remains parallel with the default major path.

10.4.12 Calling UIS\$SET_CHAR_ROTATION

The program SLOPE_ROTATE draws a series of character strings from a center point from 0 to 360 degrees at 20-degree intervals. Because the angle of character rotation offsets exactly the angle of text slope, character maintain a readable orientation.

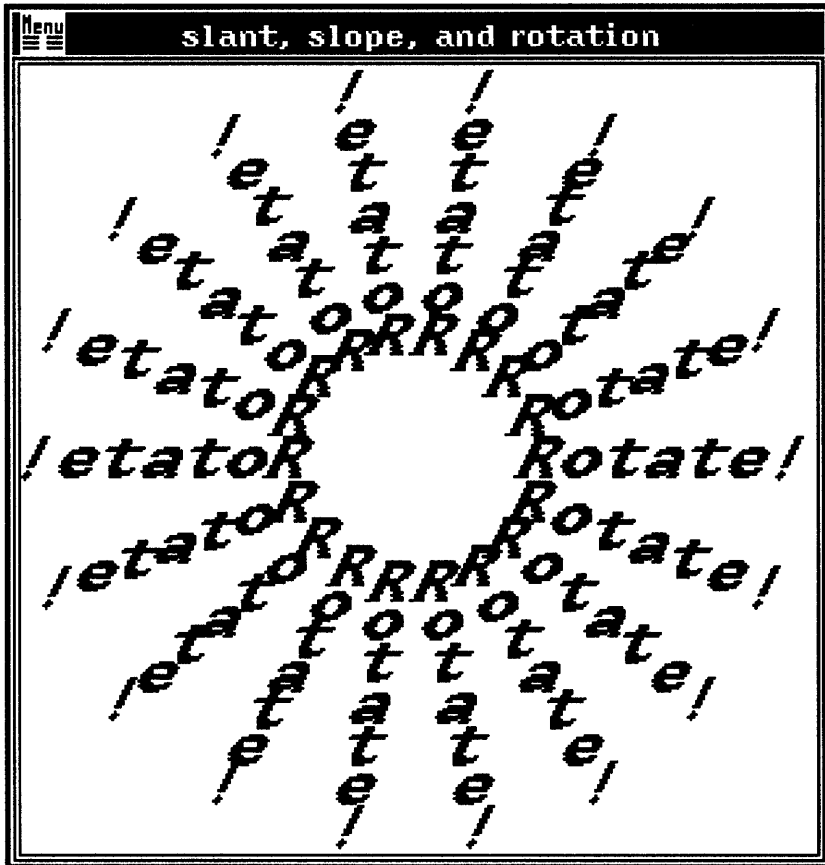
Figure 10-17 Character Rotation Without Slanting



ZK-5423-86

If you add a single call to modify the character slanting attribute, your viewport will display character rotation and slanting as the text slope from 0 to 360 degrees at 20-degree intervals as shown in Figure 10-18.

Figure 10-18 Character Rotation with Slanting



ZK-5424-86

10.4.13 Program Development VII

Programming Objective

To manipulate the width and height of character through scaling.

10-40 Text Attributes

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with title.
3. Draw a character string.
4. Increase the character size for width and height by 1.
5. Repeat steps 3 and 4.

Font names used in this program are logical names.

```
PROGRAM CHARSIZE
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL*4 WIDTH,HEIGHT

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,70.0,90.0,12.0,16.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','CHARACTER SCALING')

CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_1') ❶

CALL UIS$TEXT(VD_ID,1,'Great scott!',0.0,90.0) ❷
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,2.0,2.0) ❸
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,80.0) ❹
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,3.0,3.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,70.0)
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,4.0,4.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,60.0)
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,5.0,5.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,50.0)

CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,6.0,6.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,40.0)
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,7.0,7.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,30.0)
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,8.0,8.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,20.0)
CALL UIS$SET_CHAR_SIZE(VD_ID,1,2,,9.0,9.0)
CALL UIS$TEXT(VD_ID,2,'Great scott!',0.0,10.0)

PAUSE
END
```

A font is selected ❶.

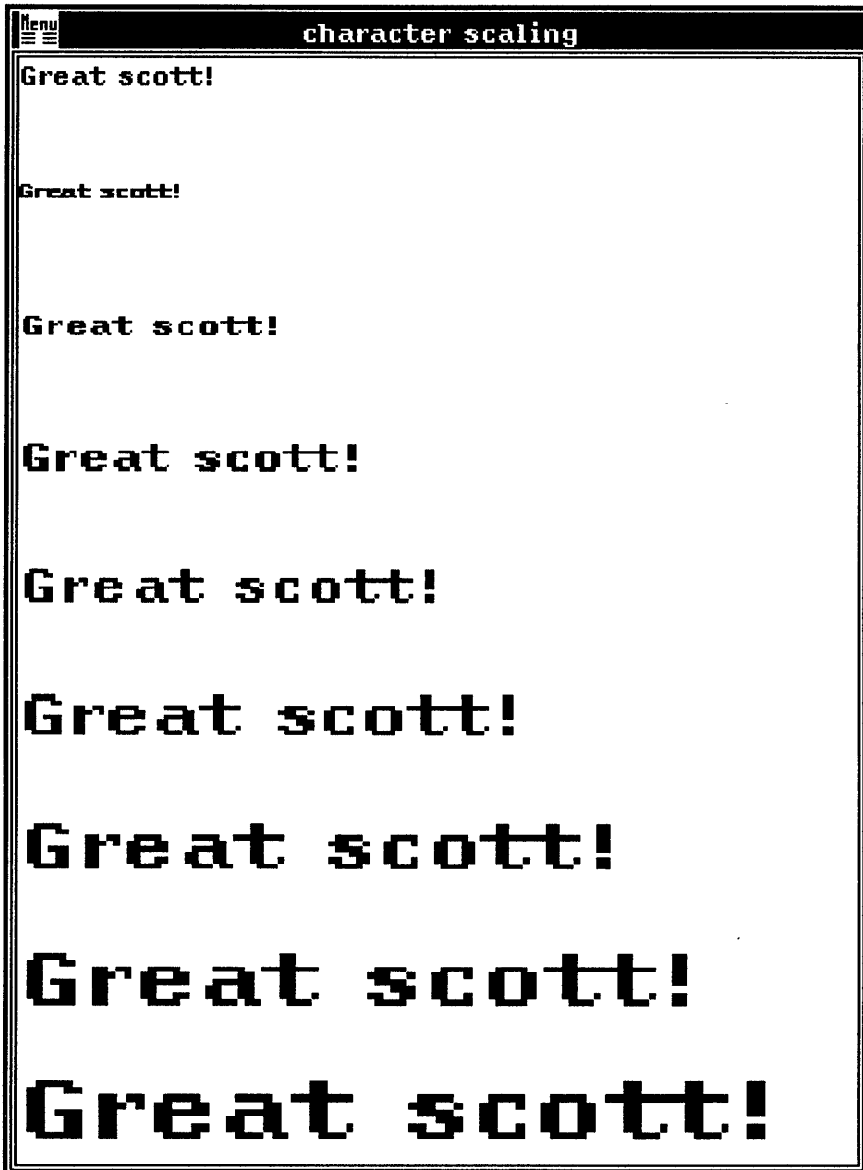
The unscaled character string Great scott! is drawn in the virtual display ❷.

The character string is redrawn as scaled text. The scale factors for the width and height are incremented ❸ each time the character string is drawn ❹.

10.4.14 Calling UIS\$SET_CHAR_SIZE

Figure 10-19 shows the character string increasing in height and width as the scale factors are incremented.

Figure 10-19 Manipulating Character Size



Chapter 11

Graphics and Windowing Attributes

11.1 Overview

This chapter discusses the following topics:

- Creating dashed lines
- Creating lines of varying widths
- Using fill patterns
- Using clipping rectangles

11.2 Using Graphics Attributes

Graphics attributes affect arc type, line width, line style, and the use of fill patterns.

11.2.1 Modifying Graphics and Windowing Attributes

When you modify graphics and windowing attributes, you do not change the default attribute settings within attribute block 0 itself. You should think of attribute block 0 as a template of default settings and you are modifying a copy of this attribute block for use within your program. Attribute modification routines contain two arguments—the input attribute block number (**iatb**) and the output attribute block number (**oatb**). Table 11–1 lists the default settings of graphics and windowing attributes.

11-2 Graphics and Windowing Attributes

Table 11-1 Default Settings of Graphics and Windowing Attributes

Attribute	Default Setting	Modification Routine
Arc type	Open	UIS\$SET_ARC_TYPE
Fill pattern	Off	UIS\$SET_FILL_PATTERN
Line style	Solid	UIS\$SET_LINE_STYLE
Line width	1.0 (unscaled)	UIS\$SET_LINE_WIDTH
Clipping rectangle	Off	UIS\$SET_CLIP

Perform attribute modification using the following procedure:

1. Choose an appropriate attribute routine to modify the attribute.
2. Specify 0 as the **iatb** argument to obtain a copy of attribute block 0.
3. Specify a number from 1 to 255 as the **oatb** argument. The attribute block can then be referenced in subsequent UIS graphics and text routines or in any other attribute modification routine.

Graphics and text routines reference modified attribute blocks in the **atb** argument as well as `UIS$MEASURE_TEXT`, `UIS$NEW_TEXT_LINE`, and `UIS$SET_ALIGNED_POSITION`.

11.2.2 Programming Options

Depending on the graphic object to be created—a line, a polygon, an ellipse, or circle—there are several attributes to choose from.

Fill Patterns

Fill patterns are used to add shading to geometric figures displayed on the workstation screen. Fill patterns are most often used to accentuate portions of a pie graph. Fill patterns range in coloration from light to heavy. Typically, light fill patterns connote light activity or minimal density in graphs. Heavy fill patterns connote the opposite meaning—heavy activity or maximum density.

You can also create your own fill pattern by selecting a character from any UIS font to serve as a fill pattern glyph.

All fill patterns are stored together in a font file in the directory `SYS$FONT`. For your convenience, this file name has been converted to a logical name `UIS$FILL_PATTERNS`.

Select a fill pattern in the following manner:

1. Using `UIS$SET_FONT`, specify `0` to select a copy of attribute block `0` to modify or specify the number of a previously modified attribute block as the input attribute block.
2. Assign an output attribute block number to this newly modified attribute block in `UIS$SET_FONT`. This attribute block number allows you to keep track of attributes. You can also modify some other element in this attribute block later on.
3. Specify the name of the fill pattern file in `UIS$SET_FONT`. Use the predefined logical name for the fill pattern file is `UIS$FILL_PATTERNS`.

If you wish to use a character from a font other than the default fill pattern file as fill pattern glyph, specify the appropriate font name.

4. Using `UIS$SET_FILL_PATTERN`, specify the actual fill pattern using a `UIS` symbol in the argument **index**. A `UIS` symbol in the form `PATT$C_xxxx` exists for each fill pattern and serves an index of each fill pattern in the file. The symbolic constant represents a hexadecimal offset indicating the fill pattern's position in the font file.

If you are creating a fill pattern from a `UIS` font other than the default fill pattern file, specify the ASCII code of the desired character in the **index** of `UIS$SET_FILL_PATTERN`.

NOTE: To disable fill patterns **without** modifying the fill pattern attribute, do not specify the **index** argument in `UIS$SET_FILL_PATTERN`.

Refer to Section 6.4 for more information about `UIS` constants.

Setting the Arc Type

Perhaps you want to draw a pie chart. You can draw chords or request that no chord be drawn using `UIS$SET_ARC_TYPE` and by specifying one of the constants shown in the following table.

Arc Type	Description
<code>UIS\$C_ARC_OPEN</code>	Does not draw any chords
<code>UIS\$C_ARC_PIE</code>	Draws a line from both end points of the arc to the center position
<code>UIS\$C_ARC_CHORD</code>	Draws a line connecting the end points of the arc

Remember that fill patterns are not drawn in the arc when the arc type attribute is specified as `OPEN`.

11-4 Graphics and Windowing Attributes

Line Width

You can increase the apparent thickness of lines displayed on the workstation screen with `UIS$SET_LINE_WIDTH`. Note that this routine affects the thickness of lines created with `UIS$LINE`, `UIS$LINE_ARRAY`, `UIS$PLOT`, `UIS$PLOT_ARRAY`, and `UIS$ELLIPSE` only.

Line Style

Occasionally, a solid line is not exactly what you need. You can create dots, hyphens, and dashes with `UIS$SET_LINE_STYLE`.

11.2.2.1 Program Development I

Programming Objective

To draw the different arc types and to demonstrate their use with fill patterns.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with title.
3. Modify the arc type attribute using the chord arc type in the attribute block 0.
4. Draw an arc using `UIS$CIRCLE` with the modified attribute block.
5. Repeat steps 3 and 4.
6. Erase the virtual display and delete the display window.
7. Create a display window and viewport with an identifying title.
8. Modify the arc type attribute. Select the pie arc type.
9. Select a fill pattern.
 - Modify the font attribute in attribute block 0.
 - Modify the fill pattern attribute block 0.
10. Draw an arc using the modified arc type, font, and fill pattern attribute blocks.

```
PROGRAM ARC
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,40.0,40.0,15.0,15.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','CHORD AND PIE')

CALL UIS$SET_ARC_TYPE(VD_ID,0,6,UIS$C_ARC_CHORD) ①
CALL UIS$CIRCLE(VD_ID,6,5.0,20.0,15.0,0.0,150.0)
```



```

CALL UIS$SET_ARC_TYPE(VD_ID,0,1,UIS$C_ARC_PIE) ②
CALL UIS$CIRCLE(VD_ID,1,23.0,20.0,15.0,0.0,150.0)

PAUSE

CALL UIS$DELETE_WINDOW(WD_ID) ③
CALL UIS$ERASE(VD_ID) ④

PAUSE

WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','FILLED PIE') ⑤
CALL UIS$SET_ARC_TYPE(VD_ID,0,1,UIS$C_ARC_PIE)
CALL UIS$SET_FONT(VD_ID,1,2,'UIS$FILL_PATTERNS')
CALL UIS$SET_FILL_PATTERN(VD_ID,2,3,PATT$C_HORIZ2_6) ⑥
CALL UIS$CIRCLE(VD_ID,3,18.0,20.0,15.0,0.0,150.0)

PAUSE

END

```

The program ARC creates two arcs and specifies two ways of closing those arcs ① ②.

In order to change the window caption, we delete the display window and its associated viewport ③. Because the second part of the program draws a new graphic object, we need to erase existing graphic objects ④.

A new display window is created and its viewport bears a new title. ⑤.

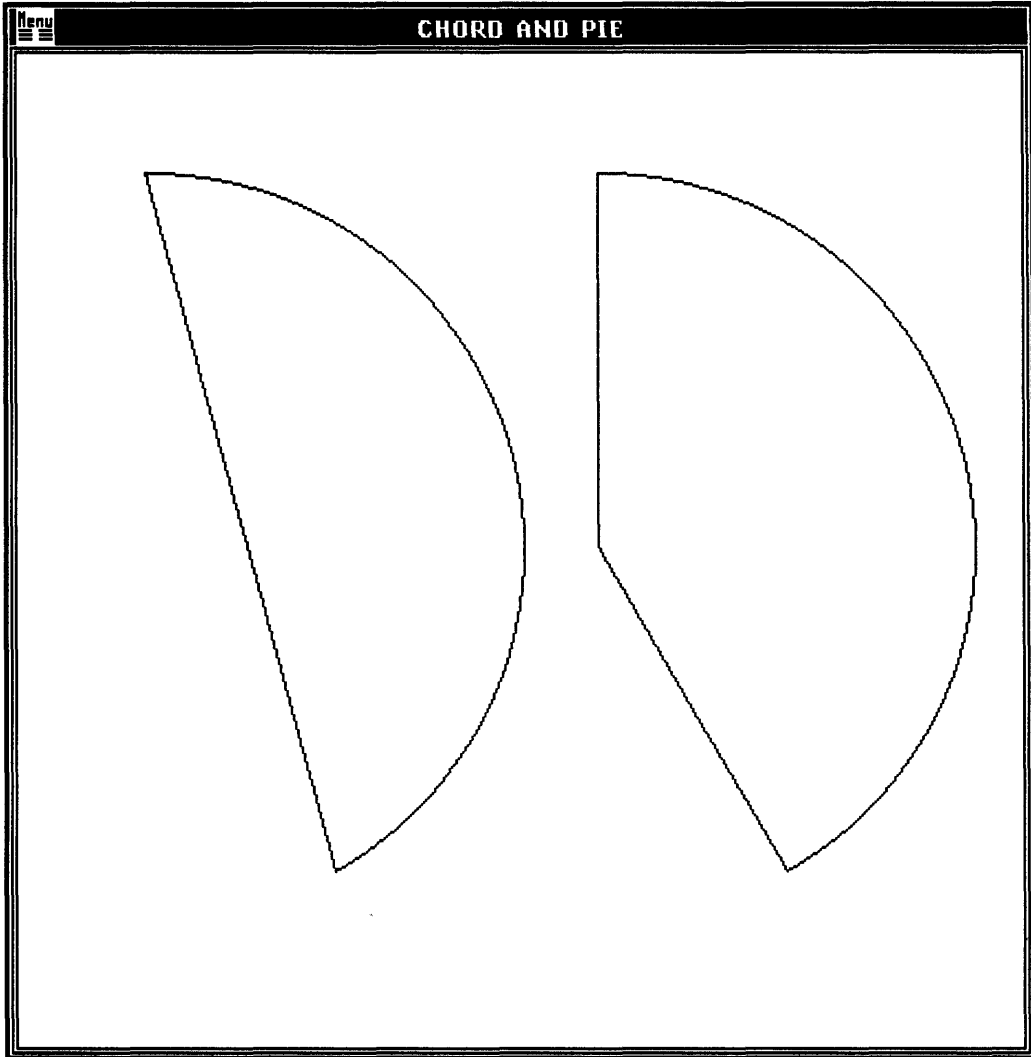
The new graphic object is another arc with a pie arc type and containing a fill pattern ⑥.

11.2.2.2 Calling UIS\$SET_ARC_TYPE and Using Fill Patterns

Figure 11-1 describes two ways of closing an arc.

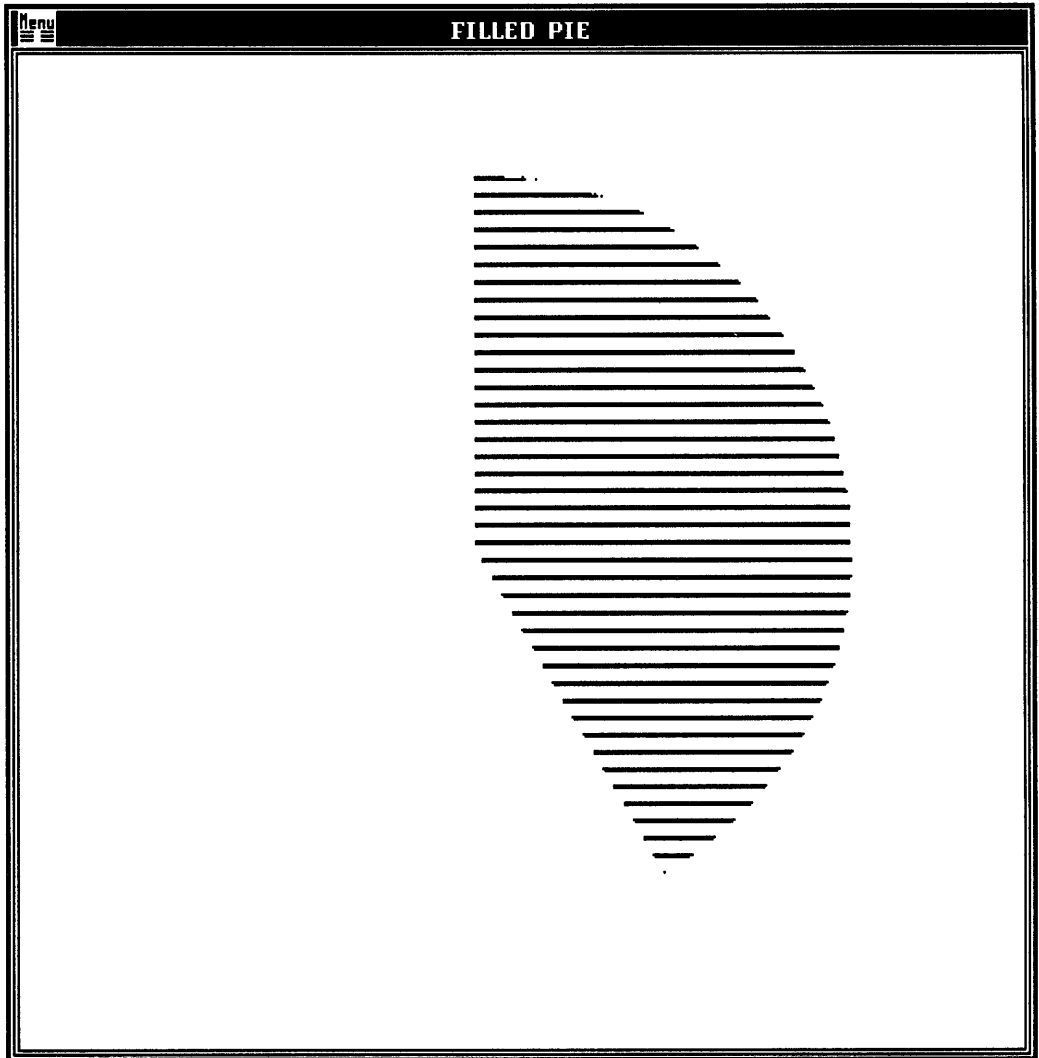
11-6 Graphics and Windowing Attributes

Figure 11-1 Closing an Arc



Finally, the second part of the program ARC executes and the fill pattern is drawn in the pie as shown in Figure 11-2.

Figure 11-2 Filling a Closed Arc



11-8 Graphics and Windowing Attributes

11.2.2.3 Program Development II

Programming Objective

To draw thickened lines.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with a title.
3. Draw two horizontal lines the width of the viewport—one near the bottom of the viewport and one near the top of the viewport.
4. Draw a vertical line connecting the horizontal lines.
5. Modify the line width attribute in attribute block 0 by a factor of 2.
6. Repeat steps 4 and 5.

```
PROGRAM LINE_WIDTH
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,60.0,30.0,15.0,15.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','LINE WIDTH')

CALL UIS$PLOT(VD_ID,0,1.0,25.0,60.0,25.0) ①
CALL UIS$PLOT(VD_ID,0,1.0,5.0,60.0,5.0) ②
CALL UIS$PLOT(VD_ID,0,5.0,5.0,5.0,25.0) ③
CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,2.0) ④
CALL UIS$PLOT(VD_ID,1,10.0,5.0,10.0,25.0) ⑤
CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,4.0)
CALL UIS$PLOT(VD_ID,1,15.0,5.0,15.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,6.0)
CALL UIS$PLOT(VD_ID,1,20.0,5.0,20.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,8.0)
CALL UIS$PLOT(VD_ID,1,25.0,5.0,25.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,10.0)
CALL UIS$PLOT(VD_ID,1,30.0,5.0,30.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,12.0)
CALL UIS$PLOT(VD_ID,1,35.0,5.0,35.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,14.0)
CALL UIS$PLOT(VD_ID,1,40.0,5.0,40.0,25.0)
```

```

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,16.0)
CALL UIS$PLOT(VD_ID,1,45.0,5.0,45.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,18.0)
CALL UIS$PLOT(VD_ID,1,50.0,5.0,50.0,25.0)

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,20.0)
CALL UIS$PLOT(VD_ID,1,55.0,5.0,55.0,25.0)

PAUSE
END

```

Two parallel lines are drawn with normal thickness the width of the display window using UIS\$PLOT ❶ ❷.

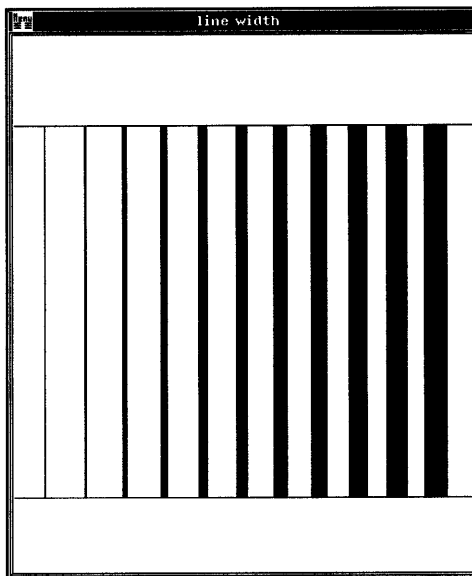
A vertical line of normal thickness is drawn ❸.

Subsequent calls modify the line width attribute ❹ and draw the resulting line ❺ from the line in the lower half of the display window to the line in the upper half of the display screen.

11.2.2.4 Calling UIS\$SET_LINE_WIDTH

Figure 11-3 shows lines are drawn from point to point with increasing thickness.

Figure 11-3 Line Width



2K 402685

NOTE: Extremely thick lines should be drawn using UIS\$PLOT or UIS\$PLOT_ARRAY to and UIS\$SET_FILL_PATTERN construct filled rectangles.

11-10 Graphics and Windowing Attributes

11.2.2.5 Program Development III

Programming Objective

To draw various patterns of thickened dots and dashes.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with title.
3. Modify the line width attribute to a thickness of 5 pixels.
4. Draw a solid thick line.
5. Modify the line style attribute.
6. Draw the dashed line.
7. Repeat steps 5 and 6.

```
PROGRAM LINE_STYLE
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,20.0,20.0,15.0,6.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','LINE STYLE AND WIDTH')

CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,5.0) ①
CALL UIS$PLOT(VD_ID,1,1.0,18.0,18.0,10.0)

CALL UIS$SET_LINE_STYLE(VD_ID,1,1,'FFFFFFF0'X) ②
CALL UIS$PLOT(VD_ID,1,1.0,14.0,18.0,10.0)

CALL UIS$SET_LINE_STYLE(VD_ID,1,2,'FOFOFOFO'X) ③
CALL UIS$PLOT(VD_ID,2,1.0,10.0,18.0,10.0)

CALL UIS$SET_LINE_STYLE(VD_ID,2,3,'90909090'X) ④
CALL UIS$PLOT(VD_ID,3,1.0,6.0,18.0,10.0)

CALL UIS$SET_LINE_STYLE(VD_ID,3,4,'10010010'X) ⑤
CALL UIS$PLOT(VD_ID,4,1.0,2.0,18.0,10.0)

PAUSE

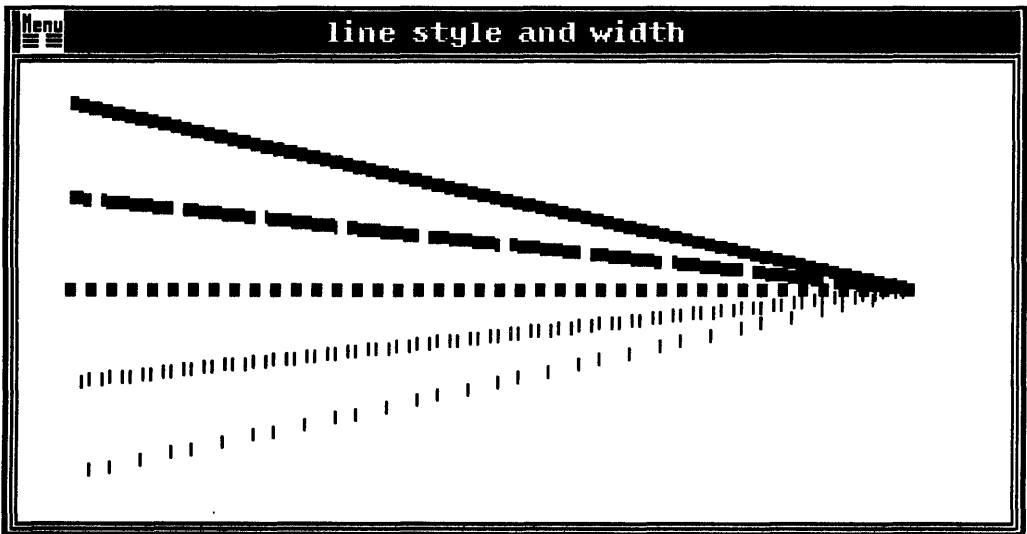
END
```

Different line styles are created by selecting different hexadecimal values in the calls to `UIS$SET_LINE_STYLE` ① ② ③ ④. The hexadecimal values set bits in the line style bit vector, which, in turn, generate a pattern.

11.2.2.6 Calling `UIS$SET_LINE_WIDTH` and `UIS$SET_LINE_STYLE`

When the program `LINE_STYLE` executes, five lines are drawn. Each line is drawn with the same width but different style. The pattern of dots and dashes is determined by the value supplied to the line style longword bit vector as shown in Figure 11-4.

Figure 11-4 Modifying Line Width and Style



ZK-4552-85

11.2.2.7 Program Development IV

Programming Objective

To construct a vertical bar graph.

Programming Tasks

1. Load arrays from `DATA` statements.
2. Create a virtual display.
3. Create a display window and viewport with a title.
4. Draw the x and y axes.
5. Draw the legend.
6. Draw the information along the x axis.
7. Draw the information along the y axis.
8. Modify the font and fill pattern attributes.

11-12 Graphics and Windowing Attributes

9. Draw the vertical bars using the appropriate fill patterns to their proper heights using the arrays.

```
PROGRAM GRAPH
IMPLICIT INTEGER(A-Z)
CHARACTER*4 STRING
REAL ARRAY1(8),ARRAY2(8),X,X2,HEIGHT,Y ①
DATA ARRAY1 /5.0,10.0,12.0,13.0,15.0,20.0,25.0,30.0/
DATA ARRAY2 /0.0, 1.0, 2.0, 1.0, 4.0, 9.0,15.0,21.0/

INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(-5.0,-5.0,50.0,50.0,20.0,20.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','GRAPH')

CALL UIS$SET_LINE_WIDTH(VD_ID,0,16,5.0) ②
CALL UIS$PLOT(VD_ID,16,0,0,0,35.0) ③
CALL UIS$PLOT(VD_ID,16,0,0,45.0,0) ④

CALL UIS$TEXT(VD_ID,0,'U.S. ADULT POPULATION VS. CAR OWNERSHIP',
2      10.0,-3.0) ⑤
```

- c Information along the y axis

```
DO 20 I = 1,7
Y = 5.0 * FLOAT (I) ⑥
N = 25 * I ⑦
ENCODE (3,10,STRING) N ⑧
10 FORMAT (I3)
20 CALL UIS$TEXT(VD_ID,0,STRING,-3.0,Y) ⑨

CALL UIS$TEXT(VD_ID,0,'(in millions)',-3.0,37.0)
```

- c Information along the x axis

```
DO 40 I = 1,8
Y = 5.0 * FLOAT (I)
N = 1900 + (10 * I)
ENCODE (4,30,string) N
30 FORMAT (I4)
40 CALL UIS$TEXT(VD_ID,0,string,Y,-1.0)

CALL UIS$SET_FONT(VD_ID,0,1,'UIS$FILL_PATTERNS') ⑩
CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_HORIZ4_4) ⑪
CALL UIS$SET_FILL_PATTERN(VD_ID,1,2,PATT$C_GREY12_16) ⑫
```

- C PLOT POPULATION RECTANGLE

```
DO 100 I = 1,8
```



```

        X = 5.0 * FLOAT(I)
        X2 = X + 2.0
        HEIGHT = ARRAY1(I) ⑬
        CALL UIS$PLOT (VD_ID,1, X,0.0, X,HEIGHT, X2,HEIGHT, X2,0.0)
C PLOT CAR RECTANGLE
        X = X + 1.0
        X2 = X + 2.0
        HEIGHT = ARRAY2(I) ⑭
        CALL UIS$PLOT (VD_ID,2, X,0.0, X,HEIGHT, X2,HEIGHT, X2,0.0)
100 CONTINUE
        PAUSE
        END

```

Two arrays, ARRAY1 and ARRAY2 are declared ① to store the height of each vertical bar in the graph.

The x and y axes are drawn ③ ④. However, a previous call to UIS\$SET_LINE_WIDTH ② has modified the attribute block controlling the appearance of lines. We have indicated that we want the width of the lines (x and y axes) to be five times wider than normal.

The legend of the graph is created in a call to UIS\$TEXT ⑤.

The y world coordinate values are computed ⑥ as multiples of 5 where I represents the number of passes through the DO loop. The adult population numbers will be written at these intervals.

The numbers along the y axis are computed and stored in the variable N ⑦ and then returned to the variable *string* as character string constants ⑧ ⑨.

Before you create the rectangles to represent the eight vertical bars in the graph, you must specify the fill pattern—either an existing one or a new pattern. Because the font attribute has not been modified in our program, UIS\$SET_FONT uses a copy of attribute block 0 to set the font attribute ⑩. In this case, specify a font ID UIS\$FILL_PATTERNS to indicate that you want the file of fill patterns.

Now we must set the fill pattern attribute using UIS\$SET_FILL_PATTERN. The program must use two different fill patterns to contrast adult population vertical bars from automobile vertical bars ⑪ ⑫.

The values previously assigned to each element of ARRAY1 and ARRAY2 control the height of the vertical bars ⑬ ⑭.

11-14 Graphics and Windowing Attributes

11.2.2.8 Calling `UIS$SET_FONT` and `UIS$SET_FILL_PATTERN`

If you ran the program `GRAPH` now it would produce the vertical bar graph as shown in Figure 11-5.

Whenever you create a fill pattern, you must include `UIS$SET_FONT` and `UIS$SET_FILL_PATTERN`. The positional order of the calls is important. Calls to `UIS` routines that modify an attribute block must precede the call that creates the graphic object.

In order to produce the desired change in the resulting graphic object, the accompanying call to `UIS$PLOT` must reference the same output attribute block number.

11.2.3 Using the Windowing Attribute

The clipping rectangle attribute modifies the size of the viewable portion of the virtual display. It does not resize the display window or display viewport.

11.2.3.1 Programming Options

There is only one attribute, namely the clipping attribute, that controls what is visible through the display window and viewport.

Clipping Rectangle

Maybe you need to restrict drawing in the virtual display to a specified rectangle. You can create clipping rectangles that view a portion of your original display window using `UIS$SET_CLIP`. These rectangles are not display windows, but they can be used to partition your virtual display into discrete areas. They create an environment within your virtual display that can be visited whenever you reference the appropriate attribute block that contains a modified clipping rectangle attribute. Note that the clipping rectangle merely restricts drawing to an area. It does not change mapping between the virtual display and the display window.

11.2.3.2 Program Development

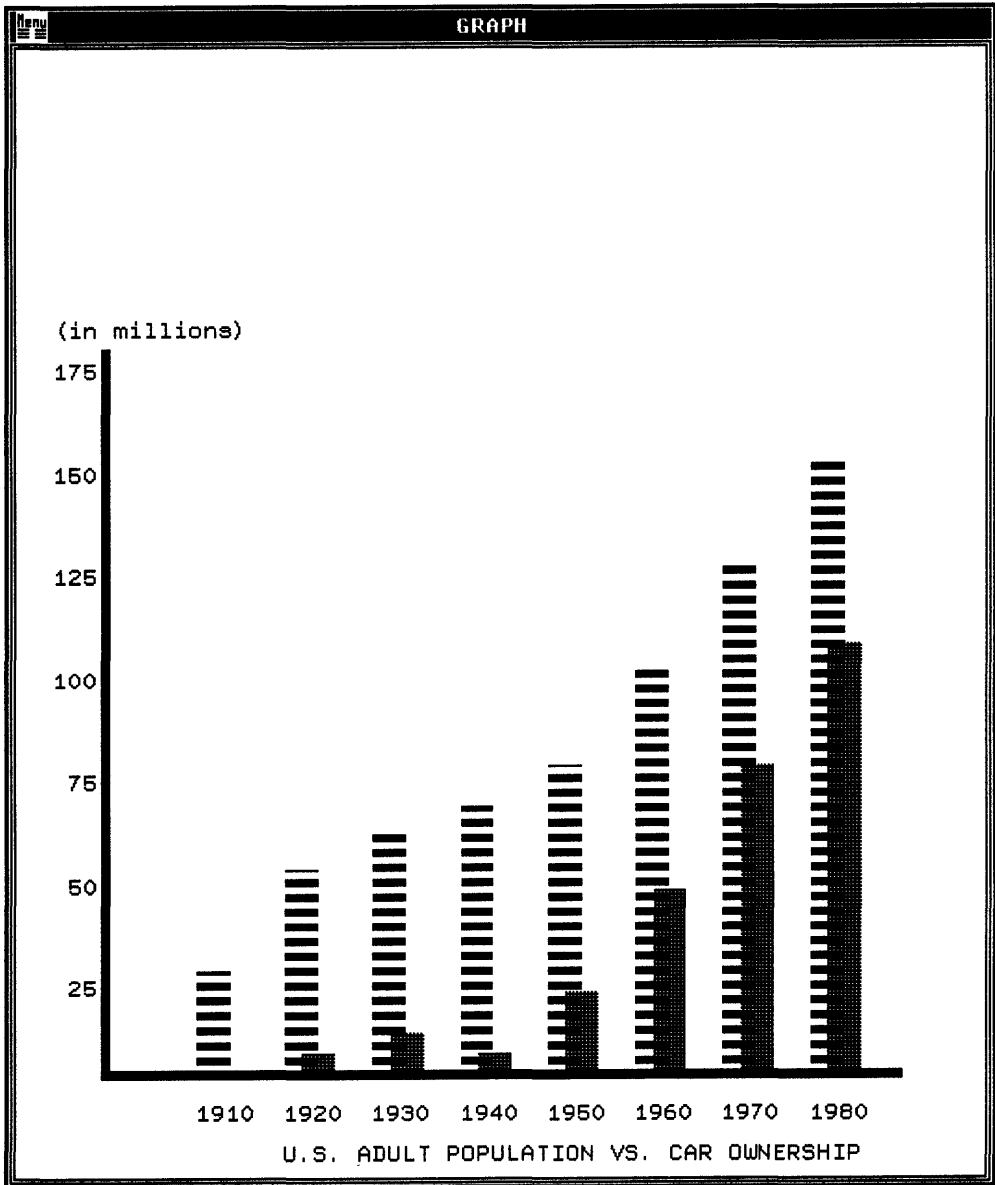
Programming Objective

To construct three clipping rectangles.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with a title.
3. Choose a font and modify the font attribute.
4. Specify a clipping rectangle and modify the clipping attribute.
5. Draw a line of text using the modified font attribute with clipping disabled.
6. Draw a line of text using the modified font attribute with clipping enabled.

Figure 11-5 Vertical Bar Graph



11-16 Graphics and Windowing Attributes

7. Repeat steps 3 through 6 two more times.

Logical names have been defined for font file names.

```
PROGRAM CLIP
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,45.0,45.0,15.0,5.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','CLIPPING')

CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_5') ①
CALL UIS$SET_CLIP(VD_ID,1,5,1.0,1.0,10.0,40.0) ②
CALL UIS$TEXT(VD_ID,1,'Still waters run deep',0.0,40.0)
CALL UIS$NEW_TEXT_LINE(VD_ID,1)
CALL UIS$TEXT(VD_ID,5,'Still waters run deep')

CALL UIS$SET_FONT(VD_ID,0,2,'MY_FONT_6') ③
CALL UIS$NEW_TEXT_LINE(VD_ID,2)
CALL UIS$SET_CLIP(VD_ID,2,6,15.0,15.0,35.0,40.0) ④
CALL UIS$TEXT(VD_ID,2,'The sleepy fox has seldom feathered breakfasts')
CALL UIS$NEW_TEXT_LINE(VD_ID,2)
CALL UIS$TEXT(VD_ID,6,'The sleepy fox has seldom feathered breakfasts')

CALL UIS$SET_FONT(VD_ID,0,3,'MY_FONT_10') ⑤
CALL UIS$NEW_TEXT_LINE(VD_ID,3)
CALL UIS$SET_CLIP(VD_ID,3,7,7.0,5.0,30.0,40.0) ⑥
CALL UIS$TEXT(VD_ID,3,'When the wind is west, the fish bite best')
CALL UIS$NEW_TEXT_LINE(VD_ID,3)
CALL UIS$TEXT(VD_ID,7,'When the wind is west, the fish bite best')

PAUSE

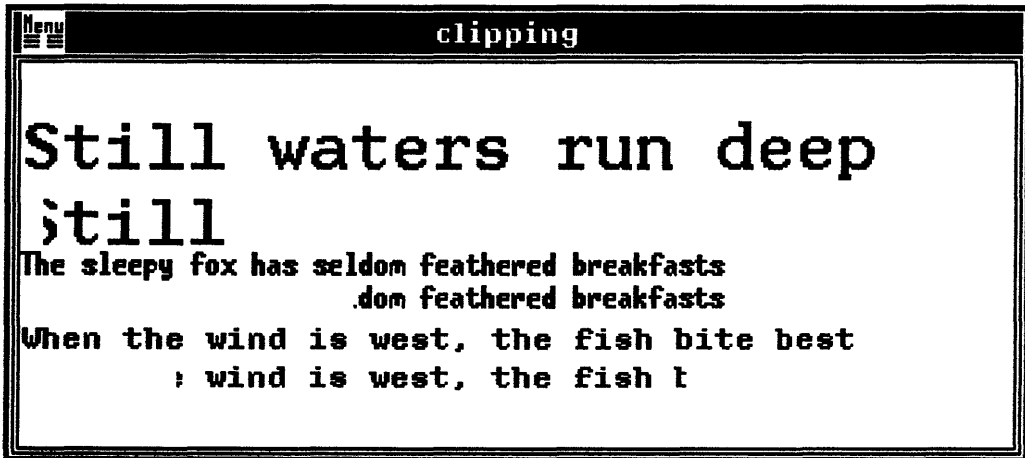
END
```

Three fonts ① ③ ⑤ are used to illustrate clipping rectangles. The call to UIS\$SET_CLIP modifies the attribute block that controls clipping rectangle size. Each call to UIS\$SET_CLIP ② ④ ⑥ specifies a different clipping rectangle size. Although, only one display viewport has been specified in this program, UIS\$SET_CLIP creates many compartments within the display window.

11.2.3.3 Calling UIS\$SET_CLIP

Your workstation screen would display the graphic objects shown in Figure 11-6.

Figure 11-6 Clipping rectangles



ZK-4554-85

As you can see, UIS\$SET_CLIP has altered the display window of the last three lines. Only portions of each lines are now visible.

Chapter 12

Inquiry Routines

12.1 Overview

Inquiry routines return program-specific information to the application; in this way, they behave like functions. However, unlike functions which return a single value through a return variable, certain UIS inquiry routines return data in two or more parameters in the argument list. This data can range from current attribute settings to current state of the pointer buttons. Your application program can use this data to establish context during program execution, to check for true or false conditions, or to verify that requested operation has been performed.

12.2 Inquiry Routines—How to Use Them

Many common graphics application program rely on program-specific data such as the position of pointer devices or the font size and so forth. Inquiry routines return such data to the program. The data can be used as input to the application as you see fit. Such routines are more properly termed *functions* when used with high-level programming languages.

12.2.1 Using Inquiry Routines

Generally, UIS routines in the form UIS\$GET_XXXX return information to the application program. Some of these routines behave like functions and return a single value to the program, while others return more than one value in the argument list. In any case, these routines obtain data about text and font size, windows, keyboard attributes, pointer position, and attribute settings. Such data can be used as input to subsequent routines.

12-2 Inquiry Routines

12.2.1.1 Programming Options

Your application program can request the following types of application-specific information:

- Color information
- Display list information
- Graphics and text attributes
- Keyboard and pointer characteristics
- Windowing information

The inquiry routines are grouped functionally in Table 12-1.

Table 12-1 Inquiry Routines

Inquiry	Information Returned
Color¹	
UIS\$GET_BACKGROUND_INDEX	Background color index
UIS\$GET_COLOR	Single RGB color value in a color map entry
UIS\$GET_COLORS	RGB color values
UIS\$GET_HW_COLOR_INFO	Hardware color map characteristics
UIS\$GET_INTENSITIES	Intensity values in virtual color map
UIS\$GET_INTENSITY	Single intensity value in a virtual color map entry
UIS\$GET_VCM_ID	Virtual color map identifier
UIS\$GET_WRITING_INDEX	Writing color index
UIS\$GET_WRITING_MODE	Writing mode
UIS\$GET_WS_COLOR	Workstation standard color
UIS\$GET_WS_INTENSITY	Workstation standard color intensity
Color Conversion²	
UIS\$HLS_TO_RGB	Converts HLS values to RGB color values
UIS\$HSV_TO_RGB	Converts HSV values to RGB color values
UIS\$RGB_TO_HLS	Converts RGB values to HLS color values
UIS\$RGB_TO_HSV	Converts RGB values to HSV color values

¹See Chapter 16 for more information about color and intensity inquiry routines.

²See Chapter 16 for more information about color conversion routines.

Table 12-1 (Cont.) Inquiry Routines

Inquiry	Information Returned
Display List	
UIS\$FIND_PRIMITIVE	Identifier of the next primitive in the specified rectangle
UIS\$FIND_SEGMENT	Segment identifier of the next segment that contains objects in a specified rectangle
UIS\$GET_CURRENT_OBJECT	Identifier of last object drawn in virtual display
UIS\$GET_NEXT_OBJECT	Identifier of next object
UIS\$GET_OBJECT_ATTRIBUTES	Object type
UIS\$GET_PARENT_SEGMENT	Parent segment identifier
UIS\$GET_PREVIOUS_OBJECT	Identifier of the previous object
UIS\$GET_ROOT_SEGMENT	Root segment identifier
Graphics	
UIS\$GET_ARC_TYPE	Arc type used to close arc
UIS\$GET_FILL_PATTERN	Fill pattern index and status
UIS\$GET_LINE_STYLE	Line style vector
UIS\$GET_LINE_WIDTH	Line width in pixels or as a world coordinate x-coordinate width
Keyboard and Pointer	
UIS\$GET_ABS_POINTER_POS	Absolute position of the pointer
UIS\$GET_BUTTONS	State of the pointer device buttons
UIS\$GET_KB_ATTRIBUTES	Keyboard characteristics
UIS\$GET_POINTER_POSITION	Position of pointer in world coordinates
UIS\$GET_TB_INFO	Returns the characteristics of the tablet
UIS\$GET_TB_POSITION	Position on tablet in centimeters
UIS\$TEST_KB	Successful or unsuccessful connection between virtual and physical keyboard
Text	
UIS\$GET_ALIGNED_POSITION	World coordinates along the x-height of the current position of the next character
UIS\$GET_CHAR_ROT	Angle of character rotation in degrees
UIS\$GET_CHAR_SIZE	If character scaling is enabled and the scaling factors used

12-4 Inquiry Routines

Table 12-1 (Cont.) Inquiry Routines

Inquiry	Information Returned
Text	
UIS\$GET_CHAR_SLANT	Angle of character slant in degrees
UIS\$GET_CHAR_SPACING	Character and line spacing factor
UIS\$GET_FONT	Font name
UIS\$GET_FONT_ATTRIBUTES	All font character characteristics
UIS\$GET_FONT_SIZE	Font size in centimeters
UIS\$GET_LEFT_MARGIN	World coordinate of left margin
UIS\$GET_POSITION	World coordinates of text baseline
UIS\$GET_TEXT_FORMATTING	Formatting mode
UIS\$GET_TEXT_MARGINS	Text margin settings for a line of text
UIS\$GET_TEXT_PATH	Direction of text drawing
UIS\$GET_TEXT_SLOPE	Angle of the text baseline in degrees
UIS\$MEASURE_TEXT	Proportions of text in world coordinates
Windowing	
UIS\$GET_CLIP	Clipping rectangle
UIS\$GET_DISPLAY_SIZE	Display screen dimensions in centimeters
UIS\$GET_VIEWPORT_ICON	Whether or not the icon is occluded
UIS\$GET_VIEWPORT_POSITION	Absolute position of display viewport on display screen
UIS\$GET_VIEWPORT_SIZE	Dimensions of the display viewport in centimeters
UIS\$GET_VISIBILITY	Whether or not viewport is occluded
UIS\$GET_WINDOW_ATTRIBUTES	Window and viewport attributes
UIS\$GET_WINDOW_SIZE	Dimensions of the display window in world coordinates

12.2.1.2 Program Development I

Programming Objective

To return font and viewport information in order to center text.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with a title.
3. Obtain the font size for a particular character string, viewport size, and display screen size.
4. Choose a font and modify the font attribute block.
5. Draw a line of centered text in the viewport using the modified font attribute and the information from the inquiry routines.
6. Print the inquiry information in the terminal emulation window.
7. Repeat steps 3 through 6.

The font file names used in this program are logical names.

```

PROGRAM CENTER
IMPLICIT INTEGER(a-z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL F_WIDTH,F_HEIGHT,D_WIDTH,D_HEIGHT
REAL V_WIDTH,V_HEIGHT

VD_ID1=UIS$CREATE_DISPLAY(1.0,1.0,15.0,2.0,15.0,2.0)
WD_ID1=UIS$CREATE_WINDOW(VD_ID1,'SYS$WORKSTATION','CENTERED TEXT')

CALL UIS$GET_FONT_SIZE('MY_FONT_7','Time has wings',
2      F_WIDTH,F_HEIGHT) ①
CALL UIS$GET_DISPLAY_SIZE('SYS$WORKSTATION',D_WIDTH,D_HEIGHT) ②
CALL UIS$GET_VIEWPORT_SIZE(WD_ID1,V_WIDTH,V_HEIGHT) ③

CALL UIS$SET_FONT(VD_ID1,0,7,'MY_FONT_7') ④
CALL UIS$TEXT(VD_ID1,7,'Time has wings',
2      (V_WIDTH-F_WIDTH)/2,
2      V_HEIGHT) ⑤

PAUSE

PRINT 50
50  FORMAT(T10,'FIRST LINE',T39,'WIDTH',T51,'HEIGHT')

PRINT 75
75  FORMAT(T2,'-----',
2      '-----')
```

12-6 Inquiry Routines

```
PRINT 100, F_WIDTH, F_HEIGHT
100  FORMAT(T2,'The dimensions of the font are:',
        2      T39,f5.2,T46,'cm.',T51,f5.2,T58,'cm.')

PRINT 150,D_WIDTH,D_HEIGHT
150  FORMAT(T2,'The dimensions of the display are:',
        2      T39,f6.2,T46,'cm.',T51,f6.2,T58,'cm.')

PRINT 200,V_WIDTH,V_HEIGHT
200  FORMAT(T2,'The dimensions of the viewport are:',
        2      T39,f6.2,T46,'cm.',T51,f6.2,T58,'cm.')

CALL UIS$SET_FONT(VD_ID1,7,8,'MY_FONT_5') ④
CALL UIS$MEASURE_TEXT(VD_ID1,8,'four seasons',
        2      F_WIDTH,F_HEIGHT)
CALL UIS$NEW_TEXT_LINE(VD_ID1,8)
CALL UIS$TEXT(VD_ID1,8,'four seasons',
        2      (V_WIDTH-F_WIDTH)/2,(V_HEIGHT-F_HEIGHT)) ⑦

TYPE *,' '

PRINT 550
550  FORMAT(T10,'SECOND LINE',T39,'WIDTH',T51,'HEIGHT')

PRINT 575
575  FORMAT(T2,'-----',
        2      '-----')

PRINT 610, F_WIDTH, F_HEIGHT
610  FORMAT(T2,'THE DIMENSIONS OF THE FONT ARE:',
        2      T39,f5.2,T46,'cm.',T51,f5.2,T58,'cm.')

PRINT 700,D_WIDTH,D_HEIGHT
700  FORMAT(T2,'The dimensions of the display are:',
        2      T39,f6.2,T46,'cm.',T51,f6.2,T58,'cm.')

PRINT 800,V_WIDTH,V_HEIGHT
800  FORMAT(T2,'The dimensions of the viewport are:',
        2      T39,f6.2,T46,'cm.',T51,f6.2,T58,'cm.')

PAUSE
END
```

The three inquiry functions `UIS$GET_FONT_SIZE`, `UIS$GET_DISPLAY_SIZE`, and `UIS$GET_VIEWPORT_SIZE` are called ① ② ③. Each function returns data to uniquely specified variables within its argument list.

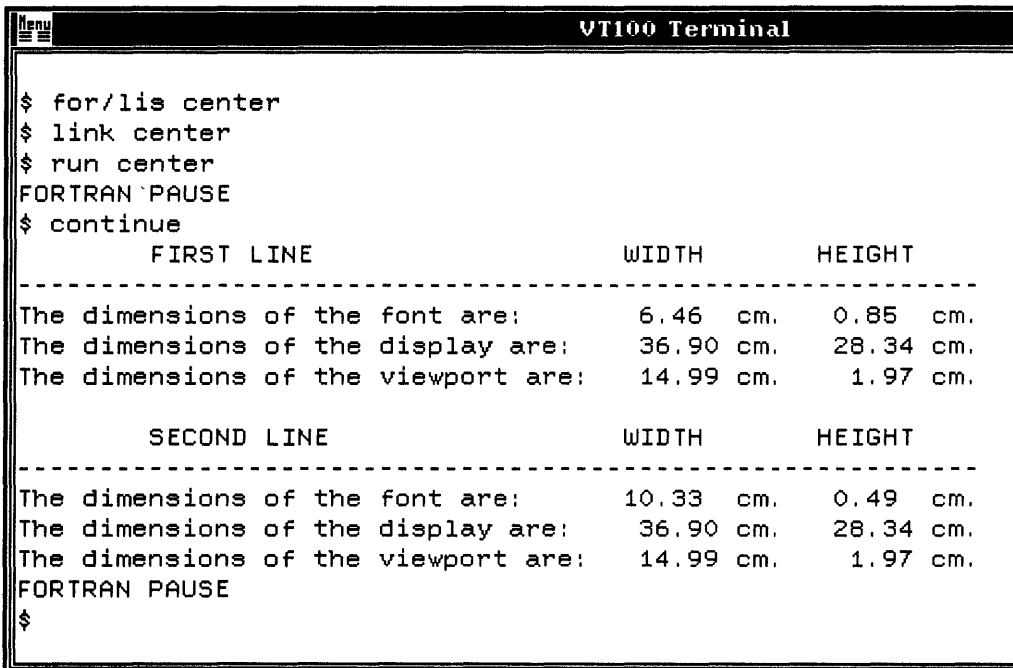
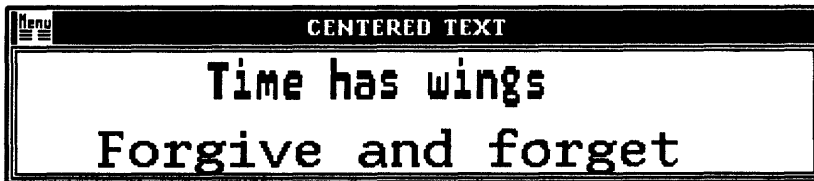
A logical name is defined ④ ⑤ to represent the 31-character font file name. The first call to `UIS$TEXT` ⑥ places a text string in the window. The starting position for creating text is calculated from the expression in the argument list. VAX FORTRAN allow arithmetic expressions as arguments. ⑦ If your application is written in a programming language other than VAX FORTRAN, please refer to the appropriate language reference manual.

In order to center the text in this window, we subtracted the length of the text from the total width of the viewport and divided the result by 2. The distance of the text from the lower border of the window (the *y* coordinate) is equal to the value of the variable *v_height*, the height of the display viewport.

12.2.1.3 Invoking UIS\$GET_FONT_SIZE, UIS\$GET_DISPLAY_SIZE, and UIS\$GET_VIEWPORT_SIZE

If you ran this program now, your workstation screen would display graphic objects as shown in Figure 12-1.

Figure 12-1 Centering Text



12-8 Inquiry Routines

Note that output from the FORTRAN PRINT or TYPE statement is not displayed in the window we have created. The TYPE and PRINT statements are equivalent to the logical names FOR\$TYPE and FOR\$PRINT which translate to the logical name SYS\$OUTPUT. Only UIS\$TEXT can write text to a virtual display.

12.2.1.4 Program Development II

Programming Objective

To construct a pie graph illustrating the operating budget of a small New England town.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport with a title.
3. Choose a font and modify the font attribute.
4. Print the title of the graph using the modified font attribute.
5. Obtain font information.
6. Modify the arc type attribute.
7. Choose a fill pattern and modify the font attribute and the fill pattern attribute.
8. Draw an arc using the modified fill pattern attribute.
9. Draw part of the legend to appear below the pie graph.
10. Obtain and print arc type and fill pattern information.
11. Repeat steps 6 through 9.

```
PROGRAM PIE_GRAPH
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
CHARACTER*32 BUFFERDESC
LOGICAL*4 FILL_ENABLED

VD_ID=UIS$CREATE_DISPLAY(-3.0, -3.0, 25.0, 25.0, 15.0, 15.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'PIE GRAPH')

CALL UIS$SET_FONT(VD_ID, 0, 9, 'MY_FONT_10')
CALL UIS$TEXT(VD_ID, 9, 'OPERATING BUDGET', 6.0, 24.0)
CALL UIS$TEXT(VD_ID, 9, 'TOWN OF GREENWICH, MASS.', 4.0, 22.0)
CALL UIS$GET_FONT(VD_ID, 9, BUFFERDESC, LENGTH) ①

PRINT 10, BUFFERDESC
FORMAT(T2, 'THE FONT NAME IS', T20, A31) ②
```

```

11 PRINT 11,LENGTH
   FORMAT(T2,'THE LENGTH OF THE FONT NAME IS ',T33,I3,T37,'CHARACTERS')

   CALL UIS$SET_ARC_TYPE(VD_ID,0,1,UIS$C_ARC_PIE) ③
   CALL UIS$SET_FONT(VD_ID,1,1,'UIS$FILL_PATTERNS')
   CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_BRICK_DOWNDIAG)
   CALL UIS$CIRCLE(VD_ID,1,10.0,10.0,8.0,0.0,50.0)
   call uis$plot(vd_id,1,0.0,0.0,2.0,0.0,2.0,-1.0,
2      0.0,-1.0,0.0,0.0)
   call uis$text(vd_id,0,'Fire',3.0,0.0)
   ARC_TYPE=UIS$GET_ARC_TYPE(VD_ID,1) ④
   FILL_ENABLED=UIS$GET_FILL_PATTERN(VD_ID,1,INDEX) ⑤

15 PRINT 15,ARC_TYPE
   FORMAT(T2,'THE ARC TYPE IS',T25,I1) ⑥

20 PRINT 20,FILL_ENABLED
   FORMAT(T2,'IS THE FILL PATTERN ENABLED?',T32,L1)

   CALL UIS$SET_FONT(VD_ID,1,2,'UIS$FILL_PATTERNS')
   CALL UIS$SET_FILL_PATTERN(VD_ID,2,2,PATT$C_DOWNDIAG4_4)
   CALL UIS$CIRCLE(VD_ID,2,10.0,10.0,8.0,50.0,95.0)
   CALL UIS$PLOT(VD_ID,2,10.0,0.0,12.0,0.0,12.0,-1.0,
2      10.0,-1.0,10.0,0.0)
   CALL UIS$TEXT(VD_ID,0,'Sanitation',14.0,0.0)

   CALL UIS$SET_FONT(VD_ID,2,3,'UIS$FILL_PATTERNS')
   CALL UIS$SET_FILL_PATTERN(VD_ID,3,3,PATT$C_HORIZ2_6)
   CALL UIS$CIRCLE(VD_ID,3,10.0,10.0,8.0,95.0,165.0)
   CALL UIS$PLOT(VD_ID,3,0.0,-2.0,2.0,-2.0,2.0,-3.0,
2      0.0,-3.0,0.0,-2.0)
   CALL UIS$TEXT(VD_ID,0,'Police',3.0,-2.0)

   CALL UIS$SET_FONT(VD_ID,3,4,'UIS$FILL_PATTERNS')
   CALL UIS$SET_FILL_PATTERN(VD_ID,4,4,PATT$C_GREY4_16D)
   CALL UIS$CIRCLE(VD_ID,4,10.0,10.0,8.0,165.0,360.0)
   CALL UIS$PLOT(VD_ID,4,10.0,-2.0,12.0,-2.0,12.0,-3.0,
2      10.0,-3.0,10.0,-2.0)
   CALL UIS$TEXT(VD_ID,0,'Schools',14.0,-2.0)

PAUSE
END

```

The program PIE_GRAPH returns information about the heading of the graph. A call to UIS\$GET_FONT ① identifies the font and its length ②. The font MY_FONT_10 is a logical name for a 31-character font file name.

Attribute block 1 contains the modified arc type attribute ③. When a new section of the arc is drawn, it will have a pie arc type which enables fill pattern.

12-10 Inquiry Routines

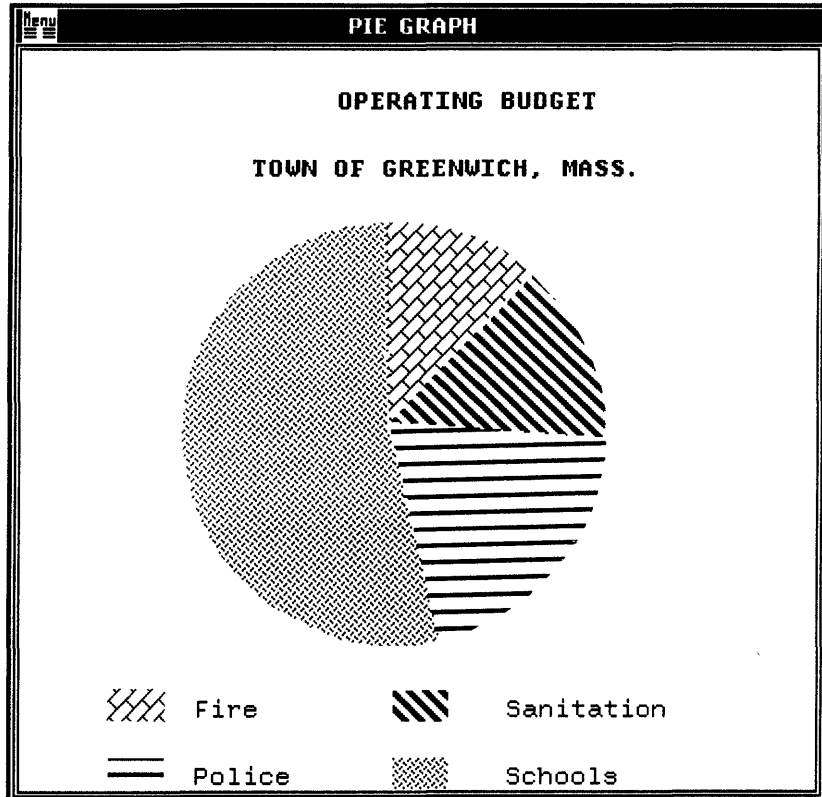
Arc type information is returned in the variable *arc_type* ④.

A call to `UIS$GET_FILL_PATTERN` ⑤ tests whether fill patterns are enabled. Fill pattern information is returned in the variable *fill_enabled* ⑥ as a Boolean value.

12.2.1.5 Invoking `UIS$GET_ARC_TYPE`, `UIS$GET_FILL_PATTERN`, and `UIS$GET_FONT`

The program `PIE_GRAPH` draws a pie graph containing four fill patterns and requests and displays certain program-specific information as shown in Figure 12-2.

Figure 12-2 Pie Graph



```

Menu
VT100 Terminal
$ for/lis pie_graph
$ link pie_graph
$ run pie_graph
THE FONT NAME IS MY_FONT_10
THE LENGTH OF THE FONT NAME IS 10 CHARACTERS
THE ARC TYPE IS 1
IS THE FILL PATTERN ENABLED? T
FORTRAN PAUSE
$ █
    
```


Chapter 13

Display Lists and Segmentation

13.1 Overview

As you have seen so far, you can use your applications to construct different types of graphic objects. Programs containing code for complex graphic objects can pose a problem because of their sheer size. In any case, the increasing complexity of your displays will require that you understand display list concepts. This chapter discusses the following topics:

- Creating and searching segments
- Editing and walking the display list
- Disabling display lists
- Creating UIS metafiles
- Attaching private data to graphic objects

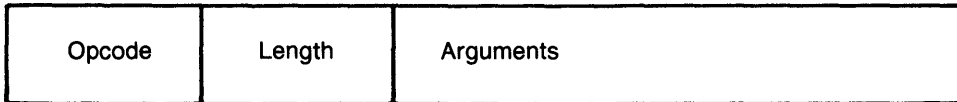
You can view creating complex objects as an opportunity to reduce the complexity of your graphic object and to modularize your coding through the use of *segmentation*.

13.2 Display Lists

UIS constructs a *display list* of encoded commands for graphics. These display lists remain resident in memory for use by UIS routines. Figure 13–1 shows the format of an entry in the display list.

13-2 Display Lists and Segmentation

Figure 13-1 Binary Encoded Instruction



ZK-5436-86

UIS signals an error if it encounters an invalid opcode.

Whenever you call UIS routines to create graphic objects or modify attribute blocks, you have added an entry to a display list. Only one display list exists for each virtual display.

A display list is a device-independent encoding of the exact contents of the virtual display. UIS maintains display lists for the following purposes:

- Automatic management of panning, zooming, resizing, and duplication of display windows
- High resolution printing of physical and virtual displays
- Structuring and manipulation of graphic objects in the virtual display
- Storing the contents of the virtual display in a buffer for later reexecution

13.3 Segments

A *segment* consists of calls to UIS graphics and text routines and any nested segments. Segments are created explicitly with a call to `UIS$BEGIN_SEGMENT`, and are terminated with a call to `UIS$END_SEGMENT`. A complex display list is a hierarchy of nested segments.

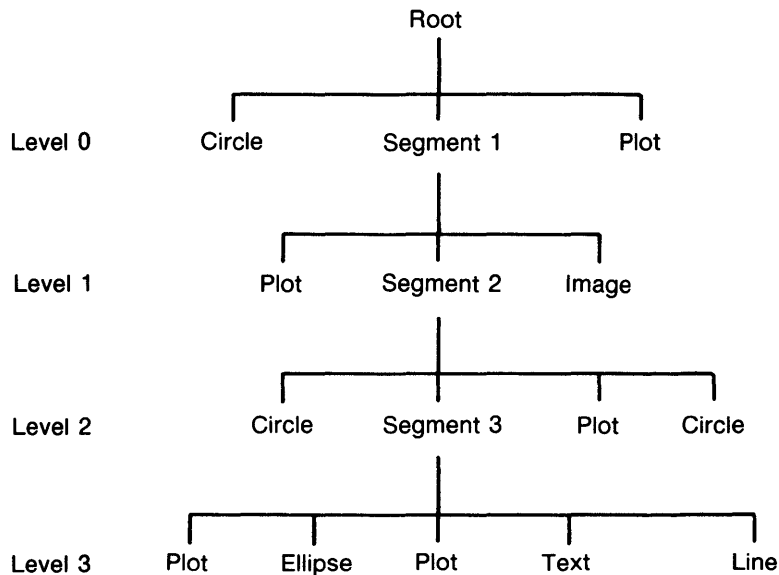
Any segment or output (graphic and text) routine not contained in an explicitly created segment is part of a top-level *root* segment.

The primary purpose of segmentation of graphics routines is to facilitate transformations—scaling, rotation, and translation. Segmentation also modularizes attributes. Complex graphic objects can be constructed in parts where each logical grouping of display list entries is contained within a segment. Such segments could be transformed or displayed individually and independently of the rest of the object. Changing attributes within a segment will not affect the attribute settings of a higher-level segment.

For example, a house, a barn, and landscape are constructed as three logical groupings, or *subpictures* of a complex display. Each subpicture is a segment of appropriate UIS routines. You could manipulate all three subpictures independently of each other.

Figure 13-2 shows a tree diagram of a display list containing nested segments. The diagram should be read from left to right and downward whenever a segment is encountered until there are no more segments. Read each level to the right and move upward to the next level where you left off.

Figure 13-2 Nested Segments



ZK-5459-86

13.3.1 Identifiers and Object Types

There are many types of UIS identifiers—for example, virtual display identifier, virtual keyboard identifier, transformation identifier, and so on. Identifiers allow your application to reference and manipulate internal objects. Managing the display list involves (1) traversing the display list downward object by object, (2) searching a segment, and (3) traversing upward through the segment path.

13-4 Display Lists and Segmentation

Segments

Each segment has a unique identifier returned by `UIS$BEGIN_SEGMENT`. If no segments were explicitly declared using `UIS$BEGIN_SEGMENT`, then the root segment has a unique identifier that can be used to manipulate the display list.

Objects

Every object in the virtual display has an object identifier. However, not all routines return identifier explicitly. Object and segment identifiers are useful in walking and editing the display list. They are used as reference points within complex display lists.

The identifier may not always be part of the calling sequence and sometimes must be returned using another UIS routine. For example, none of the graphics and text routines return identifiers explicitly. You can obtain the identifier using one of the routines listed in the table. The following table lists identifiers and the UIS routines that return them.

Graphic Object	Identifier	Routine
Segment	<code>seg_id</code>	<code>UIS\$BEGIN_SEGMENT</code> ¹
Root segment	<code>root_id</code>	<code>UIS\$GET_ROOT_SEGMENT</code>
Parent segment	<code>parent_id</code>	<code>UIS\$GET_PARENT_SEGMENT</code>
Graphic objects	<code>prev_id</code>	<code>UIS\$GET_PREVIOUS_OBJECT</code>
	<code>current_id</code>	<code>UIS\$GET_CURRENT_OBJECT</code>
	<code>next_id</code>	<code>UIS\$GET_NEXT_OBJECT</code>

¹`UIS$BEGIN_SEGMENT` returns the segment identifier in a return variable, `seg_id`.

Object Types

Even though you can manipulate the display list using segment and object identifiers, you still need to further identify those objects within a segment. You need to know exactly what type of object the display list entry is. The *object type* refers to the way UIS categorizes graphic objects and segments. There are six object types represented by the symbols listed here.

Symbol	Graphic Object
UIS\$C_OBJECT_SEGMENT	New segment
UIS\$C_OBJECT_PLOT	Point, line, or polygon
UIS\$C_OBJECT_TEXT	Text
UIS\$C_OBJECT_ELLIPSE	Ellipse or circle
UIS\$C_OBJECT_IMAGE	Raster image
UIS\$C_OBJECT_LINE	Unconnected lines

UIS\$GET_OBJECT_ATTRIBUTES returns object type information.

13.3.2 Programming Options

The behavior of display lists and segmentation can best be described separately. From the options available below, we will construct two programs. The first program illustrates disabling display lists, while the second demonstrates walking the display list.

Creating Segments

You can create an unlimited number of segments explicitly with UIS\$BEGIN_SEGMENT and UIS\$END_SEGMENT. UIS returns a unique identifier for each newly created segment that can be used by appropriate UIS routines to locate and edit segments. In addition, you can nest segments within segments.

NOTE: If UIS\$BEGIN_SEGMENT is called and no graphics and text routines are called before UIS\$END_SEGMENT is called, the segment is deleted and the identifier returned is no longer valid. If you wish to create an empty segment, call UIS\$BEGIN_SEGMENT followed by UIS\$PRIVATE. This sequence places private data in the segment and UIS\$END_SEGMENT will not consider the segment empty.

Enabling and Disabling Display Lists

Disabling a display list prevents new additions from being added to the list. Display lists are enabled and disabled explicitly with UIS\$ENABLE_DISPLAY_LIST and UIS\$DISABLE_DISPLAY_LIST. You can enable and disable a display list any number of times within a program. However, to see the results of disabling a display list, you **must** execute the display list. UIS\$EXECUTE can be used to execute the display list. The following routines may also cause the display list to be executed.

13-6 Display Lists and Segmentation

Routine	Function
UIS\$CREATE_WINDOW	Creates a display window and viewport
UIS\$DELETE_OBJECT ^{1,5}	Deletes an object in the virtual display
UIS\$EXECUTE ^{2,5}	Executes the display list
UIS\$MOVE_AREA ^{3,5}	Moves a portion of the virtual display another part of the virtual display
UIS\$MOVE_WINDOW ^{4,5}	Redefines the display window coordinate space.

¹UIS\$DELETE_OBJECT executes the display list only when the object to be deleted occluded another object.

²UIS\$EXECUTE executes the entire display list, if **bufen** and **bufaddr** are not specified.

³UIS\$MOVE_AREA executes the display list only if the specified source and destination rectangles lie within a display window.

⁴UIS\$MOVE_WINDOW executes the display list only if the window size is changed.

⁵This routine checks display list flags.

The position of UIS\$DISABLE_DISPLAY_LIST and UIS\$ENABLE_DISPLAY_LIST in your program is important. If the display list is disabled **after** the display list is executed, the viewport displays all the graphic objects drawn in the virtual display. If the display list is disabled **before** one of the routines listed above is called, the viewport displays none of the graphic objects created between calls to UIS\$DISABLE_DISPLAY_LIST and UIS\$ENABLE_DISPLAY_LIST. No binary instructions were added to the display list.

Walking the Display List

You can traverse, or *walk* the entire display list from top to bottom and from object to object using UIS\$GET_ROOT_SEGMENT and UIS\$GET_NEXT_OBJECT.

Searching a Segment

If the display list contains segments, you can search the contents of any segment in the display list with UIS\$GET_NEXT_OBJECT.

Traversing the Segment Path

Because the root segment is the ultimate parent segment, every nested segment has a parent segment. The root segment acts as the parent for all level-one segments. See Figure 13-2. A segment identifier identifies the beginning of each segment in a display list. The segment identifiers within a display list comprise its *segment path*. You can traverse the segment path from the innermost segment outward with UIS\$GET_PARENT_SEGMENT.

13.3.3 Program Development I

Programming Objective

To disable a display list.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Disable the display list.
4. Draw some graphic objects in the virtual display.
5. Reenable the display list.
6. Draw some graphic objects in the virtual display.
7. Create a second display window and viewport.

```
PROGRAM LIST
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(-1.0,-1.0,50.0,50.0,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','MORE') ①
```

c Disable the display list

```
CALL UIS$DISABLE_DISPLAY_LIST(VD_ID) ②
```

c Draw the graphic objects

```
CALL UIS$CIRCLE(VD_ID,0,15.0,15.0,5.0)
CALL UIS$CIRCLE(VD_ID,0,5.0,5.0,5.0)
CALL UIS$PLOT(VD_ID,0,27.0,17.0,35.0,17.0,35.0,24.0,27.0,24.0,
2 27.0,17.0)
CALL UIS$CIRCLE(VD_ID,0,35.0,35.0,8.0)
CALL UIS$PLOT(VD_ID,0,5.0,30.0,15.0,30.0,10.0,40.0,5.0,30.0)

PAUSE
```

c Reenable the display list

```
CALL UIS$ENABLE_DISPLAY_LIST(VD_ID) ③
```

c Draw circle and triangle

```
CALL UIS$CIRCLE(VD_ID,0,33.0,35.0,8.0) ④
CALL UIS$PLOT(VD_ID,0,7.0,31.0,17.0,31.0,12.0,41.0,7.0,31.0) ⑤

WD_ID1=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','LESS') ⑥

PAUSE
END
```

13-8 Display Lists and Segmentation

Initially, a display window and viewport labelled MORE are created ❶. The world coordinate range of the window defaults to that of the virtual display.

The display list is disabled ❷.

Five graphic objects are drawn in the virtual display—three circles, a triangle, and a square. Even though all five objects appear in the viewport MORE, no entries are added to the display list.

After the PAUSE statement, the display list is reenabled ❸ and a triangle and another circle are drawn ❹ ❺.

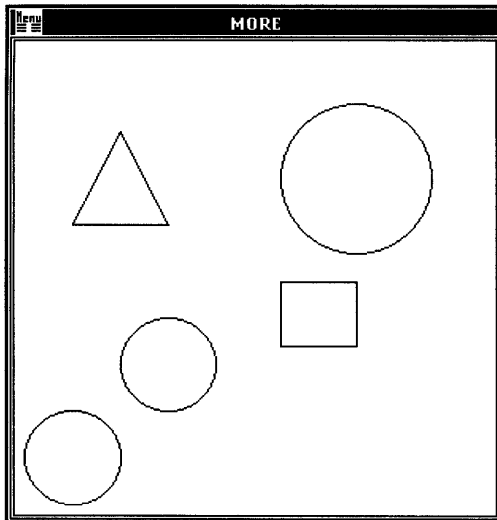
Remember the first call to `UIS$CREATE_WINDOW` was executed **before** the display list was disabled. Therefore, objects drawn in the virtual display and within the display window are displayed in the viewport, but are **not** added to the display list.

Finally, the second display window and viewport labelled LESS are created ❻. The display list is executed and all objects **except** those included within the disable-enable request appear in the viewport LESS.

13.3.3.1 Calling `UIS$DISABLE_DISPLAY_LIST` and `UIS$ENABLE_DISPLAY_LIST`

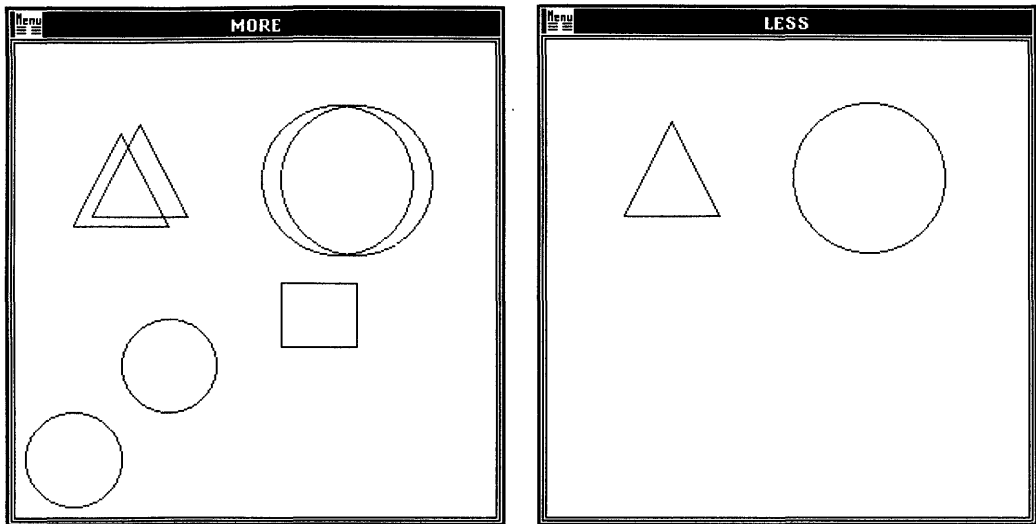
When the program executes, the viewport MORE is displayed first as shown in Figure 13-3.

Figure 13-3 Disabling a Display List



Type CONTINUE at the dollar sign prompt (\$). Figure 13-4 shows both viewports MORE and LESS. Note that the second call to UIS\$CREATE_WINDOW executes the display list.

Figure 13-4 After Display List Execution



ZK-4558-85

13.3.3.2 Program Development II

Programming Objective

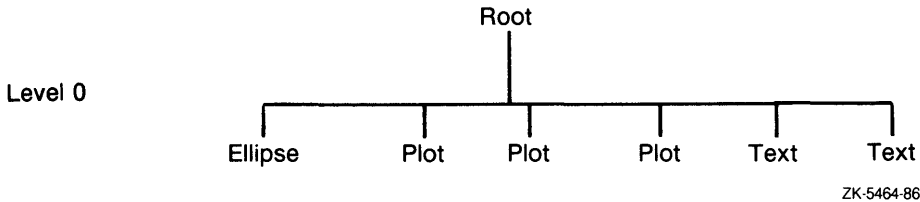
To traverse the entire display list and examine each object type.

Programming Tasks

1. Create a virtual display.
2. Draw graphic objects in the virtual display.
3. Print headings for the output in the emulation window.
4. Obtain the identifier of the root segment.
5. Walk downward through the display list.
6. Examine each object type and place its identifier in one of five arrays.

13-10 Display Lists and Segmentation

Figure 13-5 Tree Diagram—Program WALK



The program WALK draws objects in a virtual display and then identifies each object by walking the entire display list and examining the various object type values. The program also shows how you can collect and store object identifiers according to object type. If you intend to run program WALK, the subroutine DETERMINE should be compiled as a separate module and linked with WALK.

```
PROGRAM WALK
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON NEXT_ID1,TYPE1      ❶

VD_ID1=UIS$CREATE_DISPLAY(0.0,0.0,40.0,40.0,20.0,20.0)  ❷
```

C Draw objects in virtual display

```
CALL UIS$CIRCLE(VD_ID1,0,15.0,15.0,6.0)
CALL UIS$PLOT(VD_ID1,0,1.0,1.0,20.0,1.0,20.0,8.0,1.0,1.0)
CALL UIS$PLOT(VD_ID1,0,20.0,20.0,40.0,20.0,30.0,35.0,20.0,
2      20.0)
CALL UIS$PLOT(VD_ID1,0,3.0,25.0,13.0,25.0,13.0,35.0,
2      3.0,35.0,3.0,25.0)
CALL UIS$TEXT(VD_ID1,0,'The footsteps of fortune are slippery',
2      0.0,38.0)
CALL UIS$NEW_TEXT_LINE(VD_ID1,0)
CALL UIS$TEXT(VD_ID1,0,'Mirth without measure is madness')

PRINT 10
10  FORMAT(T2,'DISPLAY LIST ELEMENTS')
PRINT 20
20  FORMAT(T1,'-----')
PRINT 30
30  FORMAT(T2,'IDENTIFIER',T17,'OBJECT TYPE')

ROOT_ID1=UIS$GET_ROOT_SEGMENT(VD_ID1)  ❸
NEXT_ID1 = ROOT_ID1
```

c Walk the display list

```

DO WHILE (NEXT_ID1 .NE. 0)                                ④
TYPE1=UIS$GET_OBJECT_ATTRIBUTES(NEXT_ID1)                ⑤
CALL DETERMINE                                           ⑥
NEXT_ID1=UIS$GET_NEXT_OBJECT(NEXT_ID1)                  ⑦
ENDDO                                                    ⑧

WD_ID1=UIS$CREATE_WINDOW(VD_ID1, 'SYS$WORKSTATION')     ⑨

PAUSE
END

SUBROUTINE DETERMINE                                     ⑩
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
INTEGER*4 SEG_ARRAY(6), PLOT_ARRAY(6), TEXT_ARRAY(6), ELLIP_ARRAY(6) ⑪
INTEGER*4 LINE(6), IMAGE(6)                             ⑫
DATA H, I, J, K, L, M/1, 1, 1, 1, 1, 1, 1/             ⑬
COMMON NEXT_ID1, TYPE1                                  ⑭

IF (TYPE1 .EQ. UIS$C_OBJECT_SEGMENT) THEN               ⑮
SEG_ARRAY(H)= NEXT_ID1
PRINT 40, SEG_ARRAY(H), TYPE1
40  FORMAT(T2, I6, T19, I1, T24, 'SEGMENT')
H = H + 1
ENDIF

IF (TYPE1 .EQ. UIS$C_OBJECT_PLOT) THEN                  ⑯
PLOT_ARRAY(I) = NEXT_ID1
PRINT 50, PLOT_ARRAY(I), TYPE1
50  FORMAT(T2, I6, T19, I1, T24, 'PLOT')
I = I + 1
ENDIF

IF (TYPE1 .EQ. UIS$C_OBJECT_TEXT) THEN                  ⑰
TEXT_ARRAY(J) = NEXT_ID1
PRINT 55, TEXT_ARRAY(J), TYPE1
55  FORMAT(T2, I6, T19, I1, T24, 'TEXT')
J = J + 1
ENDIF

IF (TYPE1 .EQ. UIS$C_OBJECT_ELLIPSE) THEN              ⑱
ELLIP_ARRAY(K) = NEXT_ID1
PRINT 60, ELLIP_ARRAY(K), TYPE1
60  FORMAT(T2, I6, T19, I1, T24, 'ELLIPSE')
K = K + 1
ENDIF

```

13-12 Display Lists and Segmentation

```
IF (TYPE1 .EQ. UIS$C_OBJECT_LINE) THEN 19
LINE(L) = NEXT_ID1
PRINT 70,TEXT_LINE(L),TYPE1
70  FORMAT(T2,I6,T19,I1,T24,'NEW TEXT LINE')
L = L + 1
ENDIF

IF (TYPE1 .EQ. UIS$C_OBJECT_IMAGE) THEN 20
IMAGE(M) = NEXT_ID1
PRINT 80,IMAGE(M),TYPE1
80  FORMAT(T2,I6,T19,I1,T24,'IMAGE')
M = M + 1
ENDIF

RETURN
END
```

The variables *next_id1* and *type1* are used in both the main program and the subroutine DETERMINE. The COMMON statement ensures access to data stored in both locations by both the main program and the subroutine ① ⑭.

A virtual display is created ②. As objects are drawn in the virtual display, display list entries in the form of encoded binary data identifying the particular objects are added to the display list. Only one display list is created for each virtual display.

Because the entire display list is to be traversed, the root segment will be the starting point and its identifier must be returned ③.

A DOWHILE loop ④ ⑧ implements traversing the display list through successive calls to UIS\$GET_NEXT_OBJECT ⑦.

An object type for each display list entry is returned ⑤.

Within the DOWHILE loop the subroutine DETERMINE is called ⑥ ⑩ which sorts each object identifier according to its object type ⑮ ⑯ ⑰ ⑱ ⑳. For more information about object type symbols such as UIS\$C_OBJECT_PLOT, see UIS\$GET_OBJECT_ATTRIBUTES.

Five arrays for each object type represented in the display list are declared ⑪ ⑫. Each object identifier is stored in one of these arrays. All counter variables have been initialized to the value 1 ⑬.

A call to UIS\$CREATE_WINDOW creates a display window and viewport and executes the contents of the display list in the virtual display ⑨.

13.3.3.3 Calling UIS\$GET_NEXT_OBJECT, UIS\$GET_OBJECT_ATTRIBUTES, and UIS\$GET_ROOT_SEGMENT

The program WALK walks the display list and identifies each object in the display list. Information about each object is returned in the terminal emulation window as shown in Figure 13-6.

Figure 13-6 Display List Elements

```

$ run walk
DISPLAY LIST ELEMENTS
-----
IDENTIFIER          OBJECT TYPE
113992              UIS$C_OBJECT_SEGMENT
115328              UIS$C_OBJECT_ELLIPSE
115575              UIS$C_OBJECT_PLOT
115822              UIS$C_OBJECT_PLOT
116069              UIS$C_OBJECT_PLOT
116316              UIS$C_OBJECT_TEXT
116810              UIS$C_OBJECT_TEXT
117057              UIS$C_OBJECT_LINE
FORTRAN PAUSE
$ █

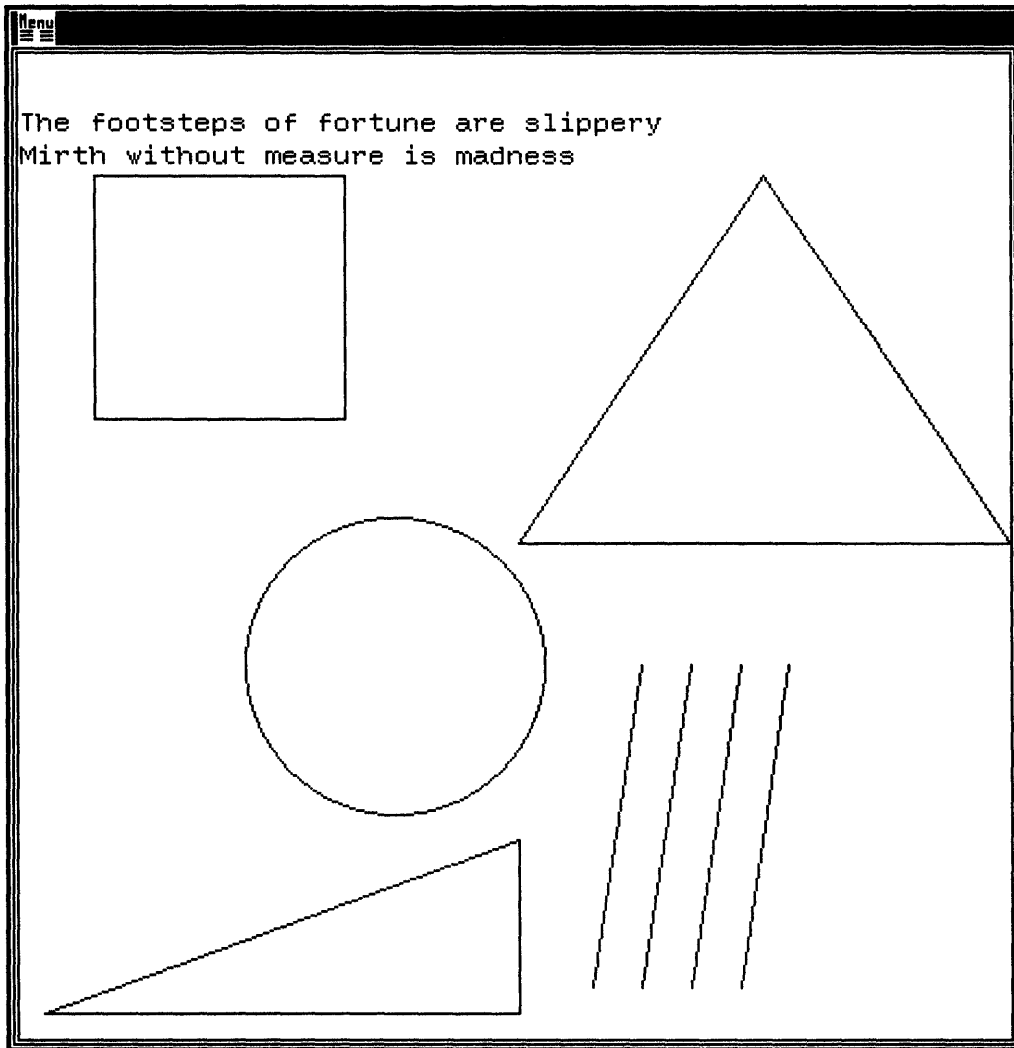
```

ZK-5255-86

The program WALK also creates a display window and viewport containing the objects in the virtual display.

13-14 Display Lists and Segmentation

Figure 13-7 Contents of the Display List



13.3.3.4 Program Development III

Programming Objective

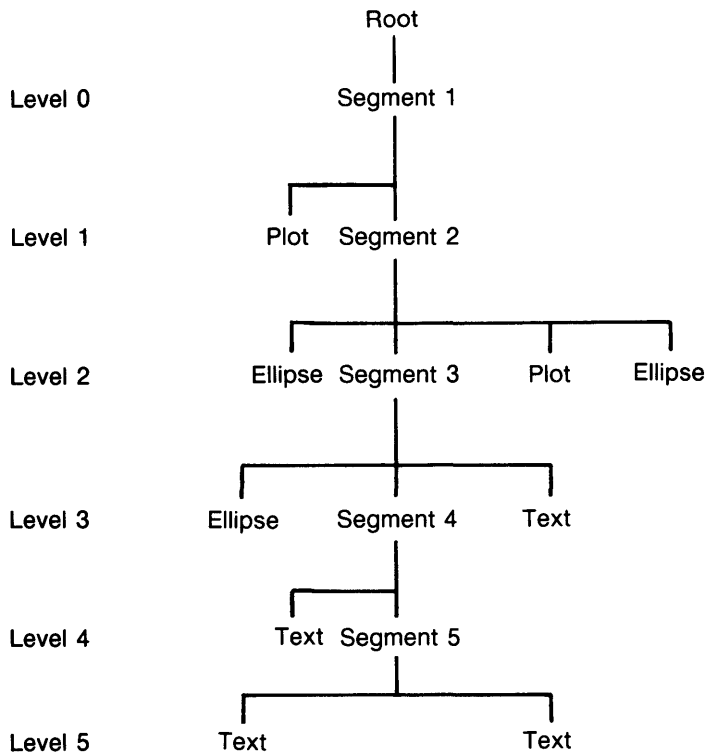
To create a display list with nested segment, traverse upward through the segment path, and then search downward through a specified segment.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Create five levels of nested segments.
4. Print the headings of the output to appear in the emulation window.
5. Beginning at the innermost nested segment, obtain and print the parent segment identifier using `UIS$GET_PARENT_SEGMENT`.
6. Print the headings of the output to appear in the emulation window.
7. Choose a segment to search.
8. Walk downward through the segment using `UIS$GET_NEXT_OBJECT`.
9. Call the subroutine `DETERMINE` to examine and store the objects in arrays by object type.

The following figure shows the structure of the display list in the program HOP.

13-16 Display Lists and Segmentation



ZK-5460-86

If you intend to run program HOP, the subroutine DETERMINE from the preceding program WALK should be compiled as a separate module and linked with HOP.

```
PROGRAM HOP
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON NEXT_ID1,TYPE1
```

```
VD_ID2=UIS$CREATE_DISPLAY(-1.0,-1.0,40.0,40.0,15.0,15.0)
```

```

SEG_ID1=UIS$BEGIN_SEGMENT(VD_ID2)
  CALL UIS$PLOT(VD_ID2,0,0.0,12.0,5.0,12.0,7.5,17.0,10.0,
2      12.0,15.0,12.0,
2      12.5,7.5,15.0,0.0,7.5,5.0,0.0,0.0,2.5,7.5,0.0,12.0)
  SEG_ID2=UIS$BEGIN_SEGMENT(VD_ID2)
  CALL UIS$CIRCLE(VD_ID2,0,7.5,8.0,8.0)
  SEG_ID3=UIS$BEGIN_SEGMENT(VD_ID2)
  CALL UIS$ELLIPSE(VD_ID2,0,25.0,8.0,5.0,8.0)
  SEG_ID4=UIS$BEGIN_SEGMENT(VD_ID2)
  CALL UIS$TEXT(VD_ID2,0,'MISERY LOVES COMPANY',
2      17.0,24.0)
  SEG_ID5=UIS$BEGIN_SEGMENT(VD_ID2)
  CALL UIS$TEXT(VD_ID2,0,'ONE SLUMBER INVITES ANOTHER',
2      1.0,39.0)
  CALL UIS$NEW_TEXT_LINE(VD_ID2,0)
  CALL UIS$TEXT(VD_ID2,0,'LIVING WELL IS THE BEST REVENGE')
  CALL UIS$END_SEGMENT(VD_ID2)
  CALL UIS$END_SEGMENT(VD_ID2)
  CALL UIS$TEXT(VD_ID2,0,'SUCCESS MAKES A FOOL SEEM WISE',
2      1.0,19.0)
  CALL UIS$END_SEGMENT(VD_ID2)
CALL UIS$PLOT(VD_ID2,0,20.0,25.0,35.0,25.0,35.0,35.0,20.0,35.0,
2      20.0,25.0)
CALL UIS$CIRCLE(VD_ID2,0,10.0,28.0,8.0)
CALL UIS$END_SEGMENT(VD_ID2)
CALL UIS$END_SEGMENT(VD_ID2)

```

C HOPPING UPWARD ALONG THE SEGMENT PATH

```

PRINT 45
45  FORMAT(T2,'SEGMENT PATH')
PRINT 55
55  FORMAT(T1,'-----')
PRINT 56
56  FORMAT(T2,'IDENTIFIER',T17,'LEVEL')

SEG_ID=SEG_ID5
I=5
PRINT 60,SEG_ID5,I

DO I=4,1,-1
PARENT_ID=UIS$GET_PARENT_SEGMENT(SEG_ID)
SEG_ID=PARENT_ID
PRINT 60,PARENT_ID,I
60  FORMAT(T2,I10,T18,I2)
ENDDO

```

13-18 Display Lists and Segmentation

C SEARCHING DOWNWARD THROUGH A NESTED SEGMENT

```
PRINT 65
65   FORMAT(T2, 'SEGMENT')
     PRINT 70
70   FORMAT(T1, '-----')
     PRINT 75
75   FORMAT(T2, 'IDENTIFIER', T17, 'OBJECT TYPE')

     NEXT_ID1=UIS$GET_NEXT_OBJECT(SEG_ID2)    ③
     DO WHILE(NEXT_ID1 .NE. 0)                ⑨
     TYPE1=UIS$GET_OBJECT_ATTRIBUTES(NEXT_ID1)
     CALL DETERMINE                            ⑩
     NEXT_ID1=UIS$GET_NEXT_OBJECT(NEXT_ID1, UIS$M_DL_SAME_SEGMENT) ⑪
     ENDDO                                     ⑫

     WD_ID2=UIS$CREATE_WINDOW(VD_ID2, 'SYS$WORKSTATION')

     PAUSE

     END
```

The program HOP contains five levels of nesting excluding the root segment. In order to walk the segment path, you must start at the innermost segment ②. The counter *l* is initialized to 5 ④ indicating the level of nesting from which you are starting.

A DO loop is declared ⑤ ⑦ containing the call to UIS\$GET_PARENT_SEGMENT ⑥. The **seg_id** argument in UIS\$GET_PARENT_SEGMENT is initialized with the segment identifier of segment 5 ③. The counter is decremented as each new parent segment identifier is returned and, in turn, is used as the **seg_id** argument in the next iteration of the loop.

The second purpose of the program is to search a specified segment. Segments are searched using **both** parameters in UIS\$GET_NEXT_OBJECT. To start at the beginning of a segment, initialize the **seg_id** to the value of the segment identifier you wish to search ③. By specifying the segment identifier of the segment you wish to search, UIS\$GET_NEXT_OBJECT returns the identifier of the next object in the segment. In this example, the second segment is chosen ①.

Another DO loop is established ⑨ ⑫ containing a call to the subroutine DETERMINE. ⑩ Note that UIS\$GET_NEXT_OBJECT ⑪ now specifies both arguments. The search will be performed on the specified segment only. If the flag UIS\$M_DL_SAME_SEGMENT were not specified, the search would proceed down to the innermost nested segment.

13.3.3.5 Calling UIS\$GET_PARENT_SEGMENT

Segment identifiers are returned beginning with the innermost nested segment as shown in Figure 13-8.

Figure 13-8 Traversing Upward Along the Segment Path

```

$ RUN HOP
SEGMENT PATH
-----
IDENTIFIER      LEVEL
      122664      5
      121576      4
      120488      3
      119400      2
      115592      1
    
```

ZK-5295-86

Object identifiers within the second-level segment are displayed as shown in Figure 13-9.

Figure 13-9 Searching Downward Through a Segment

```

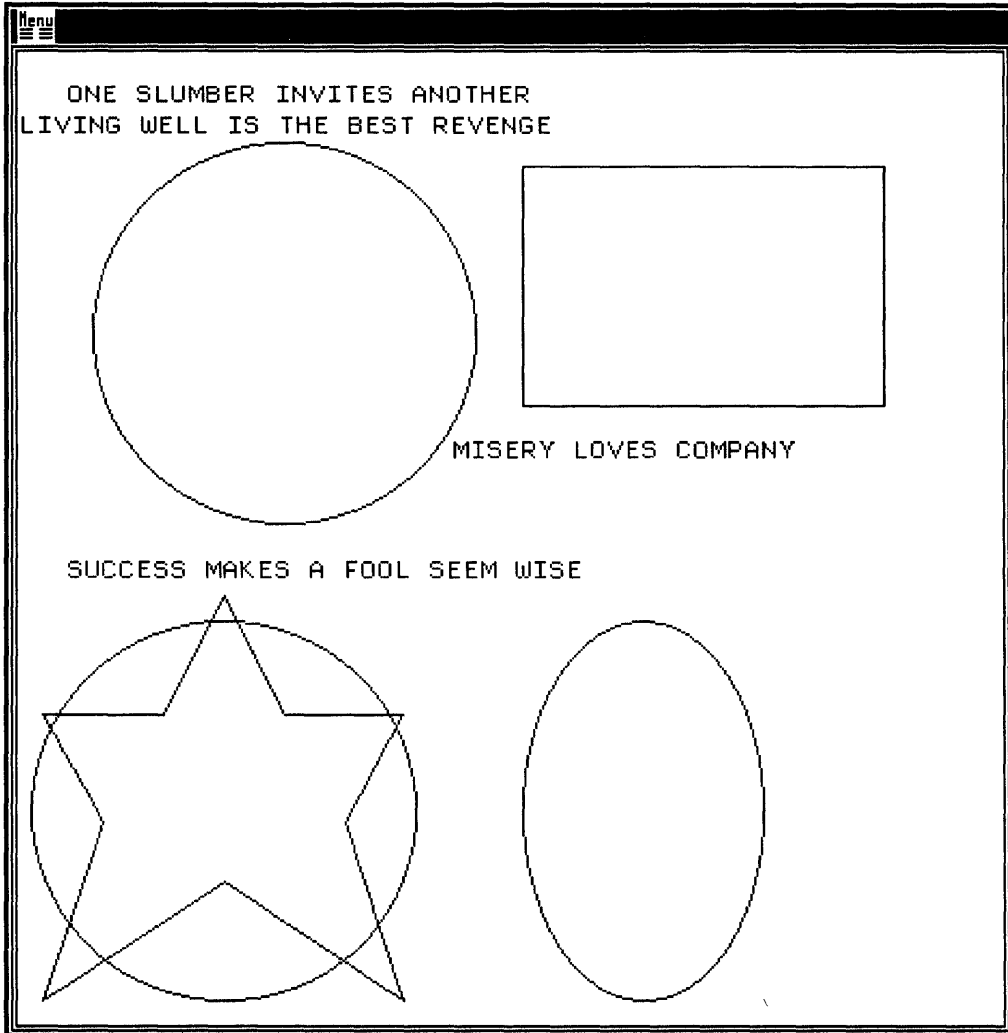
SEGMENT
-----
IDENTIFIER      OBJECT TYPE
      117175      UIS$C_OBJECT_ELLIPSE
      120488      UIS$C_OBJECT_SEGMENT
      118904      UIS$C_OBJECT_PLOT
      119151      UIS$C_OBJECT_ELLIPSE
FORTRAN PAUSE
$ █
    
```

ZK-5296-86

All the objects drawn in the virtual display are shown in Figure 13-10.

13-20 Display Lists and Segmentation

Figure 13-10 Contents of the Display List Drawn in the Virtual Display



13.4 More About Segments

When you use segments in your application programs, you are creating complex object that can be edited or searched on a segment-by-segment basis. Segments also exhibit special behavior when attribute blocks are encountered.

13.4.1 Programming Options

Other than simply creating segments, you can manipulate them as well.

Editing Display Lists

You can edit a display list that contains no explicitly defined segments as well as display lists that contain explicitly specified segments.

NOTE: You must use `UIS$SET_INSERTION_POSITION` to insert an object between existing objects in a display list.

The following routines also allow you to edit the display list.

Routine	Function
<code>UIS\$COPY_OBJECT</code>	Copies an object to another part of the display list
<code>UIS\$DELETE_OBJECT</code>	Deletes an object from the display list
<code>UIS\$INSERT_OBJECT</code>	Moves an object to another part of the display list
<code>UIS\$TRANSFORM_OBJECT</code>	Scales, rotates, and translates an object

Modifying Attribute Blocks Within Segments

As mentioned earlier, a segment consists of calls to graphics and text output routines, attribute routines, and nested segments.

When the same attribute block is modified at two different levels of nesting, modifications to the innermost attribute block take precedence over any previous modifications in outer levels. Such attribute block modifications will influence graphics and text output (where applicable) at deeper levels of nesting.

When you leave a lower-level nested segment, the original attributes of the parent segment are restored. Therefore, you can change attributes within a segment and not worry about affecting a higher-level segment.

13-22 Display Lists and Segmentation

13.4.2 Program Development I

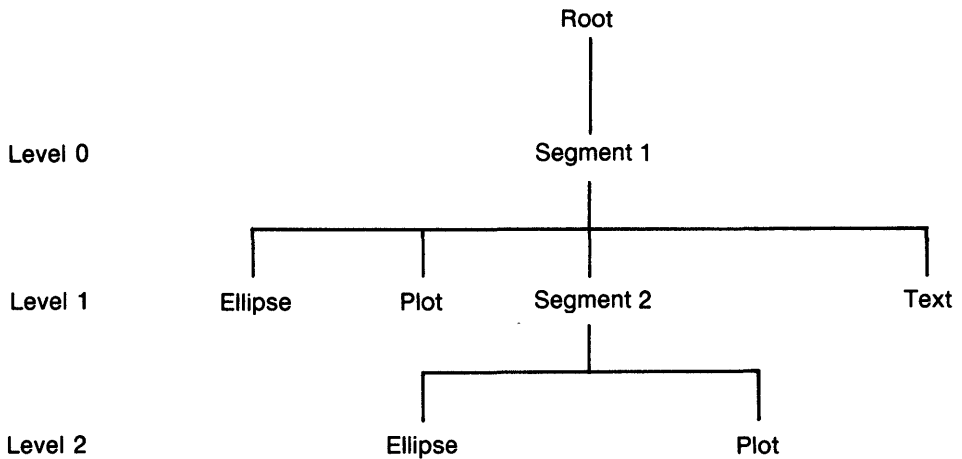
Programming Objective

To edit a display list.

Programming Tasks

1. Create a virtual display.
2. Create a series of nested segments containing calls to draw graphic objects.
3. Create a display window and viewport.
4. Delete an object in segment 1.
5. Set the editing pointer to the end of segment 1.
6. Print the headings of the output to appear in the emulation window.
7. Add a line drawing call to the end of segment 1.
8. Verify the contents of segment 1.
9. Position the pointer to the end of segment 2.
10. Add text to segment 2.
11. Verify the contents of segment 2.

Inserting an object in a specific location in the display list may not affect how the object is drawn but rather the order in which objects are drawn in the virtual display. The following diagram shows the structure of the display list before display editing in the program EDIT_LIST.



ZK-5463-86

If you intend to run program EDIT_LIST, the subroutine DETERMINE from the preceding program WALK should be compiled as a separate module and linked with EDIT_LIST.

```

PROGRAM EDIT_LIST
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON NEXT_ID1,TYPE1
  
```

```

C Create a virtual display
  VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,50.0,50.0,15.0,15.0)
  
```

13-24 Display Lists and Segmentation

c Create a segment

```
SEG_ID1=UIS$BEGIN_SEGMENT(VD_ID) ①
CALL UIS$CIRCLE(VD_ID,0,8.0,35.0,7.0) ②
CURR_ID1=UIS$GET_CURRENT_OBJECT(VD_ID) ③
CALL UIS$PLOT(VD_ID,0,17.0,27.0,32.0,27.0,24.5,42.0,17.0,27.0)
CURR_ID2=UIS$GET_CURRENT_OBJECT(VD_ID)
```

c Create another segment

```
SEG_ID2=UIS$BEGIN_SEGMENT(VD_ID) ④
CALL UIS$ELLIPSE(VD_ID,0,8.0,15.0,5.0,9.0)
CURR_ID4=UIS$GET_CURRENT_OBJECT(VD_ID)
CALL UIS$PLOT(VD_ID,0,15.0,8.0,30.0,8.0,
2 35.0,22.0,20.0,22.0,15.0,8.0)
CURR_ID5=UIS$GET_CURRENT_OBJECT(VD_ID)
CALL UIS$END_SEGMENT(VD_ID)
CALL UIS$TEXT(VD_ID,0,'The ox when weariest treads surest',
2 5.0,47.0)
CURR_ID6=UIS$GET_CURRENT_OBJECT(VD_ID)
CALL UIS$END_SEGMENT(VD_ID)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
```

PAUSE

c Delete an object from segment 1

```
CALL UIS$DELETE_OBJECT(CURR_ID1) ⑤
```

c Set the editing pointer at the end of segment 1

```
CALL UIS$SET_INSERTION_POSITION(SEG_ID1,) ⑥
CALL UIS$PLOT(VD_ID,0,29.0,42.0,44.0,42.0,36.5,27.0,29.0,42.0) ⑦
```

PRINT 20

```
20 FORMAT(T2,'CONTENTS OF SEGMENT 1')
```

PRINT 25

```
25 FORMAT(T2,'IDENTIFIER',T14,'OBJECT',T22,'TYPE')
```

PRINT 30

```
30 FORMAT('-----')
```

c Verify the contents of segment 1

```
NEXT_ID1=UIS$GET_NEXT_OBJECT(SEG_ID1)
```

```
DO WHILE(NEXT_ID1 .NE. 0)
```

```
TYPE1=UIS$GET_OBJECT_ATTRIBUTES(NEXT_ID1)
```

```
CALL DETERMINE ⑧
```

```
NEXT_ID1=UIS$GET_NEXT_OBJECT(NEXT_ID1,UIS$M_DL_SAME_SEGMENT)
```

```
ENDDO
```

PAUSE

c Set the editing pointer at the end of segment 2

```
CALL UIS$SET_INSERTION_POSITION(SEG_ID2) ⑨
CALL UIS$TEXT(VD_ID,0,'Old foxes want no tutors',
2 5.0,45.0) ⑩
```

```

PRINT 40
40  FORMAT(T2,'CONTENTS OF SEGMENT 2')
    PRINT 45
45  FORMAT(T2,'IDENTIFIER',T14,'OBJECT',T22,'TYPE')
    PRINT 50
50  FORMAT('-----')
```

c Verify the contents of segment 2

```

NEXT_ID1=UIS$GET_NEXT_OBJECT(SEG_ID2)

DO WHILE(NEXT_ID1 .NE. 0)
TYPE1=UIS$GET_OBJECT_ATTRIBUTES(NEXT_ID1)
CALL DETERMINE
NEXT_ID1=UIS$GET_NEXT_OBJECT(NEXT_ID1,UIS$M_DL_SAME_SEGMENT)
ENDDO

PAUSE
END
```

Two segments are created ❶ ❷. The second segment is nested within the first.

Successive calls to UIS\$GET_CURRENT_OBJECT ❸ retrieve an object identifier for each object in both segments. This is useful if you need to insert an object in the display list later.

A call to UIS\$DELETE_OBJECT ❹ deletes a circle ❷ from segment 1 in the display list.

The editing pointer in the display list is set at the end of segment 1 using UIS\$SET_INSERTION_POSITION ❺. A call to UIS\$PLOT is added to segment 1 ❻.

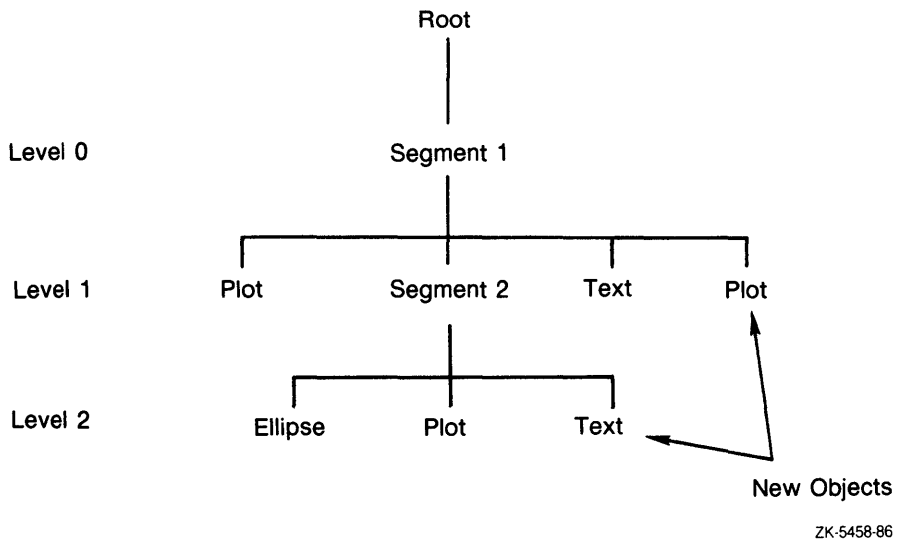
A call to the subroutine DETERMINE ❷ verifies the addition in the display list.

The editing pointer in the display list is set at the end of segment 2 using UIS\$SET_INSERTION_POSITION ❸. The binary instruction resulting from a call to UIS\$TEXT is added to segment 2 ❹.

A call to the subroutine DETERMINE ❶ verifies the changes in the display list.

The following figure shows the structure of the display list after display editing.

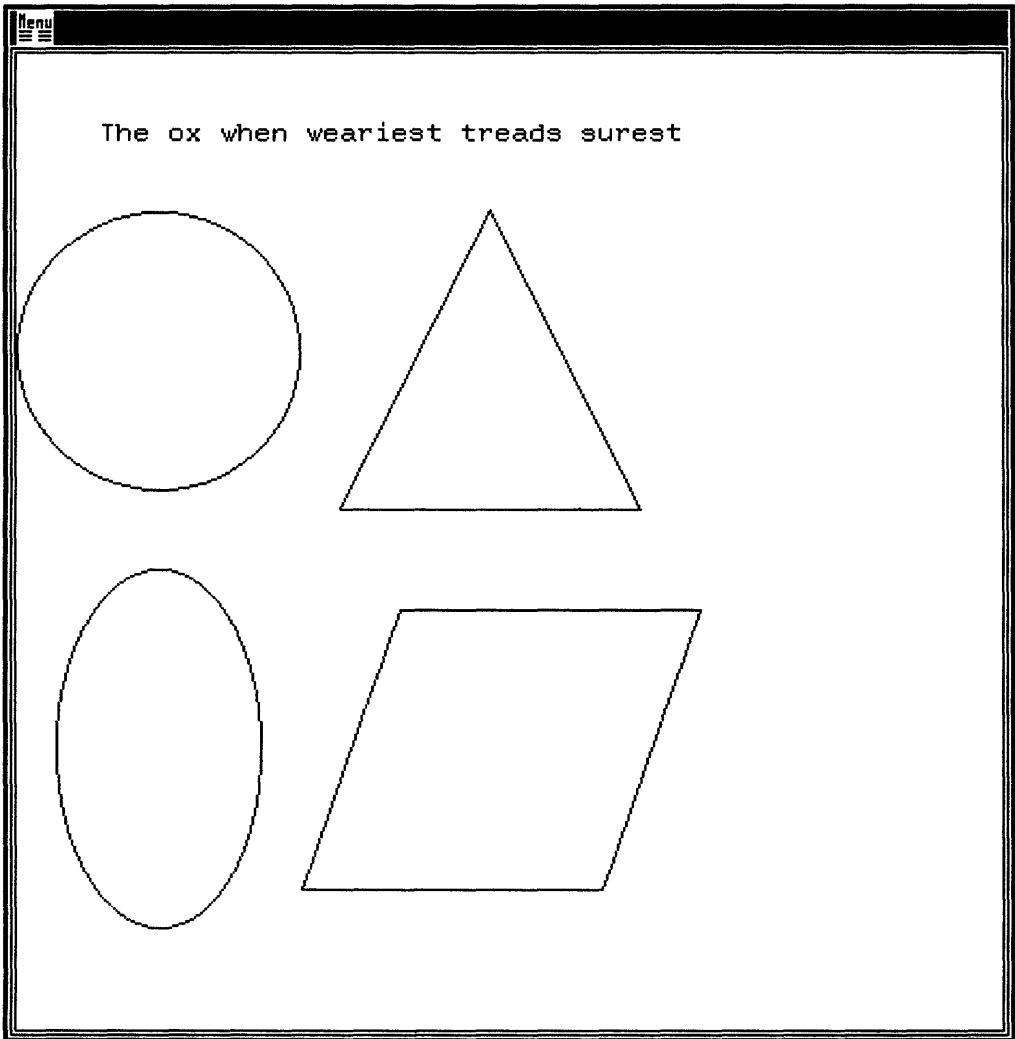
13-26 Display Lists and Segmentation



13.4.2.1 Calling `UIS$SET_INSERTION_POSITION`

The original objects, text, a circle, an ellipse, a triangle, and a parallelogram are shown in Figure 13-11.

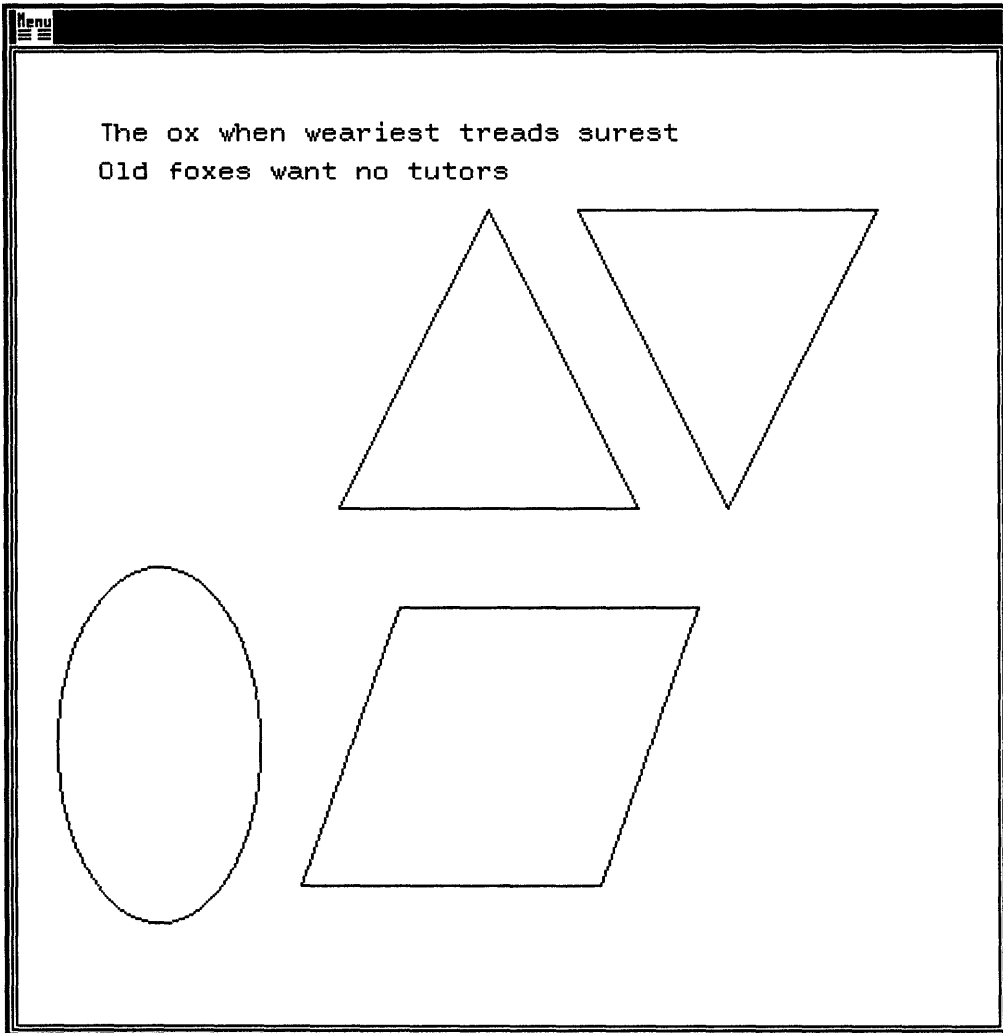
Figure 13-11 Before Display List Modification



13-28 Display Lists and Segmentation

A triangle and a line of text are added to the virtual display. The circle is deleted from the virtual display as shown in Figure 13-12.

Figure 13-12 Executing the Modified Display List



The contents of the segment are written to the emulation window as shown in Figure 13-13.

Figure 13-13 Verifying the Contents of the Display List

```

$ run edit_list
FORTRAN PAUSE
$ cont
CONTENTS OF SEGMENT 1
IDENTIFIER          OBJECT  TYPE
-----
116663              UIS$C_OBJECT_PLOT
118888              UIS$C_OBJECT_SEGMENT
117404              UIS$C_OBJECT_TEXT
117651              UIS$C_OBJECT_PLOT
FORTRAN PAUSE
$ cont
CONTENTS OF SEGMENT 2
IDENTIFIER          OBJECT  TYPE
-----
116910              UIS$C_OBJECT_ELLIPSE
117157              UIS$C_OBJECT_PLOT
116416              UIS$C_OBJECT_TEXT
FORTRAN PAUSE
$

```

ZK-5262-86

13.4.2.2 Program Development II

Programming Objective

To draw text at different levels of segmentation.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Create three levels of nested segments.
4. Modify the font character spacing attributes for each level of nesting.
5. Draw text at each level of nesting.

13-30 Display Lists and Segmentation

Font names specified in the program are logical names.

```
PROGRAM SEGMENT
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,30.0,30.0,21.0,5.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')

CALL UIS$BEGIN_SEGMENT(VD_ID) ①
  CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_6') ②
  CALL UIS$SET_CHAR_SPACING(VD_ID,1,1,0.0,1.0) ③
  CALL UIS$TEXT(VD_ID,1,'The resolved mind has no cares',0.0,30.0) ④
  CALL UIS$BEGIN_SEGMENT(VD_ID) ⑤
    CALL UIS$SET_FONT(VD_ID,1,1,'MY_FONT_13') ⑥
    CALL UIS$NEW_TEXT_LINE(VD_ID,1)
    CALL UIS$TEXT(VD_ID,1,'The camel never sees its own hump') ⑦
      CALL UIS$BEGIN_SEGMENT(VD_ID) ⑧
        CALL UIS$SET_FONT(VD_ID,1,1,'MY_FONT_7') ⑨
        CALL UIS$NEW_TEXT_LINE(VD_ID,1)
        CALL UIS$TEXT(VD_ID,1,'First things first')
        CALL UIS$END_SEGMENT(VD_ID) ⑩
  PAUSE
    CALL UIS$SET_CHAR_SPACING(VD_ID,1,1,0.0,0.0) ⑪
    CALL UIS$NEW_TEXT_LINE(VD_ID,1)
    CALL UIS$TEXT(VD_ID,1,'A new broom sweeps clean') ⑫
  CALL UIS$END_SEGMENT(VD_ID) ⑬

  CALL UIS$NEW_TEXT_LINE(VD_ID,1,)
  CALL UIS$TEXT(VD_ID,1,'No sun without a shadow') ⑭
  CALL UIS$END_SEGMENT(VD_ID) ⑮

PAUSE

END
```

The first call to `UIS$BEGIN_SEGMENT` ① and the final call to `UIS$END_SEGMENT` ⑮ establish the limits of the first-level segment. Within this segment there are two calls to `UIS$TEXT` ④ ⑭. The first call to `UIS$TEXT` establishes the current position for all text output created at the first level.

An attribute routine `UIS$SET_FONT` is called ② which modifies the font attribute. The font `MY_FONT_6` is now the current font for all text output in the first-level segment. Text created at the first level will be drawn using `MY_FONT_6`.

The calls to `UIS$BEGIN_SEGMENT` and `UIS$END_SEGMENT` ⑤ ⑬ establish the limits of the second-level segment nested within the first-level segment. The first call to `UIS$SET_FONT` ⑥ in the second-level segment references the same output attribute block number specified in the attribute routine call in the first-level segment ②. The modifications to attribute block 1 at the second level take precedence over any previous modifications of attribute block 1 at outer levels.

The second-level segment further modifies the font attribute ⑥. The font `MY_FONT_13` is now the current font for all text output in this second-level segment. The first call to `UIS$TEXT` within the second-level segment ⑦ establishes the current position for text output drawn at the second level. Calls to `UIS$TEXT` within this segment reference the same attribute block 1.

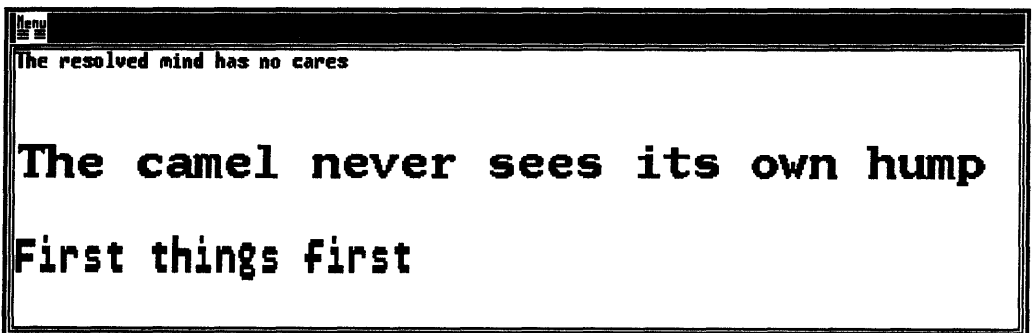
Once again, calls to `UIS$BEGIN_SEGMENT` and `UIS$END_SEGMENT` ⑧ ⑩ establish the limits of the third level of segmentation nested within the second level. The font `MY_FONT_7` is now the current font for all text output in this segment ⑨.

The line spacing component of the character spacing attribute was modified twice ③ ⑪. The first call to `UIS$SET_CHAR_SPACING` was made to increase the line spacing by a factor of 1. As the program executes, the second text drawing routine call in levels 1 and 2 ⑭ ⑫ require room to avoid overstriking existing lines.

13.4.2.3 Calling `UIS$BEGIN_SEGMENT` and `UIS$END_SEGMENT`

As the program `SEGMENT` executes each instruction sequentially, a text string is drawn in the virtual display at the first, second, and third levels of segmentation as shown in Figure 13-14. Please note the font used in text creation.

Figure 13-14 Text Output During Execution

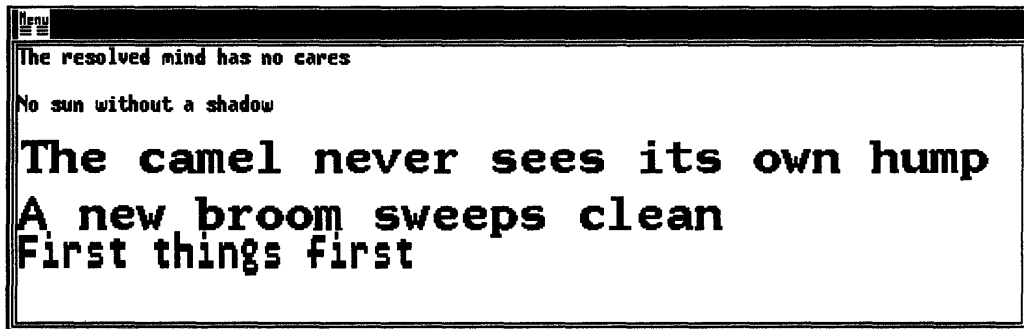


ZK-4569-85

Text strings are then created in the reverse order of segmentation—second level and then first level. Please note the font used and the order of text string creation as shown in Figure 13-15 as compared to the statements in the source program.

13-32 Display Lists and Segmentation

Figure 13-15 Final Text Output



ZK-4560-85

Chapter 14

Geometric and Attribute Transformations

14.1 Overview

Transformations alter the appearance of graphic objects and text. In Part I, viewing transformations and their possibly distorting effects on graphic objects were discussed. Already in Part II you have seen the effects of world coordinate transformations when you modify the world coordinate space and then redraw graphic objects in the new space. This chapter describes the following two types of transformations:

- Two-dimensional geometric transformations
- Attribute transformations

14.2 Geometric Transformations

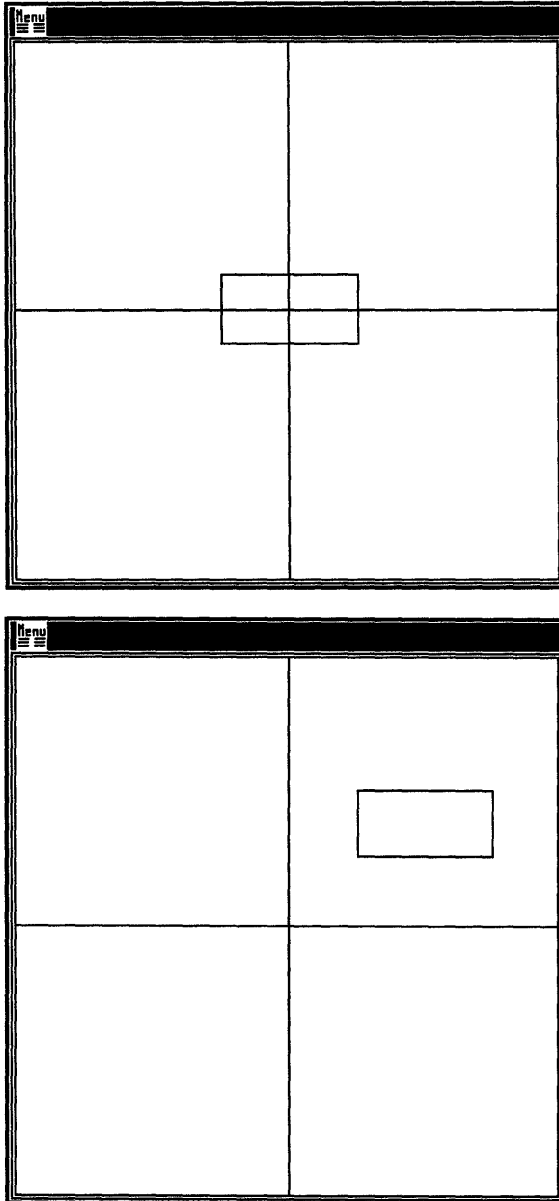
A two-dimensional geometric transformation of a graphic object involves changing the graphic object's angular orientation or its shape within the virtual display. The coordinate system is not modified. Graphic objects are geometrically transformed using the following methods: scaling, translation, and rotation.

14.2.1 Translating Graphic Objects

Translating a graphic object involves moving the object to another part of the coordinate space without altering its physical orientation with respect to the x and y axes. For example, a side of a triangle that was originally parallel to the y axis remains parallel to that axis even if the object is moved to another quadrant in the coordinate space.

14-2 Geometric and Attribute Transformations

Figure 14-1 Translating a Graphic Object



14.2.2 Scaling Graphic Objects

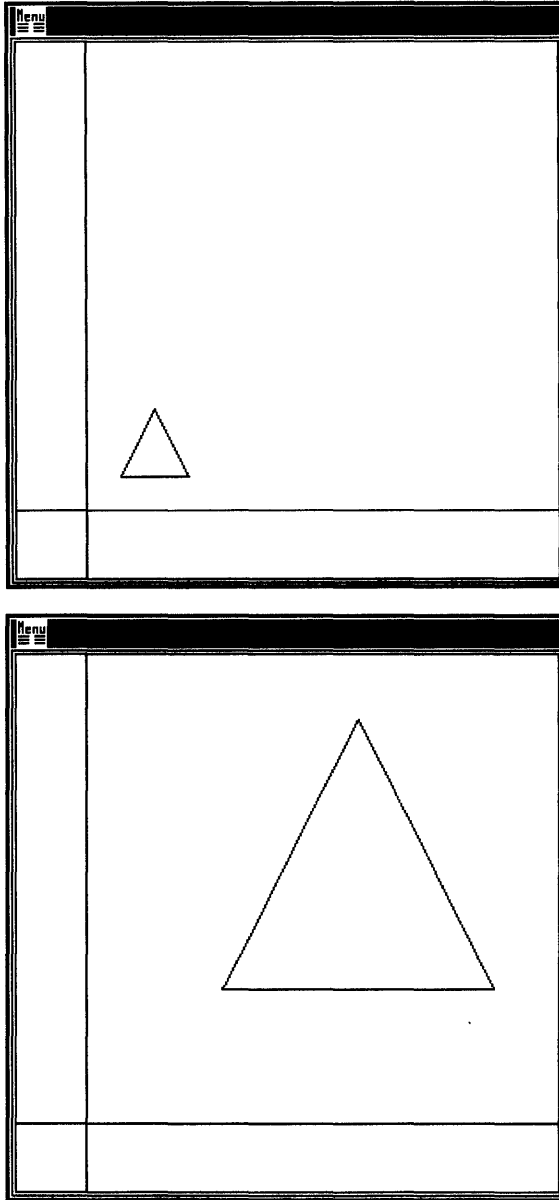
Typically, scaling involves stretching or shrinking a graphic object. Scaling a graphic object can occur in two ways: (1) simple scaling of the graphic object in the virtual display or (2) complex scaling.

Simple Scaling of Graphic Objects

Simple scaling of a graphic object involves executing a single transformation. The position of the newly scaled graphic object in the virtual display is always different from its original position with one exception. If the object's center point is at the origin, the object will not move when scaled.

14-4 Geometric and Attribute Transformations

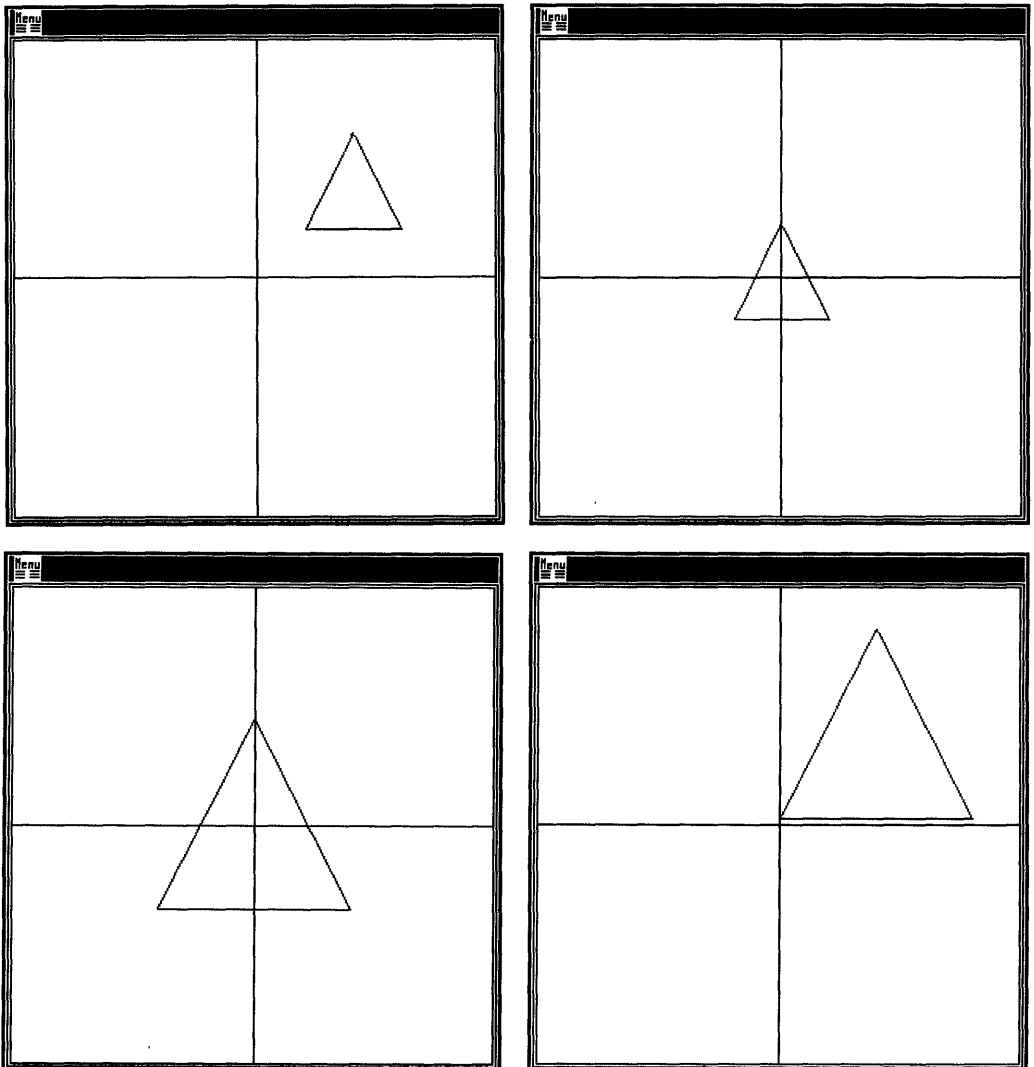
Figure 14-2 Simple Scaling



Complex Scaling of Graphic Objects

Complex scaling of graphic objects ensures that the newly scaled object maintains its previous position in the virtual display. The center of the object is first translated to the origin of the coordinate system, scaled, and finally translated to its original position.

Figure 14-3 Complex Scaling

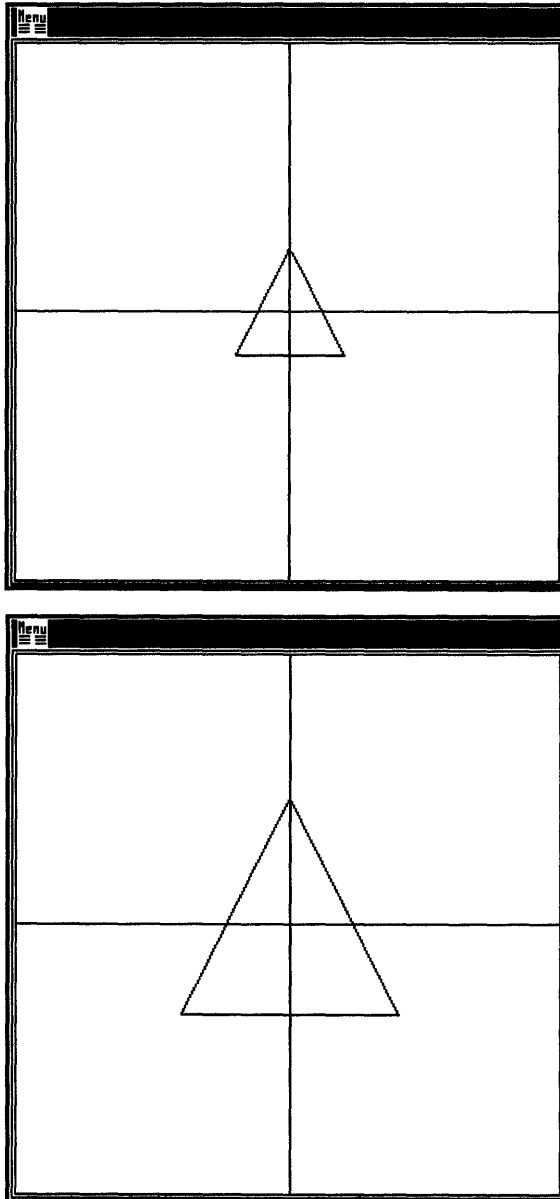


14-6 Geometric and Attribute Transformations

14.2.2.1 Uniformly Scaled Graphic Objects

For example, a photographic enlargement of a snapshot to poster size renders an object whose physical dimensions are proportional to the snapshot. In such a case, the scaling factor of the width of the object, S_x , equals the scaling factor of the height of the object, S_y .

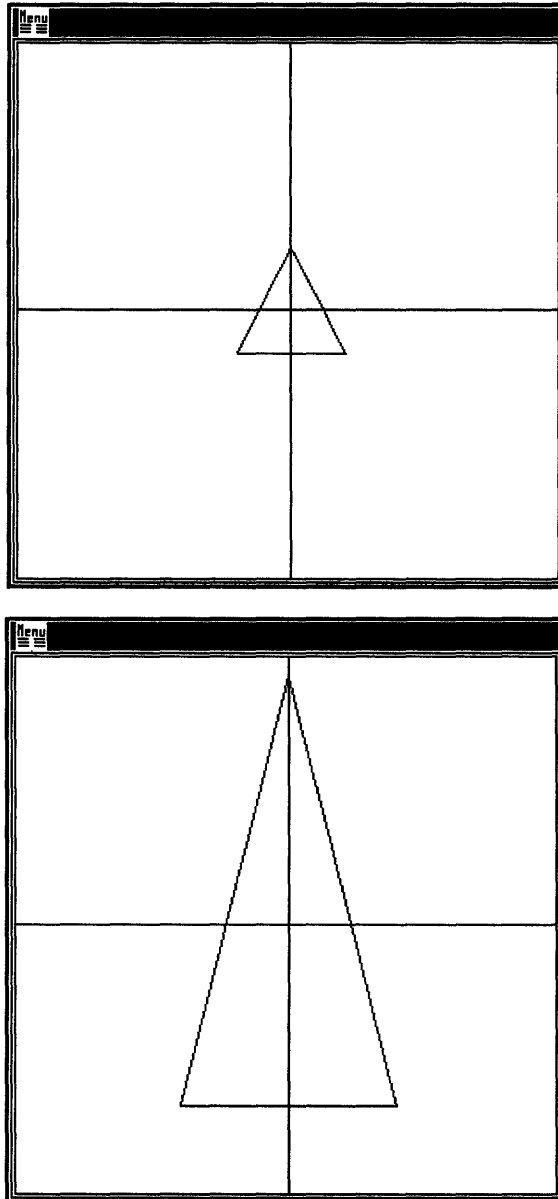
Figure 14-4 Uniformly Scaling a Graphic Object



14.2.2.2 Differentially Scaled Graphic Objects

Scaling need not be performed uniformly. For example, the height of an object may be increased while its width remains constant where s_x does not equal s_y . The object is differentially scaled as shown in Figure 14-5.

Figure 14-5 Differentially Scaling a Graphic Object



14-8 Geometric and Attribute Transformations

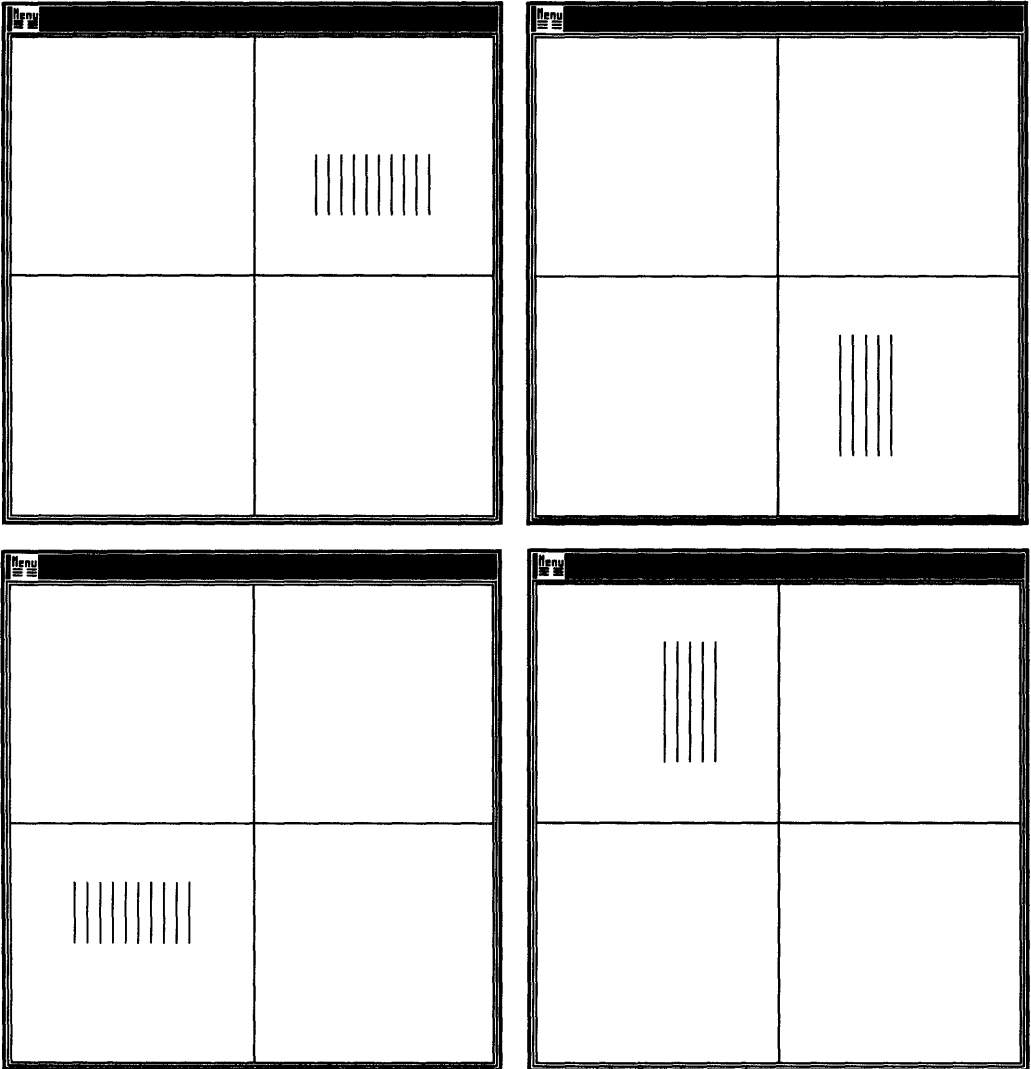
14.2.3 Rotating Graphic Objects

Generally speaking, rotation changes an object's angular orientation in the virtual display. All rotations occur about the origin of the coordinate system. Positive rotation is a counterclockwise movement.

Simple Rotation of Graphic Objects

Simple rotation of graphic objects involves executing a single transformation—no translation. With simple rotation, the object appears to revolve about the origin. Figure 14-6 shows rectangle rotating about the origin.

Figure 14-6 Simple Rotation of a Graphic Object



ZK 5399 86

Complex Rotation of Graphic Objects

Complex rotation can occur when the reference or pivotal point is the center of the object. Complex rotation of the graphic object is accomplished by first

14-10 Geometric and Attribute Transformations

translating the object to the origin so that the origin and reference point share the same coordinate values—(0.0,0.0). The object is rotated and translated to its original position in the virtual display. Figure 14-7 illustrates complex rotation of a rectangle.

14.2.4 Programming Options

You can perform geometric transformations of two types.

Two-Dimensional Geometric Transformation—COPY

You may execute a geometric transformation where the graphic object is copied using `UIS$COPY_OBJECT`. The original object remains unchanged.

Two-Dimensional Geometric Transformations—MOVE

You may execute a geometric transformation where the graphic object is transformed in the virtual display using `UIS$TRANSFORM_OBJECT`. The original object is modified.

14.2.5 Program Development I

Programming Objective

To rotate a graphic object in a positive counterclockwise 45 degrees about its center.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Create a graphic object and obtain its identifier.
4. Declare and load a two-dimensional array with translation values.
5. Execute translation.
6. Load array with rotation values.
7. Execute rotation.
8. Load array with translation values.
9. Execute the translation where the original object is erased and redraw the object in its original position in the coordinate system.

```

PROGRAM GEO_TRANSFORM_ROT
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL*4 MATRIX(2,3) ❶

VD_ID=UIS$CREATE_DISPLAY(-20.0,-20.0,20.0,20.0,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')

CALL UIS$PLOT(VD_ID,0,0.0,20.0,0.0,-20.0) ❷
CALL UIS$PLOT(VD_ID,0,-20.0,0.0,20.0,0.0) ❸

CALL UIS$PLOT(VD_ID,0,5.0,5.0,15.0,5.0,15.0,10.0,5.0,10.0,
2          5.0,5.0) ❹

CURRENT_ID=UIS$GET_CURRENT_OBJECT(VD_ID) ❺
OBJ_ID=CURRENT_ID

PAUSE

MATRIX(1,1)=1.0 ❻
MATRIX(2,1)=0.0
MATRIX(1,2)=0.0
MATRIX(2,2)=1.0
MATRIX(1,3)=-10.0
MATRIX(2,3)=-7.5
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX) ❼

PAUSE

MATRIX(1,1)=COSD(45.0) ❽
MATRIX(2,1)=-SIND(45.0)
MATRIX(1,2)=SIND(45.0)
MATRIX(2,2)=COSD(45.0)
MATRIX(1,3)=0.0
MATRIX(2,3)=0.0
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX) ❾

PAUSE

MATRIX(1,1)=1.0 ❿
MATRIX(2,1)=0.0
MATRIX(1,2)=0.0
MATRIX(2,2)=1.0
MATRIX(1,3)=10.0
MATRIX(2,3)=7.5
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX) ⓫

PAUSE
END

```

A two-dimensional array is declared ❶.

14-12 Geometric and Attribute Transformations

The x and y axes are drawn ② ③.

A rectangle is drawn using `UIS$PLOT` ④. Call `UIS$GET_CURRENT_OBJECT` to save its object identifier ⑤. The object identifier is used as an argument to the transformation routine.

The rectangle will be rotated about its center.

The VAX FORTRAN intrinsic functions `SIND` and `COSD` accept degrees as arguments ⑧.

The matrix is loaded with values three times ⑥ ⑧ ⑩ to translate, rotate the rectangle about its center, and then translate it to its original position in the virtual display.

Each transformation is performed as the original object is erased and redrawn in its new orientation. The rectangle is redrawn with each call to `UIS$TRANSFORM_OBJECT` ⑦ ⑨ ⑪.

14.2.6 Calling `UIS$TRANSFORMATION_OBJECT`

The program `GEO_TRANSFORM_ROT` translates, rotates, and translates a rectangle using `UIS$TRANSFORM_OBJECT`. With each transformation, the rectangle's previous position in the virtual display is erased as shown in Figure 14-7.

14.2.7 Program Development II

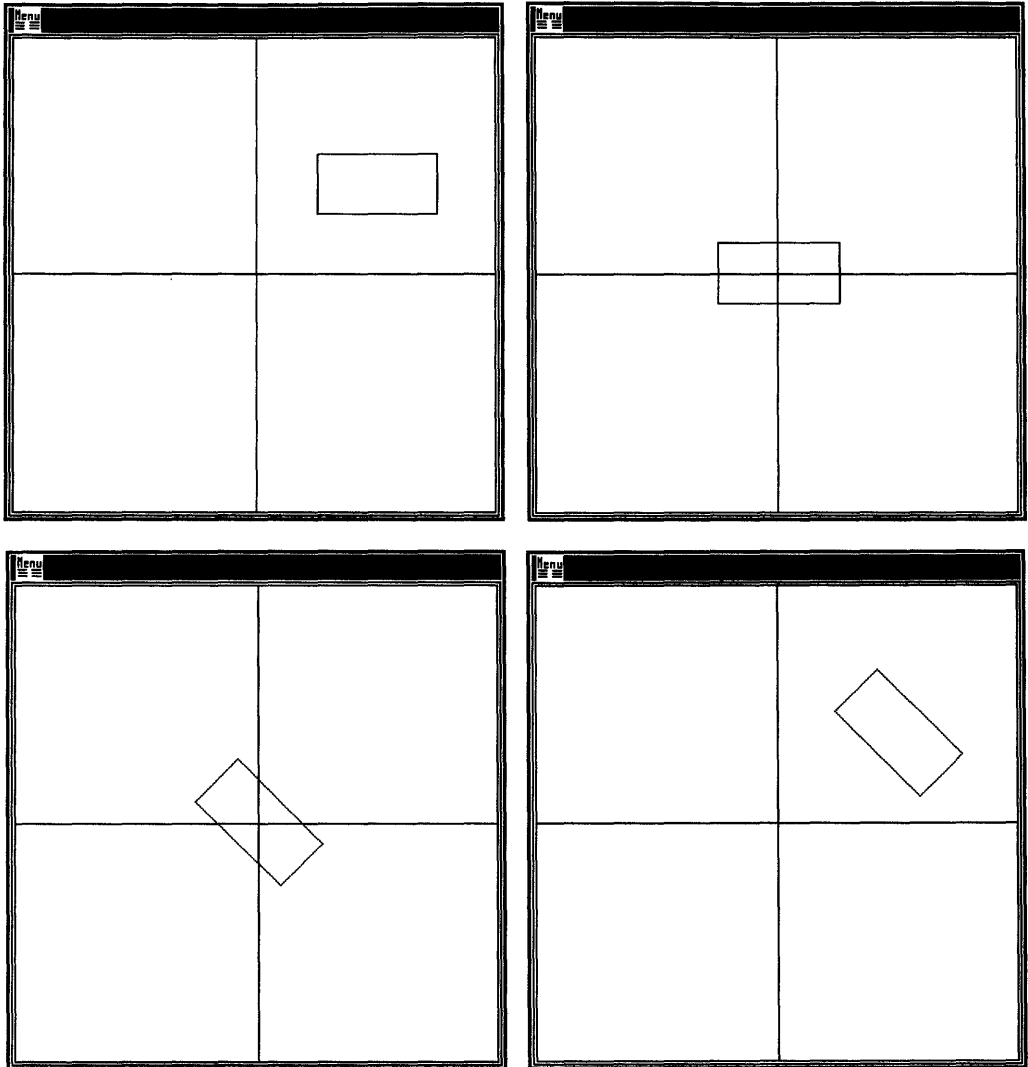
Programming Objective

To rotate a copy of the graphic object 45 degrees about its center and place the rotated copy in another quadrant.

Programming Tasks

1. Create a virtual display.
2. Create a display window and viewport.
3. Declare and load a two-dimensional array with translation values.
4. Execute the `COPY` operation and the translation.
5. Load the array with rotation values.
6. Execute rotation.
7. Load array with translation values.
8. Execute translation.

Figure 14-7 Complex Rotation of a Rectangle



14-14 Geometric and Attribute Transformations

```
PROGRAM COPY_OBJECT
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL*4 MATRIX(2,3)

VD_ID=UIS$CREATE_DISPLAY(-20.0,-20.0,20.0,20.0,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')

CALL UIS$PLOT(VD_ID,0,0.0,20.0,0.0,-20.0)
CALL UIS$PLOT(VD_ID,0,-20.0,0.0,20.0,0.0)

CALL UIS$PLOT(VD_ID,0,5.0,5.0,15.0,5.0,10.0,10.0,5.0,5.0)

CURRENT_ID=UIS$GET_CURRENT_OBJECT(VD_ID)
OBJ_ID=CURRENT_ID

PAUSE

MATRIX(1,1)=1.0
MATRIX(2,1)=0.0
MATRIX(1,2)=0.0
MATRIX(2,2)=1.0
MATRIX(1,3)=-10.0
MATRIX(2,3)=-7.5
COPY_ID=UIS$COPY_OBJECT(OBJ_ID,MATRIX) ❶

PAUSE

OBJ_ID=COPY_ID ❷

MATRIX(1,1)=COSD(45.0) ❸
MATRIX(2,1)=-SIND(45.0)
MATRIX(1,2)=SIND(45.0)
MATRIX(2,2)=COSD(45.0)
MATRIX(1,3)=0.0
MATRIX(2,3)=0.0
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX) ❹

PAUSE

MATRIX(1,1)=1.0
MATRIX(2,1)=0.0
MATRIX(1,2)=0.0
MATRIX(2,2)=1.0
MATRIX(1,3)=-10.0 ❺
MATRIX(2,3)=7.5
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX)

PAUSE
END
```


This program is almost identical to the previous program `GEO_TRANSFORM_ROT` with a few important differences.

The first transformation is executed ❶. The triangle is copied and translated to the origin of the coordinate space. The coordinates of the center of the triangle match those of the origin. The original triangle in the first quadrant remains unchanged.

The identifier of the transformed object *copy_id* is assigned to the *obj_id* ❷. It will be used as an argument in the next transformation.

The VAX FORTRAN intrinsic functions `SIND` and `COSD` accepts degrees as arguments ❸.

A call to `UIS$TRANSFORM_OBJECT` rotates the translated triangle 45 degrees ❹. The original object is erased and redrawn in its new orientation.

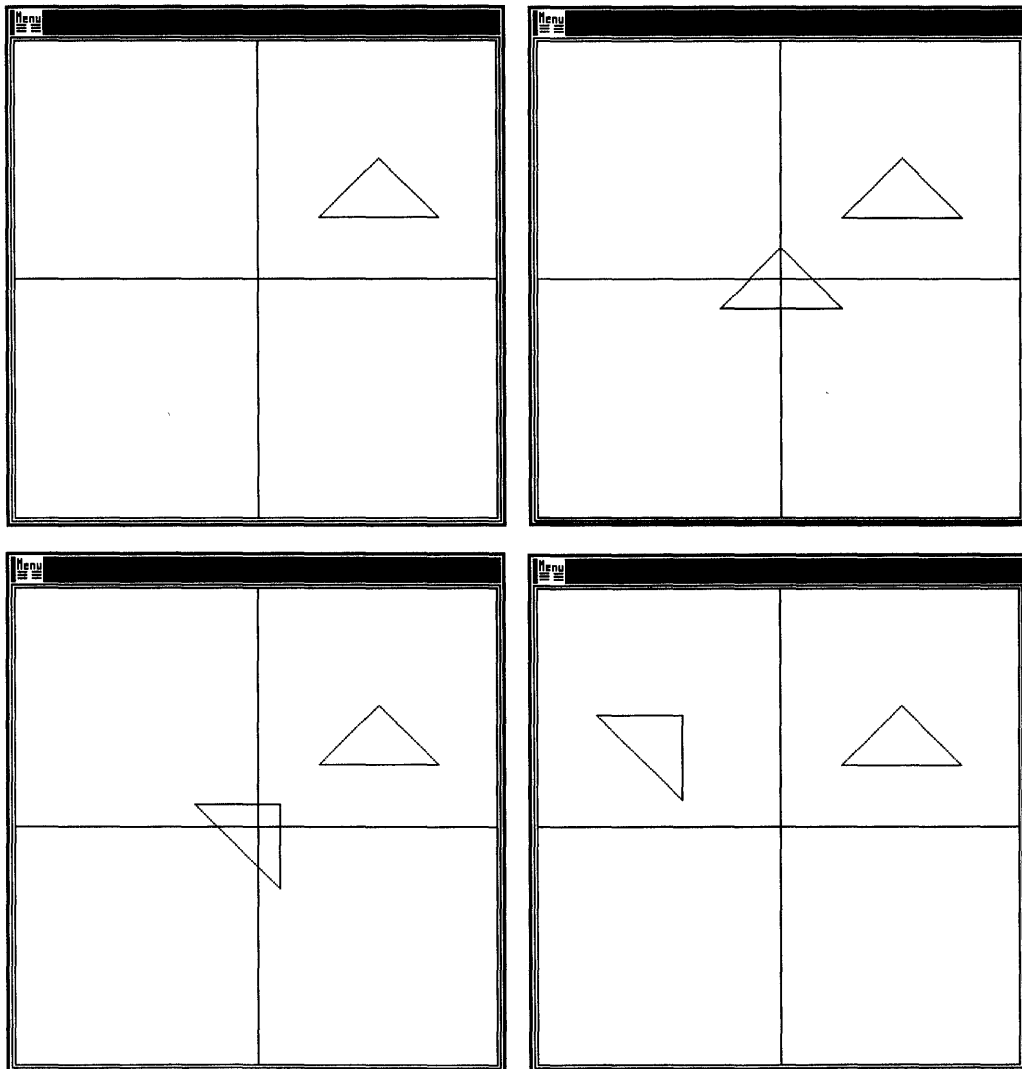
The final translation of the triangle places it in the second quadrant at a 45-degree angle to the original triangle ❺.

14.2.8 Calling `UIS$COPY_OBJECT`

The triangle is transformed similarly to the rectangle in the previous example. However, the first transformation copies the triangle. Figure 14-8 shows that the triangle still remains in the virtual display. However, the rotated copy of the triangle is translated to the second quadrant.

14-16 Geometric and Attribute Transformations

Figure 14-8 Complex Rotation of a Triangle



14.3 Attribute Transformations

Attribute transformations involve modifying graphic objects and text without having to know the attribute block of the original graphics or text objects.

14.3.1 Programming Options

Attribute Transformations

Ordinarily, when you modify the appearance of an existing graphic object, you must perform the follow procedure:

1. Obtain the object identifier.
2. Call `UIS$DELETE_OBJECT` with the object identifier.
3. Redraw the graphic object or text using the modified attribute block.

At the very least, you must use two steps—erase the virtual display using `UIS$ERASE` and redraw the object with a modified attribute block.

A call to `UIS$COPY_OBJECT` or `UIS$TRANSFORM_OBJECT` specifying the **atb** argument and omitting the **matrix** argument lets you modify the attributes of graphic objects and text in a single call.

To disable attribute transformations, omit the **atb** argument in `UIS$COPY_OBJECT` or `UIS$TRANSFORM_OBJECT`.

14.3.2 Program Development

Programming Objective

To modify the fill pattern of a circle as a transformation.

Programming Tasks

1. Create a virtual display.
2. Create a display window and a display viewport.
3. Draw a circle using default attributes.
4. Obtain its object identifier.

14-18 Geometric and Attribute Transformations

5. Modify the fill pattern attribute.
6. Transform the circle's attributes and draw the modified circle.

```
PROGRAM ATTR_TRANS
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

VD_ID=UIS$CREATE_DISPLAY(-10.5,-10.5,10.5,10.5,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')

CALL UIS$CIRCLE(VD_ID,0,0.0,0.0,10.0)
CURRENT_ID=UIS$GET_CURRENT_OBJECT(VD_ID)
OBJ_ID=CURRENT_ID

CALL UIS$SET_FONT(VD_ID,0,1,'UIS$FILL_PATTERNS') ❶
CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_DOWNDIAG1_7) ❷

PAUSE

CALL UIS$TRANSFORM_OBJECT(OBJ_ID,,1) ❸

PAUSE
END
```

A matrix is not declared in this program. Therefore, the position of any objects drawn will be the same.

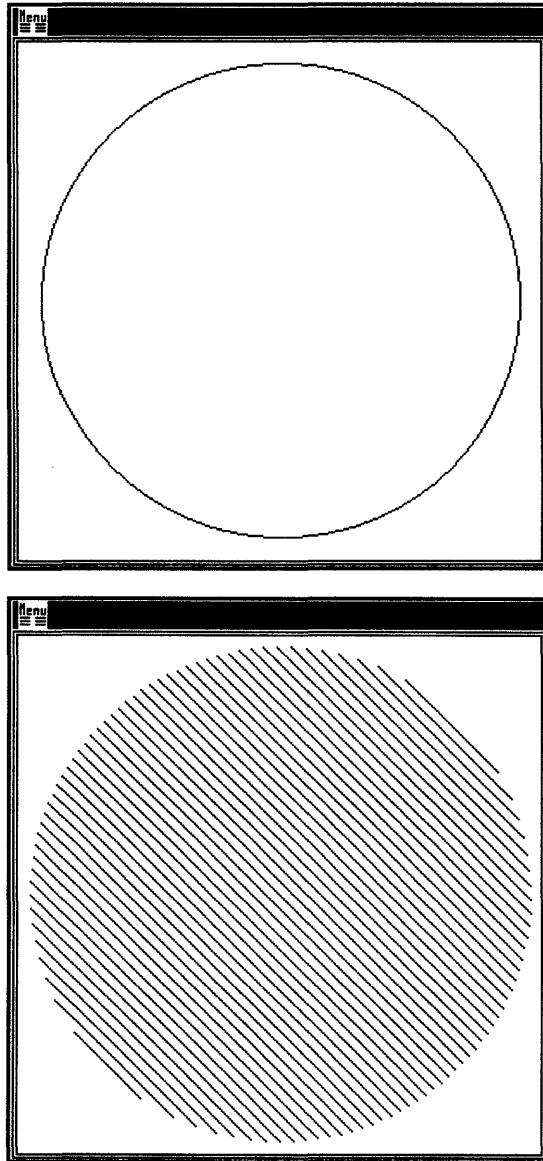
The fill pattern attribute is modified ❶ ❷.

The object identifier of the original circle and attribute block number of the newly modified attribute block are arguments in the transformation ❸.

14.3.3 Requesting Attribute Transformations

Because no matrix was specified in the transformation, the resulting transformation will not cause objects to change their positions within the virtual display. The original circle is erased and the modified circle is placed in its position as shown in Figure 14-9.

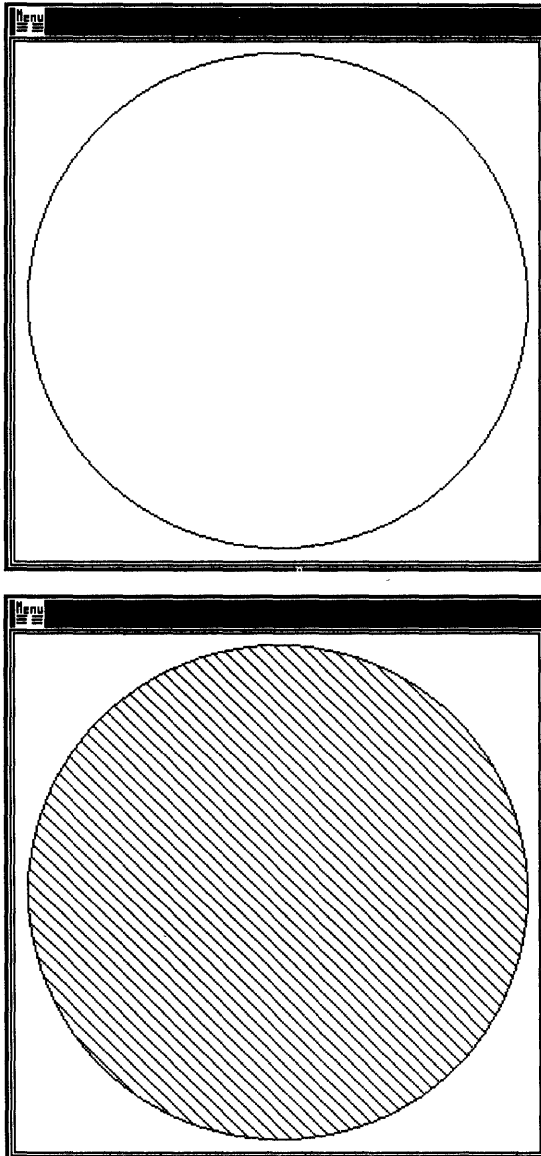
Figure 14-9 Modifying Attributes with a Transformation



14-20 Geometric and Attribute Transformations

If the call to `UIS$TRANSFORM_OBJECT` were instead a call to `UIS$COPY_OBJECT`, the original circle would remain visible in the virtual display. The modified circle would still be placed in the same position.

Figure 14-10 Modifying Attributes with a Copy



Chapter 15

Metafiles and Private Data

15.1 Overview

Many of your applications produce displays that you might wish to use again. In order to reexecute these displays you must first store them in a UIS metafile. We will describe the structure of a metafile and the contents of the binary encoded instructions in more detail.

An additional feature allows you to associate data with your graphics objects. You can specify a particular graphic object or group of objects within the display to be associated with the user-defined data. This chapter discusses metafiles and private data in the following topics:

- Extracting data from a display list
- Interpreting the user buffer
- Creating a UIS metafile
- Creating private data

Hardcopy UIS (HCUIS) translates UIS pictures to other formats. See the *MicroVMS Workstation Guide to Printing Graphics* for more information about HCUIS.

15.2 Display Lists and UIS Metafiles

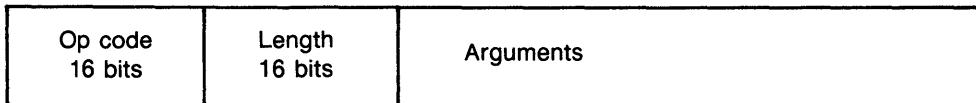
Generating graphic objects on the display screen is the purpose of your application programs. As a programmer, you are also concerned with program modularity and efficiency. With each new object drawn in the virtual display, a new entry is added to the display list. Preserving the contents of a display list as *generically encoded* binary instructions for use across many applications is highly desirable. Graphics output and attribute modifications can then be extracted from display lists and stored in user-defined buffers as *metafile components* and in files as *metafiles*.

15-2 Metafiles and Private Data

15.2.1 Generic Encoding of Graphics and Attribute Routines

As mentioned earlier, whenever an object is drawn in the virtual display or an attribute is modified, a binary encoded instruction is added to the display list of the specified virtual display. Entries in the display list are variable length instructions and are encoded as shown in Figure 15-1.

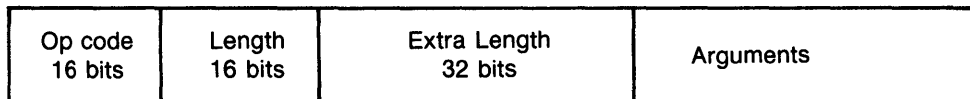
Figure 15-1 Binary Encoded Instruction



ZK-5472-86

If the length of the binary encoded instruction is greater than 32,767 bytes, the length field should be set equal to `UIS$C_LENGTH_DIFF` and the extra length should be set equal to the total number of bytes in the instruction. Figure 15-2 describes the format of a display list entry, if the length field is greater than 32,767 bytes.

Figure 15-2 Extended Binary Encoded Instruction



ZK-5473-86

15.2.1.1 Normalized Coordinates

The coordinate system used within display lists and when creating generically encoded streams is *normalized coordinates*. Normalized coordinates are floating point numbers in the range (0.0,0.0) to (`max_nc_x`,`max_nc_y`) where (0.0,0.0) refers to lower-left corner of the virtual display and (`max_nc_x`,`max_nc_y`) refers to the upper-right corner.

Normalized coordinates are used within UIS as a means of deferring the actual mapping of an application's world coordinates to device-specific coordinates until the actual output device is known. For example, the device coordinates of a printer may be very different from the device coordinates of a raster display.

15.2.1.2 Interpreting the User Buffer

When UIS routine calls are executed, binary encoded instructions are added to the display list. When you extract the contents of a display list and store them in a buffer, you have created metafile components—header data, an encoded stream of binary instructions, and trailer data. Each metafile component consists of binary encoded instructions. If you write the contents of the buffer to a file, you have created a UIS metafile. A UIS metafile is a *generically encoded* binary stream, that is, all three components exist within a single file and the file is executable on any VAXstation system. The contents of the buffer and metafile contains values that describe the extracted objects. If reexecuted, these encoded instructions cause UIS to recreate the objects drawn in the virtual display. Note that monochrome systems cannot duplicate the color of extracted objects created on color systems.

It is possible to write your own binary encoded instructions and metafiles. First, you must understand how to interpret the contents of the user-defined buffer containing the extracted data.

Opcodes

The portion of the binary encoded instruction that specifies the action that the instruction performs is the opcode. Table 15-1 lists the generic encoding symbols and the corresponding opcodes of binary encoded instructions.

Table 15-1 Generic Encoding Symbols and Opcodes

Generic Encoding Symbol	Opcode
Attribute	
GER\$_SET_WRITING_MODE	1
GER\$_SET_WRITING_INDEX	2
GER\$_SET_BACKGROUND_INDEX	3
GER\$_SET_CHAR_SPACING	4
GER\$_SET_CHAR_SLANT	5
GER\$_SET_TEXT_SLOPE	6
GER\$_SET_TEXT_PATH	7
GER\$_SET_TEXT_FORMATTING	11
GER\$_SET_CHAR_ROTATION	12
GER\$_SET_TEXT_MARGINS	13
GER\$_SET_LINE_WIDTH	14
GER\$_SET_LINE_STYLE	15
GER\$_SET_FONT	17
GER\$_SET_ARC_TYPE	26

15-4 Metafiles and Private Data

Table 15-1 (Cont.) Generic Encoding Symbols and Opcodes

Generic Encoding Symbol	Opcode
Attribute	
GER\$_SET_FILL_PATTERN	37
GER\$_SET_CLIP	38
GER\$_SET_CHAR_ENCODING	39
GER\$_SET_CHAR_SIZE	42
Graphics and Text	
GER\$_TEXT	19
GER\$_SET_POSITION	21
GER\$_PLOT	23
GER\$_ELLIPSE	25
GER\$_IMAGE	29
GER\$_ALIGN_POSITION	33
GER\$_LINE	52
Application-specific Private Data	
GER\$_PRIVATE	30
Display List	
GER\$_BEGIN ¹	31
GER\$_END ¹	32
GER\$_BEGIN_DISPLAY	34
GER\$_END_DISPLAY ¹	35
GER\$_VERSION	36
GER\$_IDENTIFICATION	43
GER\$_DATE	44
GER\$_NOP ¹	45
GER\$_PRIVATE_ECO	49
GER\$_DISPLAY_EXTENTS	51
Color	
GER\$_SET_COLORS	47

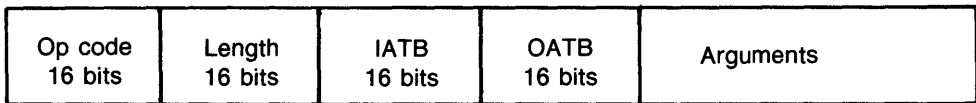
Table 15-1 (Cont.) Generic Encoding Symbols and Opcodes

Generic Encoding Symbol	Opcode
Color	
GER\$_SET_INTENSITIES	48
GER\$_CREATE_COLOR_MAP	50

Arguments

Figure 15-3 illustrates the format of an argument within a binary instruction that changes attribute settings.

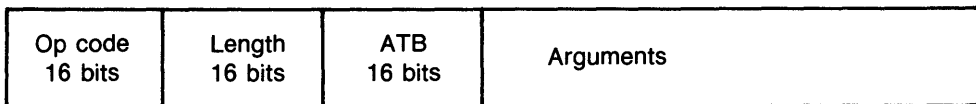
Figure 15-3 Format of Attribute-Related Argument



ZK-5474-86

Figure 15-4 illustrates the format of an argument within a binary encoded instruction that produces graphics or text.

Figure 15-4 Format of Graphics- and Text-Related Argument



ZK-5475-86

Table 15-2 lists the possible arguments that can appear in a binary encoded instruction.

Table 15-2 Arguments of Binary Encoded Instructions

Opcode	Argument ³	Data Type	Description
Attributes¹			
	iatb	word	Input attribute block for set operations
	oatb	word	Output attribute block for set operations
GER\$C_SET_ARC_TYPE	arc_type	word	arc type
GER\$C_SET_BACKGROUND_INDEX	background_index	word	Background index
GER\$C_SET_CHAR_ENCODING	char_encoding_type	word	Character encoding type
GER\$C_SET_CHAR_SIZE	char_size_flags	word	Scaling flags
	char_size_enable	bitfield mask	Font ideal size for x
	char_size_def_x	bitfield mask	Font ideal size for y
	char_size_def_y	bitfield mask	Widest char
	char_size_def_char	bitfield mask	
	char_size_example	word	Example character
	char_size_width	F_floating	Character width
	char_size_height	F_floating	Character height
GER\$C_SET_CHAR_SLANT	char_slant_angle	F_floating	Character slant angle
GER\$C_SET_CHAR_SPACING	char_space_dx	F_floating	Delta x spacing
	char_space_dy	F_floating	Delta y spacing
GER\$C_SET_CHAR_ROTATION	char_rotation_angle	F_floating	Character rotation angle
GER\$C_SET_CLIP	clip_flags	word	Clipping rectangle
	clip_x1	F_floating	
	clip_y1	F_floating	
	clip_x2	F_floating	
	clip_y2	F_floating	

¹ All attribute-related encoding items start with input attribute block (IATB) and output attribute block (OATB) numbers and then contain attribute specific information.

³ Arguments whose data type is word, longword, or character use the prefix GER\$W_, GER\$F_, or GER\$G, respectively, EXCEPT GER\$L_LINE_STYLE and GER\$L_IMAGE_SIZE. For example, GER\$W_IATB, GER\$F_CHAR_SIZE_WIDTH, or GER\$G_FONT_ID_STRING.

Table 15-2 (Cont.) Arguments of Binary Encoded Instructions

Opcode	Argument ³	Data Type	Description
Attributes¹			
GER\$C_SET_COLORS	color_count	word	Number of indices
	color_index	word	First index
	color_values	longword array	R, G, and B vectors
GER\$C_SET_FILL_PATTERN	fill_flags	word	Flags
	fill_index	word	Index
GER\$C_SET_FONT	font_id_length	word	Font name length
	font_id_string	character	Font name string
GER\$C_SET_INTENSITIES	intensity_count	word	Number of indices
	intensity_index	word	First index
	intensity_values	longword array	I vector
GER\$C_SET_LINE_STYLE	line_style	longword	32-bit bitvector
GER\$C_SET_LINE_WIDTH	line_width_nc	F_floating	Normalized coordinates
	line_width_dc	F_floating	Pixel coordinates
	line_width_mode	word	Width mode
GER\$C_SET_TEXT_FORMATTING	text_format_mode	word	Text formatting mode
GER\$C_SET_TEXT_MARGINS	text_margin_x	F_floating	Starting position
	text_margin_y	F_floating	
	text_margin_distance	F_floating	Ending position
GER\$C_SET_TEXT_PATH	text_path_major	word	Major path code
	text_path_minor	word	Minor path code
GER\$C_SET_TEXT_SLOPE	text_slope_angle	F_floating	Angle of text slope

¹All attribute-related encoding items start with input attribute block (IATB) and output attribute block (OATB) numbers and then contain attribute specific information.

³Arguments whose data type is word, longword, or character use the prefix GER\$W_, GER\$F_, or GER\$G, respectively, EXCEPT GER\$L_LINE_STYLE and GER\$L_IMAGE_SIZE. For example, GER\$W_IATB, GER\$F_CHAR_SIZE_WIDTH, or GER\$G_FONT_ID_STRING.

15-8 Metafiles and Private Data

Table 15-2 (Cont.) Arguments of Binary Encoded Instructions

Opcode	Argument ³	Data Type	Description
Attributes¹			
GER\$_SET_	writing_mode	word	Writing mode
WRITING_MODE			
GER\$_SET_	writing_index	word	Writing index
WRITING_INDEX			
Graphics and Text²			
	output_atb	word	ATB for graphics and text operations
GER\$_ELLIPSE	ellipse_x	F_floating	Center point
	ellipse_y	F_floating	
	ellipse_width	F_floating	Radius width and height
	ellipse_height	F_floating	
	ellipse_start_deg	F_floating	
GER\$_IMAGE	ellipse_end_deg	F_floating	
	image_x1	F_floating	Lower-left corner of raster image
	image_y1	F_floating	
	image_x2	F_floating	Upper-right corner of raster image
	image_y2	F_floating	
	image_width	word	Image width in pixels
	image_height	word	Image height in pixels
	image_bpp	word	Bits per pixel
image_size	longword	Number of bytes in image	

¹All attribute-related encoding items start with input attribute block (IATB) and output attribute block (OATB) numbers and then contain attribute specific information.

²All output-related encoding items start with an attribute block (ATB) number and then followed by graphics and text output information.

³Arguments whose data type is word, longword, or character use the prefix GER\$_W_, GER\$_F_, or GER\$_G, respectively, EXCEPT GER\$_L_LINE_STYLE and GER\$_L_IMAGE_SIZE. For example, GER\$_W_IATB, GER\$_F_CHAR_SIZE_WIDTH, or GER\$_G_FONT_ID_STRING.

Table 15-2 (Cont.) Arguments of Binary Encoded Instructions

Opcode	Argument³	Data Type	Description
Graphics and Text²			
	image_data	byte array	Place to store actual data
GER\$_PLOT	plot_count	word	Number of points
	plot_data	longword array	Points
GER\$_TEXT	text_encoding	word	8- or 16-bit encoding
	text_length	word	Text length in bytes
	text_data	character	Text string
GER\$_LINE	line_count	word	Number of points
	line_data	longword array	Points
Color Map			
GER\$_CREATE_ COLOR_MAP	color_map_attributes	longword	Color map attributes
	color_map_resident	bitfield mask	
	color_map_no_bind	bitfield mask	
	color_map_share	bitfield mask	
	color_map_system	bitfield mask	
	color_map_name_size	word	
	color_map_size	word	
	color_map_name	character	Virtual color map name
Private Data			
GER\$_PRIVATE	private_facnum	word	Facility number
	private_length	word	Length of data
	private_data	byte array	Data
Metafile			
GER\$_VERSION	version_major	word	Encoding version number

²All output-related encoding items start with an attribute block (ATB) number and then followed by graphics and text output information.

³Arguments whose data type is word, longword, or character use the prefix GER\$_, GER\$_, or GER\$_, respectively, EXCEPT GER\$__LINE_STYLE and GER\$__IMAGE_SIZE. For example, GER\$__IATB, GER\$__CHAR_SIZE_WIDTH, or GER\$__FONT_ID_STRING.

15-10 Metafiles and Private Data

Table 15-2 (Cont.) Arguments of Binary Encoded Instructions

Opcode	Argument ³	Data Type	Description
Metafile			
	version_minor	word	
	version_eco	word	
GER\$_	identification_length	word	
IDENTIFICATION	identification_string	character	
GER\$_DATE	date_length	word	File creation date
	date_string	character	
GER\$_PRIVATE_ ECO	private_eco_facnum	word	
	private_eco_major	word	
	private_eco_minor	word	
	private_eco_eco	word	
Miscellaneous			
GER\$_DISPLAY_ EXTENTS	extent_minx	F_floating	Extent rectangle
	extent_miny	F_floating	
	extent_maxx	F_floating	
	extent_maxy	F_floating	
GER\$_SET_ POSITION	text_pos_x	F_floating	Text position
	text_pos_y	F_floating	
GER\$_ALIGN_ POSITION	align_pos_atb	word	Attribute block
	align_pos_x	F_floating	
	align_pos_y	F_floating	
GER\$_BEGIN_ DISPLAY	display_wc_minx	f_floating	Dimensions of virtual display
	display_wc_miny	f_floating	
	display_wc_maxx	f_floating	
	display_wc_maxy	f_floating	
	display_width	f_floating	

³Arguments whose data type is word, longword, or character use the prefix GER\$_W_, GER\$_F_, or GER\$_G_, respectively, EXCEPT GER\$_L_LINE_STYLE and GER\$_L_IMAGE_SIZE. For example, GER\$_W_IATB, GER\$_F_CHAR_SIZE_WIDTH, or GER\$_G_FONT_ID_STRING.

Table 15-2 (Cont.) Arguments of Binary Encoded Instructions

Opcode	Argument ³	Data Type	Description
Miscellaneous			
	display_height	f_floating	
GER\$C_END_ DISPLAY	No arguments		

³Arguments whose data type is word, longword, or character use the prefix GER\$W_, GER\$F_, or GER\$G, respectively, EXCEPT GER\$L_LINE_STYLE and GER\$L_IMAGE_SIZE. For example, GER\$W_IATB, GER\$F_CHAR_SIZE_WIDTH, or GER\$G_FONT_ID_STRING.

15.2.2 Creating UIS Metafiles

UIS metafiles are encoded binary instructions which when extracted from a display list with UIS\$EXTRACT_OBJECT or UIS\$EXTRACT_REGION are *generically encoded*. UIS metafiles consist of the following parts: (1) header information, (2) generically encoded binary instructions, and (3) a trailer. The header and trailer are special binary instructions that indicate the beginning and end of a UIS metafile. The *generic encoding* of UIS metafiles allows you to store the extracted contents of the display list in a buffer or file. Table 15-3 lists the parts of a UIS metafile.

Table 15-3 Structure of UIS Metafiles

Generic Encoding Symbol	Function
Header Information	
GER\$C_VERSION	Level of generic encoding syntax. The version always appears first.
GER\$C_IDENTIFICATION	User-specified optional identification string.
GER\$C_DATE	Optional and user-specified.
GER\$C_PRIVATE_ECO ^{1,2}	Optional and user-specified.
GER\$C_CREATE_COLOR_MAP	Used by UIS\$EXECUTE_DISPLAY.
GER\$C_SET_COLORS	Used by UIS\$EXECUTE_DISPLAY.
GER\$C_BEGIN_DISPLAY	Dimensions of the virtual display to be created by UIS\$EXECUTE_DISPLAY.

¹Engineering Change Order

²See Table 15-1 for the generic symbols in each of these categories of binary encoded instructions.

15-12 Metafiles and Private Data

Table 15-3 (Cont.) Structure of UIS Metafiles

Generic Encoding Symbol	Function
Encoded Binary Instructions²	
GER\$C_DISPLAY_EXTENTS ³	Bounds of an extent rectangle used in UIS\$EXTRACT_REGION.
Segment	Express the hierarchical structure within a display list and identify the attributes associated with a segment.
Attribute	Allow the modification of any attribute in any attribute block. A generic encoding opcode exists for each attribute.
Graphics and text	Contain the data necessary to draw graphic objects.
Application-specific	Associate data with a user-specified facility.
Trailer	
GER\$C_END_DISPLAY	Ends the UIS metafile.

²See Table 15-1 for the generic symbols in each of these categories of binary encoded instructions.

³Generated by UIS\$EXTRACT_REGION only

15.2.3 Structure of a UIS Metafile

A UIS metafile consists of three parts—header information, binary instructions, and trailer information. Figure 15-5 illustrates the structure of a UIS metafile containing a single extracted graphic object. Note that attribute modification instructions precede the object and private data instructions follow it. Also, if the extracted object lay previously within a segment, segmentation instructions must surround it in the metafile.

Figure 15-5 Structure of UIS Metafile

Header Information	GER\$C_VERSION	Length	Arguments		
	GER\$C_IDENTIFICATION	Length	Arguments		
	GER\$C_DATE	Length	Arguments		
	GER\$C_BEGIN_DISPLAY	Length	Arguments		
Beginning Segmentation Instruction	GER\$C_BEGIN	Length	No arguments		
Attribute Modification Instructions	GER\$C_SET_FONT	Length	IATB	OATB	Arguments
	GER\$C_SET_FILL_PATTERN	Length	IATB	OATB	Arguments
Extracted Graphic Object	GER\$C_ELLIPSE	Length	ATB	Arguments	
Private Data	GER\$C_PRIVATE	Length	Arguments		
	GER\$C_PRIVATE	Length	Arguments		
Ending Segmentation Instruction	GER\$C_END	Length	No Arguments		
Trailer Information	GER\$C_END_DISPLAY	Length	No arguments		

ZK 5476 86

Private data is discussed later in this chapter.

15-14 Metafiles and Private Data

15.2.4 Programming Options

The ability to create UIS metafiles allows you to save display screen output in files for reexecution at a later time.

Creating UIS Metafiles

You can extract an object or the contents of a region within a virtual display using `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_OBJECT` or `UIS$EXTRACT_REGION`, `UIS$EXTRACT_TRAILER` and store the data in a buffer or file as a metafile using the following procedure:

1. Determine the size of the buffer needed to store the header information, binary encoded stream, and trailer using `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_OBJECT` or `UIS$EXTRACT_REGION`, and `UIS$EXTRACT_TRAILER` omitting the buffer length and buffer address parameters.
2. Call `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_OBJECT` or `UIS$EXTRACT_REGION`, and `UIS$EXTRACT_TRAILER` again, specifying the previously omitted parameters to extract the header information, binary encoded instructions, and trailer and to store the data in three buffers.
3. Use the VAX FORTRAN `OPEN` and `WRITE` statements to write the contents of the buffers to an external file.

Executing the Metafile

UIS metafiles extracted and stored in a buffer can be written to the same virtual display using `UIS$EXECUTE`.

`UIS$EXECUTE_DISPLAY` creates a new virtual display and executes the metafile in the new display space. However, you must call `UIS$CREATE_WINDOW` to view the graphic object in the virtual display.

15.2.5 Program Development I

Programming Objective

To extract the contents of a region in the virtual display and create a UIS metafile.

Programming Tasks

1. Initialize variables.
2. Create a virtual display.
3. Draw graphic objects in the virtual display.
4. Create a display window and viewport.
5. Determine the size of each part of the metafile.

6. Allocate the space in buffers for each part of the metafile.
7. Extract the contents of the specified region in a buffer.
8. Write the contents of the buffer to an external file.

```
PROGRAM EXTRACT
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
DATA RETLEN1,RETLEN2,RETLEN3/3*0/

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,30.0,30.0,20.0,20.0)
```

c Draw some objects

```
CALL UIS$PLOT(VD_ID,0,7.0,10.0,16.0,10.0,7.0,15.0,
2      7.0,10.0) ①
CALL UIS$ELLIPSE(VD_ID,0,20.0,20.0,9.0,5.0) ②
CALL UIS$TEXT(VD_ID,0,'Haste and wisdom are things far odd',
2      11.0,15.0) ③
```

c Create a display window

```
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
PAUSE
```

c Find out how much space to allocate for each part of the metafile

```
CALL UIS$EXTRACT_HEADER(VD_ID,,RETLEN1) ④
CALL UIS$EXTRACT_REGION(VD_ID,,,,,RETLEN2) ⑤
CALL UIS$EXTRACT_TRAILER(VD_ID,,RETLEN3) ⑥
```

c Virtual memory is allocated for the buffers

```
STATUS=LIB$GET_VM(RETLEN1,ENCODED1) ⑦
IF (.NOT.STATUS) CALL LIB$STOP(%VAL(STATUS)) ⑧
STATUS=LIB$GET_VM(RETLEN2,ENCODED2) ⑨
IF (.NOT.STATUS) CALL LIB$STOP(%VAL(STATUS)) ⑩
STATUS=LIB$GET_VM(RETLEN3,ENCODED3) ⑪
IF (.NOT.STATUS) CALL LIB$STOP(%VAL(STATUS)) ⑫

RETLEN=RETLEN1+RETLEN2+RETLEN3

TYPE *,'HEADER DATA',RETLEN1,' BYTES' ⑬
TYPE *,'BINARY INSTRUCTION',RETLEN2,' BYTES' ⑭
TYPE *,'TRAILING DATA',RETLEN3,' BYTES' ⑮

TYPE *,'NO. OF BYTES ALLOCATED = ',RETLEN ⑯

PAUSE
```

C Extract the data and store it in a buffer

```
CALL UIS$EXTRACT_HEADER(VD_ID,RETLEN1,%VAL(ENCODED1)) ⑰
CALL UIS$EXTRACT_REGION(VD_ID,,,,RETLEN2,%VAL(ENCODED2)) ⑱
CALL UIS$EXTRACT_TRAILER(VD_ID,RETLEN3,%VAL(ENCODED3)) ⑲
```

15-16 Metafiles and Private Data

```
c Write the contents of the buffer to an external file
  OPEN(UNIT=10,FILE='$DISK:[MY_DIR]METAFILE.DAT',STATUS='NEW') 20

c Call subroutine to write the contents of the buffer
  CALL BUFFERWRITE(%VAL(ENCODED1),RETLEN1,10) 21
  CALL BUFFERWRITE(%VAL(ENCODED2),RETLEN2,10) 22
  CALL BUFFERWRITE(%VAL(ENCODED3),RETLEN3,10) 23

c Close the external file
  CLOSE(UNIT=10,STATUS='SAVE') 24

  END

  SUBROUTINE BUFFERWRITE(BUFFER,LENGTH,LUN) 25
  IMPLICIT INTEGER(A-Z)
  BYTE BUFFER(LENGTH)

  WRITE(LUN,500)BUFFER 26
500  FORMAT(T3,I7)

  RETURN
  END
```

Calls to `UIS$PLOT`, `UIS$ELLIPSE`, and `UIS$TEXT` ① ② ③ draw objects in the virtual display.

Next, you must find out how much space must be allocated for the buffers that will hold the header data, binary encoded stream, and trailing data. ④ ⑤ ⑥. The variables `retlen1`, `retlen2`, and `retlen3` receive the length of the header data, binary encoded stream, and trailing data.

Virtual memory is allocated for the buffers and the address of each buffer is stored in the pointers `encoded1`, `encoded2`, and `encoded3` using `LIB$GET_VM`. ⑦ ⑧ ⑨. A test for completion status of each Run-Time Library call ⑩ ⑪ ⑫ is performed.

The length of the header data, encoded stream, and trailing data are typed in the emulation window ⑬ ⑭ ⑮ as well as the total number of bytes allocated ⑯.

The contents of the display list are extracted using `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_REGION`, and `UIS$EXTRACT_TRAILER` stored at the location indicated by pointers `encoded1`, `encoded2`, and `encoded3` ⑰ ⑱ ⑲. Using the VAX FORTRAN built-in function `%VAL`, the pointers `encoded1`, `encoded2`, and `encoded3` are evaluated in terms of the actual data they store—the addresses of the starting point of each buffer.

An external file is opened with the VAX FORTRAN `OPEN` statement for program output 20.

The pointer `encoded` was implicitly declared as a longword integer. Therefore, you cannot simply write the data to the file `PRIVATE.DAT`.

The subroutine BUFFERWRITE is called ⑳ ㉑ ㉒ three times to perform this task. Three arguments are passed in the call ㉓—buffer address, buffer size, and the VAX FORTRAN logical unit number of the output device. An array BUFFER is constructed from this data.

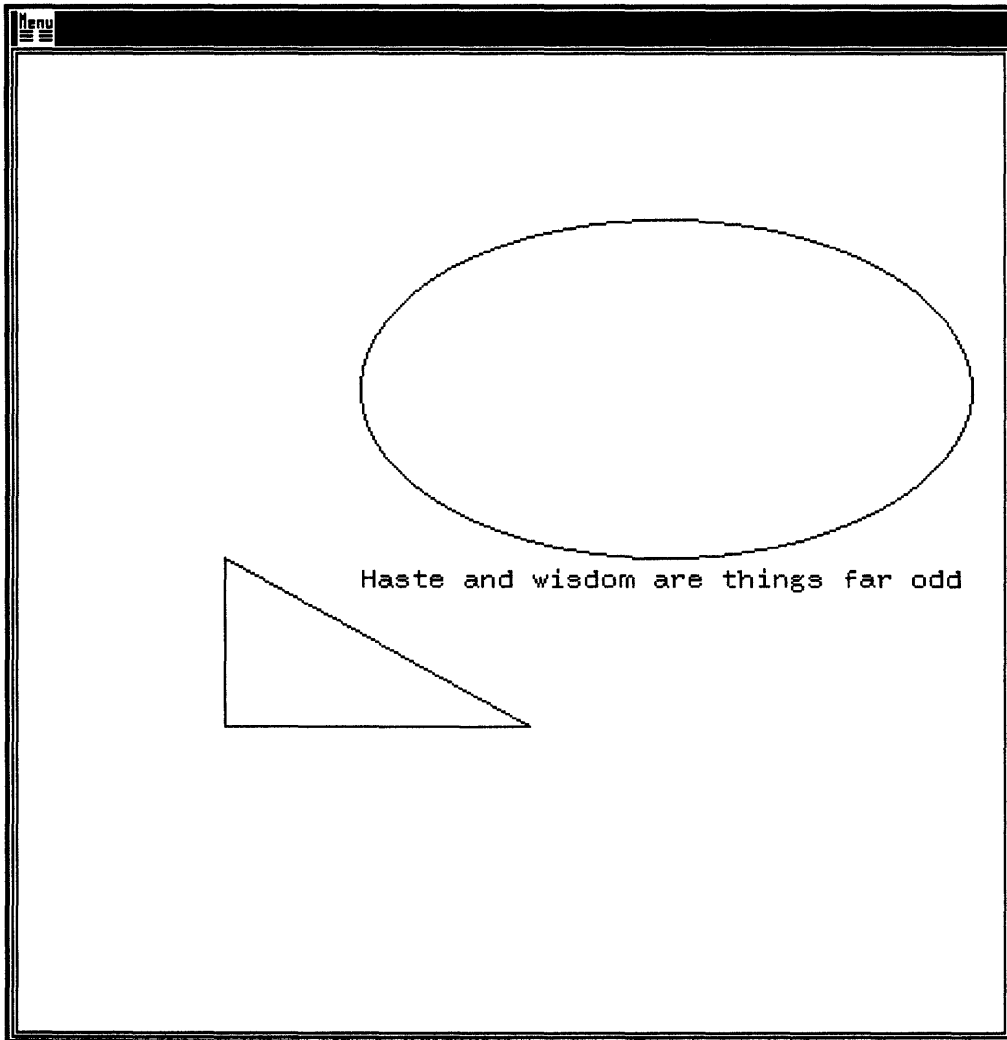
The subroutine BUFFERWRITE writes the contents of BUFFER to the UIS metafile PRIVATE.DAT ㉔. First the header data is stored in the metafile, then the binary encoded stream, and, finally, the trailing data is written to PRIVATE.DAT.

Prior to program termination, the VAX FORTRAN CLOSE statement closes the file ㉕.

15.2.5.1 Calling UIS\$EXTRACT_HEADER, UIS\$EXTRACT_REGION, and UIS\$EXTRACT_TRAILER

A triangle, an ellipse, and text are drawn in a virtual display as shown in Figure 15-6.

Figure 15-6 Original Objects Drawn in the Virtual Display



The terminal emulation window shown in Figure 15-7 shows buffer size information for metafile components.

Figure 15-7 After Buffer Execution

```

$ run extract
FORTRAN PAUSE
$ cont
HEADER DATA          101  BYTES
BINARY INSTRUCTION          151  BYTES
TRAILING DATA           4  BYTES
TOTAL NO. OF BYTES ALLOCATED =          256
FORTRAN PAUSE
$ █

```

ZK 5265 86

15.3 Display Lists and Private Data

As mentioned earlier, display lists are created when graphics routines are executed. Application-specific or *private data* can be bound to graphic objects. The binary encoded instructions contained in the display list points to internal buffers that contain the private data.

15.3.1 Using Private Data

Private data is used to include some application-specific information with the graphic objects displayed on the workstation screen. The nature of this information is entirely at the discretion of the user. For example, an application that draws a vertical bar graph and plots relative humidity over a 24-hour period could create data on an hourly basis. The private data, in this case, indicating temperature or wind speed could be associated with each vertical bar. Private data is not displayed on the workstation screen and is not available to users unless extracted into a buffer or metafile and executed. Private data can be attached to any graphic object drawn in the virtual display.

15-20 Metafiles and Private Data

15.3.2 Programming Options

We will construct a program that reads data from an external file and uses it as private data.

Creating Private Data

You can create private data with `UIS$PRIVATE`.

Extracting Private Data

You can extract private data and store it in a buffer using `UIS$EXTRACT_PRIVATE` using the following procedure:

1. Determine the size of the buffer needed to store the header information, binary encoded stream, and the trailer using `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_PRIVATE`, and `UIS$EXTRACT_TRAILER` omitting the buffer length and buffer address parameters.
2. Call `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_PRIVATE` and `UIS$EXTRACT_TRAILER` again specifying the previously omitted parameters to extract the private data and store the data in a buffer.
3. Use the VAX FORTRAN OPEN statement to write the contents of the buffer to an external file.

Deleting Private Data

You can delete private data associated with a graphic object using `UIS$DELETE_PRIVATE`.

15.3.3 Program Development II

Programming Objectives

1. To append private data to an object in the display list.
2. To extract the private data.
3. To create a UIS metafile containing the private data instruction.

Programming Tasks

1. Declare an array to receive the private data from an external file.
2. Type out the contents of the array to verify it.
3. Create private data and append it to the last object in the display list.
4. Determine how large the buffers must be.
5. Allocate memory for the buffers.

6. Extract the private data.
7. Write the contents of the buffers to an external file.

Please note that in order to run this program, you should modify the file specifications in the OPEN statements and construct a data file similar to DATA.DAT.

```

PROGRAM PRIVATE
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
BYTE PRIV(1:23) ①

c Construct a descriptor
INTEGER*4 PRIV_DESC(2) ②
PRIV_DESC(1)=23
PRIV_DESC(2)=%LOC(PRIV) ③

c Open external file containing private data
OPEN(UNIT=8,FILE='$DISK:[MY_DIR]DATA.DAT',STATUS='OLD') ④

c Read data into array
READ(8,50)PRIV ⑤
50 FORMAT(A7)

CLOSE(UNIT=8,STATUS='SAVE')

VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,30.0,30.0,15.0,15.0) ⑥

c draw the hot air balloon
CALL UIS$SET_FONT(VD_ID,0,2,'MY_FONT_5')
INDEX=87
CALL UIS$SET_FILL_PATTERN(VD_ID,2,2,INDEX)

CALL UIS$CIRCLE(VD_ID,2,12.0,20.0,8.0)
CALL UIS$LINE(VD_ID,2,10.0,12.0,10.0,8.0,14.0,12.0,14.0,8.0,
2 10.0,10.0,14.0,10.0,10.0,8.0,14.0,8.0)

c draw house
CALL UIS$PLOT(VD_ID,0,15.0,8.0,29.0,8.0,22.0,13.0,
2 15.0,8.0)
CALL UIS$LINE(VD_ID,0,15.0,8.0,15.0,0.0,29.0,8.0,29.0,0.0)

c draw door
CALL UIS$PLOT(VD_ID,0,21.0,0.0,21.0,4.0,23.0,4.0,23.0,0.0)

C create windows
CALL UIS$PLOT(VD_ID,0,17.0,2.0,17.0,6.0,19.0,6.0,19.0,2.0,
2 17.0,2.0)
CALL UIS$LINE(VD_ID,0,17.0,4.0,19.0,4.0,18.0,2.0,18.0,6.0)

CALL UIS$PLOT(VD_ID,0,25.0,2.0,25.0,6.0,27.0,6.0,27.0,2.0,
2 25.0,2.0)
CALL UIS$LINE(VD_ID,0,25.0,4.0,27.0,4.0,26.0,2.0,26.0,6.0)

```

15-22 Metafiles and Private Data

c create chimney

```
CALL UIS$LINE(VD_ID,0,26.0,11.0,28.0,11.0,26.0,11.0,26.0,10.0,  
2      28.0,11.0,28.0,9.0)
```

c create smoke

```
CALL UIS$ELLIPSE(VD_ID,0,27.0,13.0,2.5,1.0)  
CALL UIS$ELLIPSE(VD_ID,0,27.25,16.0,2.25,1.0)  
CALL UIS$ELLIPSE(VD_ID,0,27.5,19.0,2.0,1.0)  
CALL UIS$ELLIPSE(VD_ID,0,27.75,22.0,1.75,1.0)  
CALL UIS$ELLIPSE(VD_ID,0,28.0,25.0,1.5,1.0)  
CALL UIS$ELLIPSE(VD_ID,0,28.25,28.0,1.25,1.0)  
CURR_ID=UIS$GET_CURRENT_OBJECT(VD_ID) ⑦
```

c type out buffer containing private data

```
TYPE *,PRIV ⑧
```

c Create private data

```
FACNUM = 1  
CALL UIS$PRIVATE(vd_id,FACNUM,PRIV_DESC) ⑨  
  
CALL UIS$SET_LINE_WIDTH(VD_ID,0,3,15.0)  
CALL UIS$PLOT(VD_ID,3,1.0,29.0,4.0,11.0)  
  
CALL UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')  
  
PAUSE
```

c Determine size of buffer

```
CALL UIS$EXTRACT_HEADER(VD_ID,,RETLN1) ⑩  
CALL UIS$EXTRACT_PRIVATE(CURR_ID,,RETLN2) ⑪  
CALL UIS$EXTRACT_TRAILER(VD_ID,,RETLN3) ⑫  
  
RETLN=RETLN1+RETLN2+RETLN3  
  
TYPE *,'BUFFER SIZE FOR HEADER INFO',RETLN1,'BYTES' ⑬  
TYPE *,'BUFFER SIZE REQUIRED',RETLN2,'BYTES' ⑭  
TYPE *,'BUFFER SIZE FOR TRAILING INFO',RETLN3,'BYTES' ⑮
```

C Allocate the virtual memory for the buffer

```
STATUS=LIB$GET_VM(RETLN1,EXT_PRIV1) ⑯  
IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS)) ⑰  
STATUS=LIB$GET_VM(RETLN2,EXT_PRIV2) ⑱  
IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS)) ⑲  
STATUS=LIB$GET_VM(RETLN3,EXT_PRIV3) ⑳  
IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS)) ㉑
```

c Extract and store private data in buffer

```
CALL UIS$EXTRACT_HEADER(VD_ID,RETLN1,%VAL(EXT_PRIV1)) ㉒  
CALL UIS$EXTRACT_PRIVATE(CURR_ID,RETLN2,%VAL(EXT_PRIV2)) ㉓  
CALL UIS$EXTRACT_TRAILER(VD_ID,RETLN3,%VAL(EXT_PRIV3)) ㉔
```

```

        CALL BUFFERTYPE(%VAL(EXT_PRIV2),RETLEN2) 25
C Open an external file
  OPEN(UNIT=11,FILE='$DISK:[MY_DIR]PRIVATE.DAT',STATUS='NEW', 26
    2      FORM='UNFORMATTED')

c Write the contents of the buffer
  CALL BUFFERWRITE(%VAL(EXT_PRIV1),RETLEN1,11) 27
  CALL BUFFERWRITE(%VAL(EXT_PRIV2),RETLEN2,11) 28
  CALL BUFFERWRITE(%VAL(EXT_PRIV3),RETLEN3,11) 29

C Close the file
  CLOSE(UNIT=11,STATUS='SAVE')

  PAUSE

  END

  SUBROUTINE BUFFERWRITE(BUFFER,LENGTH,LUN) 30
  IMPLICIT INTEGER(A-Z)
  BYTE BUFFER(LENGTH)

  WRITE(LUN,500)BUFFER 31
500  FORMAT(T3,I7)

  RETURN
  END

  SUBROUTINE BUFFERTYPE(BUFFER,length) 32
  IMPLICIT INTEGER(A-Z)
  BYTE BUFFER(length)

  TYPE *,buffer 33

  RETURN
  END
    
```

A data file DATA.DAT of private data is constructed. It consists of a sentence. Because each character requires a byte of storage, the total number of characters in the data file is specified as the upper bound of array PRIV ① as well as the buffer length in the descriptor you must construct for UIS\$PRIVATE ②.

An external file DATA.DAT is opened ④ and read into the array PRIV ⑤.

A circle, a triangle, and text are drawn in the virtual display ⑥.

UIS\$GET_CURRENT_OBJECT retrieves the identifier of the last object drawn in the virtual display ⑦.

The array PRIV is typed out to verify its contents ⑧.

UIS\$PRIVATE associates the sentence contained in the array PRIV with the objects drawn in the virtual display ⑨. Note that the location of the array PRIV is passed by descriptor ③.

15-24 Metafiles and Private Data

Suppose you want to extract the data and store it in a buffer as a UIS metafile. You must first determine how much space the header data, binary encoded private data, and trailing data will occupy by calling `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_PRIVATE`, and `UIS$EXTRACT_TRAILER` without specifying the `buflen` and `bufaddr` arguments ⑩ ⑪ ⑫.

The variables `retlen1`, `retlen2`, and `retlen3` are typed out to reveal the size of each part of the display list ⑬ ⑭ ⑮.

A call to `LIB$GET_VM` allocates virtual memory for three buffers using the value of `retlen1`, `retlen2`, and `retlen3` and stores the location of each buffer in the pointers `ext_priv1`, `ext_priv2`, and `ext_priv3`, respectively ⑯ ⑰ ⑱. A test for completion status is performed for each Run-Time Library call ⑲ ⑳ ㉑.

If you did not use `LIB$GET_VM`, you would have to explicitly declare an array with an actual length in the beginning of the program. However, at that point in the program, you would have no idea how large such an array would need to be.

A call to `UIS$EXTRACT_HEADER`, `UIS$EXTRACT_PRIVATE`, and `UIS$EXTRACT_TRAILER`, specifying the omitted parameters, extracts the header data, binary encoded private data, and the trailing data and stores them in separate buffers ㉒ ㉓ ㉔. Because `ext_priv1`, `ext_priv2`, and `ext_priv3` are pointers, you must obtain the actual data that they store using the VAX FORTRAN built-in function `%VAL`.

Suppose you want to look at the contents of the user buffer before you write the contents to an external file.

Because the pointer `ext_priv` was implicitly declared a longword integer and functions as a pointer, we cannot simply type the data in the user.

A subroutine `BUFFERTYPE` is called referencing the pointer `ext_priv2` and the size of the buffer ㉕. Two arguments are passed in the call—the pointer name and the size of the buffer. The subroutine `BUFFERTYPE` reads the data from the location to which `ext_priv2` points ㉖ and writes the data in terminal emulation window ㉗.

The file `PRIVATE.DAT` is opened ㉘.

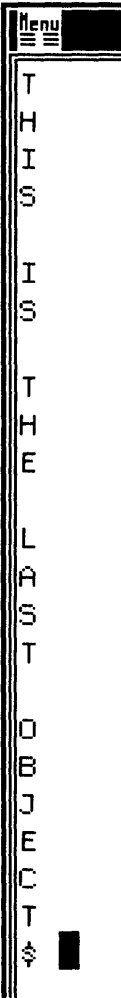
The subroutine `BUFFERWRITE` ㉙ is called three times to write the header, private, and trailer data to the external file ㉚ ㉛ ㉜. Three arguments are passed in the call—buffer address, buffer size, and the VAX FORTRAN logical unit number of the output device. An array `BUFFER` is declared from this data and an association with an external file is established.

The subroutine `BUFFERWRITE` writes the contents of `BUFFER` to the file `PRIVATE.DAT` ㉝. The file is closed and saved.

15.3.3.1 Calling UIS\$PRIVATE and UIS\$EXTRACT_PRIVATE

Figure 15-8 shows the sample containing character string private data in the external file DATA.DAT

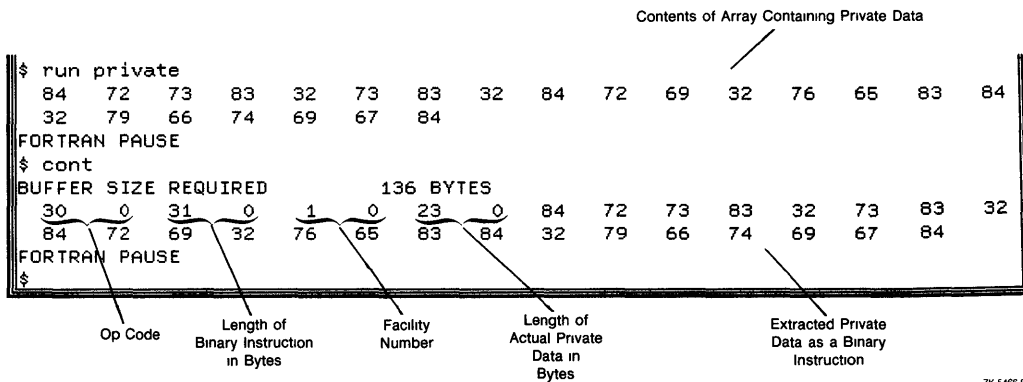
Figure 15-8 Private Data



15-26 Metafiles and Private Data

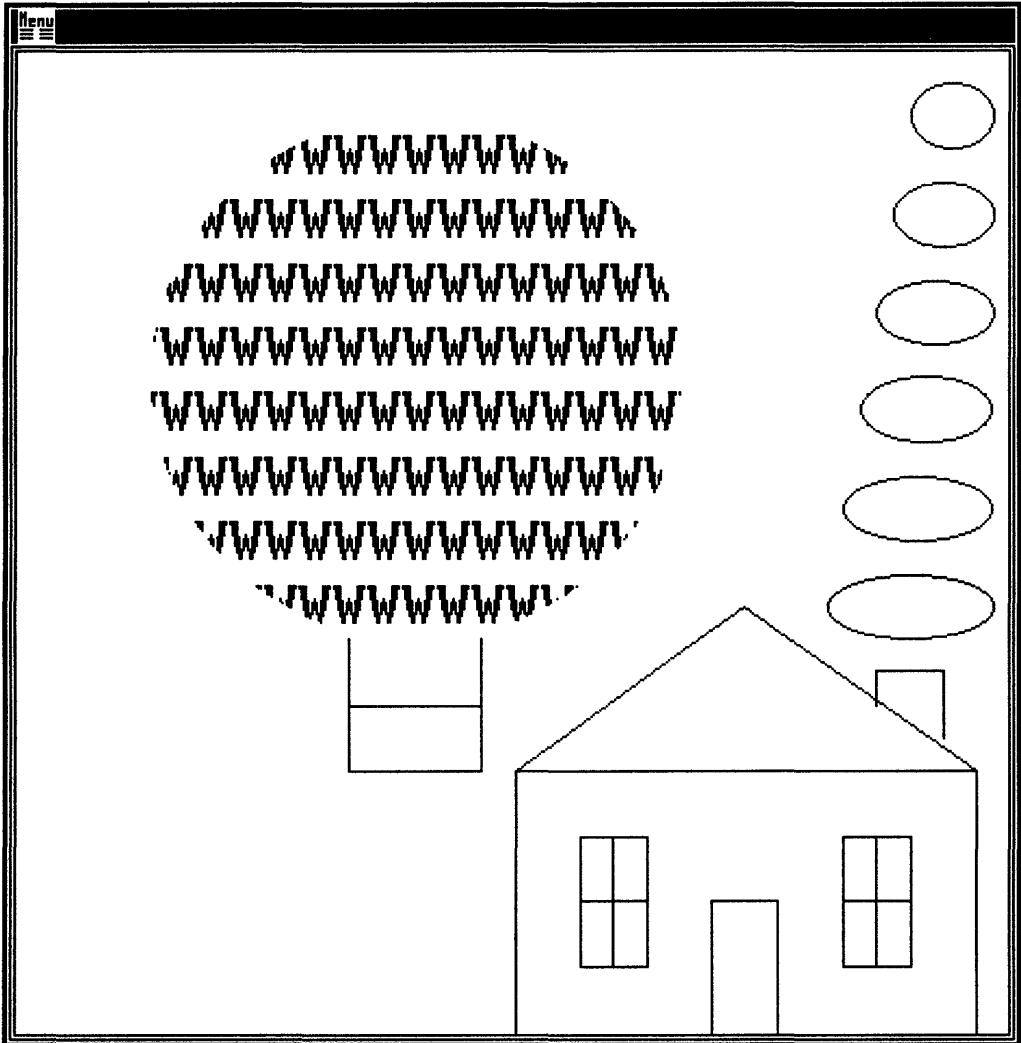
Figure 15-9 shows the contents of the array PRIV read from the external file DATA.DAT. Note that each number is an ASCII code. The required buffer size is also shown. In addition, the extracted generically encoded binary private data instruction is shown as metafile opcodes and ASCII codes.

Figure 15-9 Verifying the Contents of the Temporary Array and User Buffer



The private data was appended to the last ellipse drawn—the smallest cloud of a smoke rising from the chimney shown in Figure 15-10.

Figure 15-10 Hot Air Balloon



Chapter 16

Programming in Color

16.1 Overview

Until now we have assumed that the one way to change the appearance of graphic objects and text is through modification of attribute settings in attribute block 0. However, depending on the VAXstation color system you have, you can draw graphic objects in over 16 million colors. This chapter discusses the following topics:

- Using color and intensity routines
- Setting entries in virtual color maps
- Creating shareable color maps
- Using color map segments
- Using color and intensity inquiry routines

This chapter is meaningful for VAXstation users programming in either an intensity or color environment.

16.2 Color and Intensity Routines—How to Use Them

Color and intensity routines allow your application to draw graphic objects in either color or shades of gray. These routines create and load the structures known as virtual color maps and color map segments that hold the color values that your application use. Such routines perform the following tasks:

- Create and delete virtual color maps
- Load virtual color map entries with color values
- Create and delete color map segments
- Load entries in color map segments

We will discuss color map segments later in this chapter.

16-2 Programming in Color

16.2.1 Step 1—Creating a Virtual Color Map

Whether you are programming in a color or an intensity environment, you must create a virtual color map using `UIS$CREATE_COLOR_MAP`. The virtual color map is a storage location similar to an artist's palette. Within the color map, you can store color values in locations known as entries. The virtual color map can vary according to the needs of your application. You can specify the *attributes* of the virtual color map as you see fit.

16.2.2 Step 2—Setting Virtual Color Map Attributes

The attributes specified for a virtual color map are either required or optional. You **must** specify the size of the virtual color map, that is, how many color map values it will hold. You can also specify optionally a name for the virtual color map. Other optional attributes are access and residency.

Virtual Color Map Size

As with any storage location, size is a consideration. For every color your application uses, you will need an entry in the virtual color map. You can specify a maximum size of 32,768 entries.

Access to Virtual Color Maps

Another important consideration involves who should have access to your virtual color map. What processes should you allow to have access to your virtual color map? Virtual color maps can be either *private* or *shareable*. If you specify that the virtual color map is private, no other processes have access to it. You can designate a virtual color map shareable for a certain group of users or shareable among all users.

Virtual Color Map Residency

Another attribute that you can specify explicitly, is *residency*. For application-specific reasons, you may wish to dedicate the color resources to the execution of your application. Since this precludes sharing the hardware color resources among applications, you should use this feature carefully.

16.2.3 Step 3—Setting Entries in the Virtual Color Map

At this point, depending on your color environment, your application must load color values into the color map entries using `UIS$SET_COLOR`, `UIS$SET_COLORS`, `UIS$SET_INTENSITIES`, or `UIS$SET_INTENSITY`.

Color and intensity values are expressed as floating-point number between *0.0* and *1.0*, inclusive. The color subsystem uses the red green blue (RGB) color model. The colors that result from the use of color values which denote percentages of red, green, and blue are sometimes not readily apparent from the value chosen. Therefore, it is recommended that you use color setup menus of the human interface to determine the appropriate RGB color component values. You can use these menus as you write your application.

Setting Single Entries

If your application uses only a few colors or intensities, you may require a small virtual color map. In such a case, you could load each color map entry using `UIS$SET_COLOR` or `UIS$SET_INTENSITY` each time.

Setting Multiple Entries

If, on the other hand, your virtual color map is large, you can arrange your color map values in an array using a single call to `UIS$SET_COLORS` or `UIS$SET_INTENSITIES`.

16.2.4 Programming Options

Whenever your application requires a range of color or intensities, you will need to use several of the UIS routines listed in Table 16-1.

Table 16-1 Color and Intensity Routines

Routine	Function
Virtual Color Maps	
<code>UIS\$CREATE_COLOR_MAP</code>	Creates a virtual color map
<code>UIS\$DELETE_COLOR_MAP</code>	Deletes a virtual color map
Loading Virtual Color Map Entries	
<code>UIS\$SET_COLOR</code>	Sets a single RGB color value in a virtual color map
<code>UIS\$SET_COLORS</code>	Sets multiple RGB color values in a virtual color map
<code>UIS\$SET_INTENSITIES</code>	Sets a single intensity value in a virtual color map
<code>UIS\$SET_INTENSITY</code>	Sets multiple RGB color values in a virtual color map

16-4 Programming in Color

Table 16-1 (Cont.) Color and Intensity Routines

Routine	Function
Color Map Segments	
UIS\$CREATE_COLOR_MAP_SEG	Creates a color map segment
UIS\$DELETE_COLOR_MAP_SEG	Deletes a color map segment

16.2.5 Program Development I

Programming Objective

To create and load a color map with single entries.

Programming Tasks

1. Establish a size for the virtual color map.
2. Create the virtual color map.
3. Create a virtual display.
4. Create a display window and viewport.
5. Load a single color map entry with one color value using UIS\$SET_COLOR.

```
PROGRAM SINGLE_ENTRY
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL J,K
DATA J/17.0/           ①
DATA K/16/            ②
DATA VCM_SIZE/8/

VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE)    ③
VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,40.0,40.0,15.0,15.0,VCM_ID) ④
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','WINDOW #1')

CALL UIS$SET_COLOR(VD_ID,0.0,0.40,0.30,0.0) ⑤
CALL UIS$SET_COLOR(VD_ID,1.0,0.5,0.5,0.5) ⑥
CALL UIS$SET_COLOR(VD_ID,2.0,0.5,0.25,0.5) ⑦
CALL UIS$SET_COLOR(VD_ID,3.0,0.0,0.7,0.3) ⑧
CALL UIS$SET_COLOR(VD_ID,4.0,0.25,0.25,0.9) ⑨
CALL UIS$SET_COLOR(VD_ID,5.0,0.90,0.5,0.0) ⑩
CALL UIS$SET_COLOR(VD_ID,6.0,0.80,0.30,0.0) ⑪
CALL UIS$SET_COLOR(VD_ID,7.0,0.35,0.65,0.95) ⑫
```

```

CALL UIS$SET_WRITING_INDEX(VD_ID,0,9,2) ⑬
CALL UIS$SET_WRITING_INDEX(VD_ID,0,10,3) ⑭
CALL UIS$SET_WRITING_INDEX(VD_ID,0,11,4) ⑮
CALL UIS$SET_WRITING_INDEX(VD_ID,0,12,5) ⑯
CALL UIS$SET_WRITING_INDEX(VD_ID,0,13,6) ⑰

DO I=9,13,1
CALL UIS$CIRCLE(VD_ID,I,J,20.0,10.0) ⑱
J=J+2.0
ENDDO

PAUSE

DO I=9,13
CALL UIS$CIRCLE(VD_ID,I,21.0,K,10.0) ⑲
K=K+2.0
ENDDO

PAUSE

END

```

The counters *j* and *k* are declared and initialized ① ②.

An eight-entry virtual color map is created with no attributes specified ③.

The virtual color map is associated with the virtual display in UIS\$CREATE_DISPLAY ④ during creation of the virtual display.

Each color value is loaded into a virtual color map using successive calls to UIS\$SET_COLOR ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫.

The default writing color attribute setting in attribute block 0 is modified such that five new default writing colors are associated with a virtual color map entry ⑬ ⑭ ⑮ ⑯ ⑰.

The **atb** argument in the call to UIS\$CIRCLE within the DO loop references the modified attribute block. As a result, five circles are drawn horizontally ⑱ each with a different default writing color.

Five circles are drawn vertically ⑲ using the same colors as the horizontally drawn circles.

16-6 Programming in Color

16.2.6 Program Development II

Programming Objective

To create and load a color map with more than one entry at a time.

Programming Task

1. Load the arrays with color component values.
2. Establish color map size.
3. Load eight color map entries in a single call using UIS\$SET_COLORS.

```
PROGRAM MULTIPLE_ENTRY
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL J, K
REAL R_VECTOR(8), G_VECTOR(8), B_VECTOR(8)      ①
DATA J/17.0/                                     ②
DATA K/16/                                       ③
DATA R_VECTOR/0.40,0.50,0.50,0.0,0.25,0.90,0.80,0.35/ ④
DATA G_VECTOR/0.30,0.50,0.25,0.70,0.25,0.50,0.30,0.65/ ⑤
DATA B_VECTOR/0.0,0.50,0.50,0.30,0.90,0.0,0.0,0.95/ ⑥
DATA VCM_SIZE/8/

VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE)           ⑦
VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,40.0,15.0,15.0,VCM_ID) ⑧
WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'COLOR')

CALL UIS$SET_COLORS(VD_ID, 0, 8, R_VECTOR, G_VECTOR, B_VECTOR) ⑨

CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 9, 2)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 10, 3)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 11, 4)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 12, 5)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 13, 6)

DO I=9, 13, 1
CALL UIS$CIRCLE(VD_ID, I, J, 20.0, 10.0)
J=J+2.0
ENDDO

PAUSE

DO I=9, 13
CALL UIS$CIRCLE(VD_ID, I, 21.0, K, 10.0)
K=K+2.0
ENDDO

PAUSE

END
```


Three arrays are declared ❶ to hold eight R, G, and B color component values each.

The counters j and k are declared and initialized ❷ ❸.

The arrays R_VECTOR, G_VECTOR, and B_VECTOR are loaded with color component values ❹ ❺ ❻.

An eight-entry virtual color map is created ❼ and associated with a newly created virtual display ❽.

The R, G, and B color component values stored in the arrays are loaded in the virtual color map using a single call to UIS\$SET_COLORS ❾.

The remaining portions of the program are identical to the previous program SINGLE_ENTRY.

16.2.6.1 Program Development III

Programming Objective

To create a shareable color map.

Programming Task

1. Load arrays containing color component values.
2. Create the color map attributes list specifying the shareable attribute.
3. Create a virtual display specifying a name for the color map.
4. Create a display window and display viewport.
5. Load color values into the color map.
6. Program 2 must perform steps 2 through 4 and reference the name of the color map specified in Program 1.

```

PROGRAM SHAREABLE_MAP
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL J,K
REAL R_VECTOR(8),G_VECTOR(8),B_VECTOR(8)
INTEGER*4 VCM_ATTRIBUTES(3)
DATA J/17.0/
DATA K/16/
DATA R_VECTOR/0.40,0.50,0.50,0.0,0.25,0.90,0.80/
DATA G_VECTOR/0.30,0.50,0.25,0.70,0.25,0.50,0.30/
DATA B_VECTOR/0.0,0.50,0.50,0.30,0.90,0.0,0.0/
DATA VCM_SIZE/8/

VCM_ATTRIBUTES(1)=VCMAL$C_ATTRIBUTES
VCM_ATTRIBUTES(2)=VCMAL$M_SHARE
VCM_ATTRIBUTES(3)=VCMAL$C_END_OF_LIST

```

16-8 Programming in Color

```
VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE, 'LIVING_COLOR', VCM_ATTRIBUTES) ⑥
VD_ID=UIS$CREATE_DISPLAY(1.0, 1.0, 40.0, 40.0, 15.0, 15.0, VCM_ID) ⑨
WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'PROCESS #1')

CALL UIS$SET_COLORS(VD_ID, 0, 8, R_VECTOR, G_VECTOR, B_VECTOR)

CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 9, 2) ⑩
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 10, 3)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 11, 4)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 12, 5)
CALL UIS$SET_WRITING_INDEX(VD_ID, 0, 13, 6)

DO I=9, 13, 1
CALL UIS$CIRCLE(VD_ID, I, J, 20.0, 10.0)
J=J+2.0
ENDDO

VD_ID2=UIS$CREATE_DISPLAY(1.0, 1.0, 40.0, 40.0, 15.0, 15.0, VCM_ID) ⑪
WD_ID2=UIS$CREATE_WINDOW(VD_ID2, 'SYS$WORKSTATION', 'WINDOW #2')

CALL UIS$SET_WRITING_INDEX(VD_ID2, 0, 9, 2) ⑫
CALL UIS$SET_WRITING_INDEX(VD_ID2, 0, 10, 3)
CALL UIS$SET_WRITING_INDEX(VD_ID2, 0, 11, 4)
CALL UIS$SET_WRITING_INDEX(VD_ID2, 0, 12, 5)
CALL UIS$SET_WRITING_INDEX(VD_ID2, 0, 13, 6)

DO I=9, 13, 1
CALL UIS$CIRCLE(VD_ID2, I, 21.0, K, 10.0)
K=K+2.0
ENDDO

PAUSE

END
```

The counters *j* and *k* are declared and initialized ① ③ ④.

An integer array `VCM_ATTRIBUTES` is declared to have three elements ②.

The array elements are assigned attribute values defined by UIS constants ⑤ ⑥ ⑦. The structure contains an attribute code followed by a longword value for that attribute. The final element contains a longword 0 to terminate the list.

An eight-entry virtual color map is created using `UIS$CREATE_COLOR_MAP` and the array `VCM_ATTRIBUTES` is used as an argument ⑧.

The newly created virtual display references the virtual color map ⑨. Objects drawn in the virtual display can use this virtual color map.

Different default writing color are defined ⑩ as in previous programs simply to highlight and differentiate the objects drawn.

A second virtual display is created ❶. The second call to `UIS$CREATE_DISPLAY` references the same virtual color map identifier as the first. Both virtual displays will share the use of color value assignments in this virtual color map.

However, you must call `UIS$SET_WRITING_INDEX` ❷ again to change the default setting of the writing color so that objects will be drawn in colors identical to those drawn in the first virtual display.

Here is a portion of a second program that uses the virtual color map that uses the virtual color map `LIVING_COLOR` in the program `SHAREABLE_MAP`.

```

.
.
.
PROGRAM SECOND_PROGRAM
.
.
.
INTEGER*4 VCM_ATTRIBUTES(3) ❶
DATA VCM_SIZE/8/ ❷

VCM_ATTRIBUTES(1)=VCMAL$C_ATTRIBUTES
VCM_ATTRIBUTES(2)=VCMAL$M_SHARE
VCM_ATTRIBUTES(3)=VCMAL$C_END_OF_LIST
VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE, 'LIVING_COLOR', VCM_ATTRIBUTES) ❸
VD_ID2=UIS$CREATE_DISPLAY(1.0, 1.0, 35.0, 35.0, 10.0, 10.0, VCM_ID)

WD_ID2=UIS$CREATE_WINDOW(VD_ID2, 'SYS$WORKSTATION', 'PROCESS #2)
.
.
.

```

An array of virtual color map attributes specifying the same attributes as those indicated in the preceding program `SHAREABLE_MAP` ❶. The application `SECOND_PROGRAM` must declare the virtual color map size ❷ as this is a required argument in `UIS$CREATE_COLOR_MAP`.

The shareable color map is referenced by name in a call to `UIS$CREATE_COLOR_MAP` ❸.

16.3 Color Map Segments

Through the use of color map segments, you can control the binding of the virtual color map to the hardware color map.

16-10 Programming in Color

16.3.1 Programming Options

Creating and Deleting Color Map Segments

You can create and delete color map segments using `UIS$CREATE_COLOR_MAP_SEG` and `UIS$DELETE_COLOR_MAP_SEG`.

16.3.2 Program Development

The program `COLOR_SEG` is a portion of a longer program and shows how to bind your virtual color map to the hardware color map.

```
PROGRAM COLOR_SEG
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
INTEGER*4 VCM_ATTRIBUTES(3) ①
DATA VCM_SIZE,PLACEMENT_DATA/8,16/ ②
.
.
.
VCM_ATTRIBUTES(1)=VCMAL$C_ATTRIBUTES ③
VCM_ATTRIBUTES(2)=VCMAL$M_NOBIND ④
VCM_ATTRIBUTES(3)=VCMAL$C_END_OF_LIST ⑤
VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE,,VCM_ATTRIBUTES) ⑥
CMS_ID=UIS$CREATE_COLOR_MAP_SEG(VCM_ID,'SYS$WORKSTATION',
2 UIS$C_COLOR_EXACT,PLACEMENT_DATA) ⑦
VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,30.0,30.0,10.0,10.0,VCM_ID)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
.
.
.
```

Two declarations are established—an array `VCM_ATTRIBUTES` is declared ① and the virtual color map size is initialized to 8 ②.

Because the color map segment is created with exact placement, the **placement_data** argument of `UIS$CREATE_COLOR_MAP_SEG` must be initialized to the starting index in the hardware color map where binding is to occur.

The elements of array `VCM_ATTRIBUTES` are assigned an attribute code ③, an attribute value `VCMAL$M_NOBIND` ④, and a terminating value ⑤.

`UIS$CREATE_COLOR_MAP` is called before any other `UIS` routine.

16.3.3 Calling UIS\$CREATE_COLOR_MAP_SEG

No special graphics effects are displayed on the VAXstation screen.

16.4 Color and Intensity Inquiry Routines

As mentioned previously in Chapter 12, certain routines called inquiry routines provide application with status information. There are several UIS color and intensity routines that return information to the application. Color and intensity inquiry routines return information about the color setup, virtual color map, and hardware color map. Such information can be used as direct input to your application.

16.4.1 Programming Options

Your application can use one or more inquiry routines, where appropriate. Table 16-3 lists color and intensity inquiry routines.

Table 16-3 Color and Intensity Inquiry Routines

Routine	Information Returned
Virtual Color Map	
UIS\$GET_COLOR	Single RGB value from a virtual color map
UIS\$GET_COLORS	Multiple RGB values from a virtual color map
UIS\$GET_INTENSITIES	Multiple intensity values from a virtual color map
UIS\$GET_INTENSITY	Single intensity value from a virtual color map
Hardware Color Map	
UIS\$GET_HW_COLOR_INFO	Device type; number of indexes; number of colors; bits of precision for R, G, and B values; reserved entries; and regeneration characteristics.
Color Value Conversion	
UIS\$HLS_TO_RGB	Converts HLS color values to RGB color values
UIS\$HSV_TO_RGB	Converts HSV color values to RGB color values
UIS\$RGB_TO_HLS	Converts RGB color values to HLS color values
UIS\$RGB_TO_HSV	Converts RGB color values to HSV color values

16-12 Programming in Color

Table 16-3 (Cont.) Color and Intensity Inquiry Routines

Routine	Information Returned
Workstation Standard Colors	
UIS\$GET_WS_COLOR	Workstation standard RGB color value
UIS\$GET_WS_INTENSITY	Workstation standard intensity value
Color Setup	
UIS\$GET_BACKGROUND_INDEX	Window background index
UIS\$GET_WRITING_INDEX	Window foreground index
UIS\$GET_WRITING_MODE	Writing mode

16.4.2 Program Development I

Programming Objective

To retrieve hardware color map information.

Programming Tasks

1. Create a virtual color map.
2. Create a virtual display.
3. Create a display window and viewport.
4. Obtain the number of color map indices, possible colors, maps, bits of precision for each color component, and reserved entries.

```
PROGRAM GET_INFO
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
REAL J,K
REAL R_VECTOR(8),G_VECTOR(8),B_VECTOR(8)

REAL RETR_VECTOR(8),RETG_VECTOR(8),RETB_VECTOR(8)
INTEGER*4 VCM_ATTRIBUTES(3)
DATA J/17.0/
DATA K/16/
DATA R_VECTOR/0.40,0.50,0.50,0.0,0.25,0.90,0.80,0.35/
DATA G_VECTOR/0.30,0.50,0.25,0.70,0.25,0.50,0.30,0.65/
DATA B_VECTOR/0.0,0.50,0.50,0.30,0.90,0.0,0.0,0.95/

VCM_ATTRIBUTES(1)=VCMAL$C_ATTRIBUTES
VCM_ATTRIBUTES(2)=VCMAL$M_SHARE
VCM_ATTRIBUTES(3)=VCMAL$C_END_OF_LIST
```

```

VCM_ID=UIS$CREATE_COLOR_MAP(VCM_SIZE,VCM_ATTRIBUTES)
VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,40.0,40.0,15.0,15.0,VCM_ID)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','COLOR')

CALL UIS$SET_COLORS(VD_ID,0,8,R_VECTOR,G_VECTOR,B_VECTOR)

CALL UIS$SET_WRITING_INDEX(VD_ID,0,9,2)
CALL UIS$SET_WRITING_INDEX(VD_ID,0,10,3)
CALL UIS$SET_WRITING_INDEX(VD_ID,0,11,4)
CALL UIS$SET_WRITING_INDEX(VD_ID,0,12,5)
CALL UIS$SET_WRITING_INDEX(VD_ID,0,13,6)

CALL UIS$GET_COLORS(VD_ID,0,8,RETR_VECTOR,RETG_VECTOR,RETB_VECTOR) ❶

TYPE 50
50  format(T8,'RED',T18,'GREEN',T30,'BLUE')

TYPE 100,RETR_VECTOR,RETG_VECTOR,RETB_VECTOR
100  FORMAT(F11.3,F11.3,F11.3)

CALL UIS$GET_HW_COLOR_INFO(, ,
2     INDICES,COLORS,MAPS,RBITS,GBITS,BBITS, ,RES_INDICES) ❷

TYPE 150,INDICES,COLORS
150  FORMAT(T2,'NO. OF INDICES=',I3,T22,'NO. OF COLORS=',I8)

TYPE 200,MAPS
200  FORMAT(T2,'NO.OF MAPS=',i3)

TYPE 225,RBITS,GBITS,BBITS
225  FORMAT(T2,'NO. OF BITS OF PRECISION',T28,'RED',I3,T37,'GREEN',I3,
2     T48,'BLUE',I3)

TYPE 250,RES_INDICES
250  FORMAT(T2,'NO. OF RESERVED ENTRIES',I3)

TYPE*,'VCM Indexes Used In Virtual Display 1'

DO I=9,13,1
CALL UIS$CIRCLE(VD_ID,I,J,20.0,10.0)
INDEX=UIS$GET_WRITING_INDEX(VD_ID,I) ❸
TYPE*,INDEX
J=J+2.0
ENDDO

VD_ID2=UIS$CREATE_DISPLAY(1.0,1.0,40.0,40.0,15.0,15.0,VCM_ID)
WD_ID2=UIS$CREATE_WINDOW(VD_ID2,'SYS$WORKSTATION','WINDOW #2')

CALL UIS$SET_WRITING_INDEX(VD_ID2,0,9,2)
CALL UIS$SET_WRITING_INDEX(VD_ID2,0,10,3)
CALL UIS$SET_WRITING_INDEX(VD_ID2,0,11,4)
CALL UIS$SET_WRITING_INDEX(VD_ID2,0,12,5)
CALL UIS$SET_WRITING_INDEX(VD_ID2,0,13,6)

```

16-14 Programming in Color

```
TYPE*, 'VCM Indexes Used In Virtual Display 2'  
DO I=9,13  
CALL UIS$CIRCLE(VD_ID2,I,21.0,K,10.0)  
INDEX=UIS$GET_WRITING_INDEX(VD_ID2,I) ④  
TYPE*,INDEX  
K=K+2.0  
ENDDO  
  
PAUSE  
  
END
```

With the inclusion of only three inquiry routines, a great deal of information is returned. A call to `UIS$GET_COLORS` ① returns the R, G, and B color component values in the color map entries of the virtual color map.

A call to `UIS$GET_HW_COLOR_INFO` ② returns the number of binary bits of precision for R, G, and B color map values. In addition, total number of hardware color map entries as well as the number of reserved entries.

Writing color information must be returned from two locations in the program. The first call to `UIS$GET_WRITING_INDEX` within the DO loop ③ returns all the default writing indexes as they are being used in the first virtual display.

The second call to `UIS$GET_WRITING_INDEX` ④ returns each writing index used to draw graphic objects in the second virtual display.

16.4.2.1 Calling `UIS$GET_COLORS`, `UIS$GET_HW_COLOR_INFO`, `UIS$GET_WRITING_INDEX`

Figure 16-1 shows the information returned in the user's emulation window.

Figure 16-1 Different Types of Information Returned from Inquiry Routines

```

$ run get_info
      red      green      blue
0.400      0.500      0.500
0.000      0.250      0.900
0.800      0.000      0.300
0.500      0.250      0.700
0.250      0.500      0.300
0.000      0.000      0.500
0.500      0.300      0.900
0.000      0.000      0.000
no. of indices=256  no. of colors=16777216
no. of maps= 1
no. of bits of precision  red  8  green  8  blue  8
no. of reserved entries  6
VCM Indexes Used In Virtual Display 1
      2
      3
      4
      5
      6
FORTRAN PAUSE
$

```

ZK 5453-86

16.4.3 Program II—Creating an HSV Color Wheel

```
PROGRAM COLOR_WHEEL
```

```
c
```

```
c This program draws a color wheel once and then continually
c changes its appearance by updating the virtual color map.
```

```
c
```

```

IMPLICIT INTEGER*4(A-Z)
PARAMETER DISPLAY_SIZE=4.0*2.54
REAL*4 R,G,B,H,L,S,V,START_DEG,END_DEG
REAL*4 R_VECTOR(0:255),G_VECTOR(0:255),B_VECTOR(0:255)
INCLUDE 'SYS$LIBRARY:UISUSRDEF'

```

16-16 Programming in Color

```
c
c Find out some information about the workstation color characteristics
c
      CALL UIS$GET_HW_COLOR_INFO(,INDICES,,MAPS,,,,RES_INDICES,REGEN)
c
c Only attempt to run this program on color map hardware systems.
c
      IF (MAPS .EQ. 0 .OR. REGEN .NE. UIS$C_DEV_RETRO) STOP
c
c Make the virtual color map size dependent upon the available
c hardware, but no greater than 64 entries
c
      MAP_SIZE=MIN(INDICES-RES_INDICES, 64)
      VCM_ID=UIS$CREATE_COLOR_MAP(MAP_SIZE)
c
c Create the virtual display and a single window
c
      VD_ID=UIS$CREATE_DISPLAY(0.0, 0.0, 1.0, 1.0,
1          DISPLAY_SIZE, DISPLAY_SIZE, VCM_ID)
      WD_ID=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION')
c
c Establish some attributes for drawing
c
      CALL UIS$SET_ARC_TYPE(VD_ID, 0, 1, UIS$C_ARC_PIE)
      CALL UIS$SET_FONT(VD_ID, 1, 1, 'UIS$FILL_PATTERNS')
      CALL UIS$SET_FILL_PATTERN(VD_ID, 1, 1, PATT$C_FOREGROUND)
c
c Set window background to black and draw wedges of a circle.
c The initial colors of the wedges are determined by traversing
c 360 degrees around the HSV color model, varying H, while S and
c V are both 1.0.
c
      CALL UIS$SET_COLOR(VD_ID, 0, 0.0, 0.0, 0.0)
      DO I=1,MAP_SIZE-1
          START_DEG=(I-1)*(360.0/FLOAT(MAP_SIZE-1))
          END_DEG=START_DEG+(360.0/FLOAT(MAP_SIZE-1))
          CALL UIS$HSV_TO_RGB(START_DEG, 1.0, 1.0, R, G, B)
          CALL UIS$SET_COLOR(VD_ID, I, R, G, B)
          CALL UIS$SET_WRITING_INDEX(VD_ID, 1, 1, I)
          CALL UIS$CIRCLE(VD_ID, 1, 0.5, 0.5, 0.4, START_DEG, END_DEG)
      END DO
```

```

c
      V=1.0
c
c The next set of sequential and nested loops
c traverse the HSV color model cone.
c
100   CONTINUE
c
c Vary S from 1.0 to 0.0 in 0.01 increments
c
      DO IS=99,0,-1
      S=FLOAT(IS)/100.0
c
      DO I=1,MAP_SIZE-1
      START_DEG=(I-1)*(360.0/FLOAT(MAP_SIZE-1))
      IF (S .EQ. 0.0) START_DEG=UIS$C_COLOR_UNDEFINED
      CALL UIS$HSV_TO_RGB(START_DEG, S, V,
1      R_VECTOR(I), G_VECTOR(I), B_VECTOR(I))
      END DO      ! I
      CALL UIS$SET_COLORS(VD_ID, 1, MAP_SIZE-1,
1      R_VECTOR(1), G_VECTOR(1), B_VECTOR(1))
c
      end do      ! s=1.0,0.0
c
c Vary V from 1.0 to 0.0 in 0.01 increments
c
      DO IV=99,0,-1
      V=FLOAT(IV)/100.0
c
      DO I=1,MAP_SIZE-1
      START_DEG=(I-1)*(360.0/FLOAT(MAP_SIZE-1))
      IF (S .EQ. 0.0) START_DEG=UIS$C_COLOR_UNDEFINED
      CALL UIS$HSV_TO_RGB(START_DEG, S, V,
1      R_VECTOR(I), G_VECTOR(I), B_VECTOR(I))
      END DO      ! I
      CALL UIS$SET_COLORS(VD_ID, 1, MAP_SIZE-1,
1      R_VECTOR(1), G_VECTOR(1), B_VECTOR(1))
c
      END DO      ! V=1.0,0.0

```

16-18 Programming in Color

```
c
c Vary V from 0.0 to 1.0 in 0.01 increments
c
  DO IV=1,100, 1
  V=FLOAT(IV)/100.0
c
  DO I=1,MAP_SIZE-1
    START_DEG=(I-1)*(360.0/FLOAT(MAP_SIZE-1))
    IF (S .EQ. 0.0) START_DEG=UIS$C_COLOR_UNDEFINED
    CALL UIS$HSV_TO_RGB(START_DEG, S, V,
  1      R_VECTOR(I), G_VECTOR(I), B_VECTOR(I))
  END DO      ! I
  CALL UIS$SET_COLORS(VD_ID, 1, MAP_SIZE-1,
  1      R_VECTOR(1), G_VECTOR(1), B_VECTOR(1))
c
  END DO      ! V=0.0,1.0
c
c Vary S from 0.0 to 1.0 in 0.01 increments
c
  DO IS=1,100,1
  S=FLOAT(IS)/100.0
c
  DO I=1,MAP_SIZE-1
    START_DEG=(I-1)*(360.0/FLOAT(MAP_SIZE-1))
    IF (S .EQ. 0.0) START_DEG=UIS$C_COLOR_UNDEFINED
    CALL UIS$HSV_TO_RGB(START_DEG, S, V,
  1      R_VECTOR(I), G_VECTOR(I), B_VECTOR(I))
  END DO      ! I
  CALL UIS$SET_COLORS(VD_ID, 1, MAP_SIZE-1,
  1      R_VECTOR(1), G_VECTOR(1), B_VECTOR(1))
c
  END DO      ! S=0.0,1.0
c
c Repeat HSV color cone traversal indefinitely
c
  GOTO 100
c
  END
```

Chapter 17

Asynchronous System Trap Routines

17.1 Overview

Frequently, an application program relies on certain run-time events to trigger the execution of an application-specific task. Such run-time events can range from power failure to simply striking a key on the keyboard. Several UIS routines *enable* this type of behavior for the duration of the program or until the enabling UIS routine is explicitly disabled. Such routines enable the use of asynchronous system trap (AST) routines. This chapter discusses AST routines and how they can be used to perform the following tasks:

- Creating a virtual keyboard
- Creating a pointer pattern
- Using a pointer
- Resizing a display window
- Closing a display window
- Shrinking a display viewport to an icon

The use of AST routines is not restricted to the tasks listed here.

17.1.1 Using AST Routines

Generally speaking, certain UIS routines associate or, *bind*, a specific run-time event or action to a subroutine. When that action occurs, control passes from the main program to a user-written subroutine. The subroutine then performs some application-specific task. When the subroutine completes execution, control is transferred to the next statement in the main program. However, the association between the run-time event and the execution of the subroutine remains in effect. If the action occurred again during program execution, the subroutine would be called again. The process executing the main program is suspended when the run-time event occurs and until the subroutine completes execution. Thus, execution of the

17-2 Asynchronous System Trap Routines

subroutine occurs asynchronously with respect to execution of the main program. The user-written subroutine is known as an *asynchronous system trap* routine or AST routine.

As with any subprogram or subroutine, the AST routine can be coded within the main program according to the conventions of the particular programming language or separately as a module in a library. However, to make use of such modules, you must compile and link them with your program.

17.1.2 AST-Enabling Routines

Several UIS routines enable AST routine execution whenever a particular run-time event occurs. The actual event may involve the keyboard, pointer, or the occurrence of a program-related event, such as the movement or resizing of a window. Such *AST-enabling* routines reference AST routines in their argument lists. Table 17-1 lists each AST-enabling routine and the event that triggers AST routine execution.

Table 17-1 AST-Enabling Routines

Routine	Event
UIS\$SET_ADDOPT_AST	An additional option is chosen using the human interface.
UIS\$SET_BUTTON_AST	A button is depressed or released on a pointer device.
UIS\$SET_CLOSE_AST	A display window is deleted with the human interface.
UIS\$SET_TB_AST	A digitizer is moved within a specified data region on the tablet.
UIS\$SET_EXPAND_ICON_AST	An icon is expanded to display viewport with the user interface.
UIS\$SET_GAIN_KB_AST	A virtual keyboard is bound to a physical keyboard.
UIS\$SET_KB_AST	A key is struck.
UIS\$SET_LOSE_KB_AST	A virtual keyboard is disconnected from a physical keyboard.
UIS\$SET_MOVE_INFO_AST	A window is moved in the virtual display.
UIS\$SET_POINTER_AST	A pointer moves into or exits an area of the virtual display.
UIS\$SET_RESIZE_AST	A display window is resized with the human interface.
UIS\$SET_SHRINK_ICON_TO_AST	A display viewport is shrunk with the human interface.

17.2 Using Keyboard and Pointer Devices

The keyboard and pointer devices are resources for use within your application program. The keyboard and pointer are mentioned here to illustrate routines that can make use of input from such workstation peripheral devices during application program execution. An effective way of using keyboard and pointer devices is in conjunction with AST routines.

17.2.1 Using AST Routines with Virtual Keyboards

You can use your keyboard as a virtual device whose characteristics are transportable from virtual display to virtual display. When the keyboard is used as a virtual device, you can create an unlimited number of them (subject to system and process resources) with different characteristics and associate each with any virtual display you choose.

17.2.1.1 Step 1—Creating a Virtual Keyboard

A virtual keyboard is created using `UIS$CREATE_KB`. There is no limit to the number of virtual keyboards you can create.

17.2.1.2 Step 2—Binding the Virtual Keyboard to the Display Window

In addition, you must bind the virtual keyboard to a specified display window using `UIS$ENABLE_VIEWPORT_KB` or `UIS$ENABLE_KB`. `UIS$ENABLE_VIEWPORT_KB` and `UIS$ENABLE_KB` also define how the physical keyboard and the virtual keyboard are assigned to each other.

If your display screen contains one or more display viewports and you have assigned virtual keyboards to their associated display windows, you can move from display viewport to viewport through the assignment list using the `CYCLE` key. An assignment list of display windows keeps track of which viewport is active.

A viewport is active when the KB icon background color on the viewport is highlighted. The physical keyboard is now assigned to a virtual keyboard. The virtual keyboard and all enabled characteristics can then be used with the physical keyboard. You can bind more than one display window to the same virtual keyboard. In this case, the KB icons in all display viewports are highlighted at the same time when the desired windows are assigned a physical keyboard.

Table 17-2 shows how each routine performs physical-to-virtual keyboards assignments.

17-4 Asynchronous System Trap Routines

Table 17-2 Connecting Physical Keyboards and Virtual Keyboards

Routine	Function
UIS\$ENABLE_VIEWPORT_KB	Adds the display window to the assignment list. Use the [CYCLE] key to move from viewport to viewport.
UIS\$ENABLE_KB	Places the display window at the top of the assignment list and is active. Use the [CYCLE] key to move to other viewports.

To terminate the binding of the specified virtual keyboard to the physical keyboard, use `UIS$DISABLE_VIEWPORT_KB` or `UIS$DISABLE_KB`. Table 17-3 shows how each routine terminates physical-to-virtual keyboard assignments.

Table 17-3 Disconnecting Physical Keyboards and Virtual Keyboards

Routine	Function
UIS\$DISABLE_VIEWPORT_KB	Removes a display window from the assignment list. Invoke <code>UIS\$ENABLE_VIEWPORT_KB</code> or <code>UIS\$ENABLE_KB</code> to make the viewport active.
UIS\$DISABLE_KB	Places a display window at the bottom of the assignment list. Use the [CYCLE] key to make the viewport active.

17.2.1.3 Step 3—Enabling Virtual Keyboard AST Routines

Even though you have created a virtual keyboard and you have bound it to a specified display window, you still cannot write characters to that display window. You must associate the act of striking a key with the action taken by a subroutine using `UIS$SET_KB_AST`.

17.2.2 Programming Options

After you have created the virtual keyboard, your application may verify successful connection with the physical keyboard. You may want to be notified in the event such connections are made or broken. These and other options are available to your application.

Gaining and Losing Virtual Keyboards

Connecting and disconnecting virtual keyboards may occur many times within your application program. These events may be so significant that whenever a virtual keyboard is disconnected or lost, you may want your program to initiate some action through a subroutine. For example, `UIS$SET_GAIN_KB_AST` and `UIS$SET_LOSE_KB_AST` enable AST routines that could allow your program to perform

housekeeping functions such as deleting unused virtual keyboards, display windows, and display viewports when a virtual keyboard is disconnected.

Enabling and Disabling Keyboard Characteristics

Keyboard characteristics are assigned to specific virtual keyboards using `UIS$SET_KB_ATTRIBUTES`.

Testing Physical Keyboards

You can verify the connection between a specified virtual keyboard and the physical keyboard with `UIS$TEST_KB`.

Deleting Virtual Keyboards

To delete a virtual keyboard, use `UIS$DELETE_KB`.

17.2.3 Program Development

Programming Objectives

To type keyboard characters directly to the virtual display using AST routines.

Programming Tasks

1. Declare subroutine and the appropriate variables to be included in the `COMMON` statement.
2. Create a virtual display.
3. Create a virtual keyboard.
4. Create a display window and viewport.
5. Bind the virtual keyboard to the display window.
6. Enable keyboard AST routines using `UIS$SET_KB_AST`.
7. Create a subroutine to send each keystroke to the virtual display.

```
PROGRAM AST
  IMPLICIT INTEGER(A-Z)
  INCLUDE 'SYS$LIBRARY:UISENTRY'
  INCLUDE 'SYS$LIBRARY:UISUSRDEF'
  LOGICAL*1 KEYBUF(4)
  EXTERNAL KEYSTRIKE ①
  COMMON KB_ID,VD_ID,KEYBUF,WD_ID,COUNT ②

  VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,31.0,31.0,20.0,5.0)
  KB_ID=UIS$CREATE_KB('SYS$WORKSTATION') ③
```

17-6 Asynchronous System Trap Routines

```
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','KEYBOARD AST')
CALL UIS$ENABLE_VIEWPORT_KB(KB_ID,WD_ID) ④
CALL UIS$SET_ALIGNED_POSITION(VD_ID,1,1.0,30.0)
COUNT=0
CALL UIS$SET_KB_AST(KB_ID,KEYSTRIKE,0,KEYBUF) ⑤
CALL SYS$HIBER() ⑥
END
SUBROUTINE KEYSTRIKE ⑦
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
LOGICAL*1 KEYBUF(4)
COMMON KB_ID,VD_ID,KEYBUF,WD_ID,COUNT ⑧
STRUCTURE/TEXT/ ⑨
INTEGER*2 BUFLN,BUFCODE
INTEGER*4 BUFADR
END STRUCTURE
RECORD/TEXT/DESC ⑩
DESC.BUFLN=1
DESC.BUFADR=%LOC(KEYBUF) ⑪
STATUS=UIS$TEST_KB(KB_ID) ⑫
CALL UIS$SET_FONT(VD_ID,1,2,'MY_FONT_13')
IF ((COUNT .EQ. 60) .OR. (KEYBUF(1) .EQ. 13)) THEN ⑬
CALL UIS$NEW_TEXT_LINE(VD_ID,2)
COUNT=0
ELSE
CALL UIS$TEXT(VD_ID,2,DESC)
COUNT=COUNT+1
END IF
RETURN
END
```

The name of the AST routine KEYSTRIKE is declared ① using the EXTERNAL statement. The EXTERNAL statement defines the symbolic name of the routine as an address. The routine name can then be used as an argument in a parameter list as in the **astadr** argument of an AST-enabling routine.

The COMMON statement allows certain variables used in both program units (the main program and the subroutine) to share the same storage area ② ③. You can use either the COMMON statement or the **astprm** argument in the AST-enabling routine to pass data to the AST routine.

The virtual keyboard is created ④ and bound to a display window ④.

In our program the AST-enabling routine that references the subroutine KEYSTRIKE ⑤ ⑦ is UIS\$SET_KB_AST. Note that there is no separate call to the subroutine KEYSTRIKE.

Whenever a key is struck, the ASCII character code for that character is stored in the variable *keybuf* ⑤ and subroutine KEYSTRIKE is executed. The subroutine KEYSTRIKE is an AST routine.

The subroutine KEYSTRIKE retrieves the character code stored in the variable *keybuf*. The data structure TEXT, a character string descriptor, is created ⑨. The variable DESC denoting a record is defined to have the same structure as TEXT ⑩. The address of *keybuf* is assigned to a longword in the descriptor ⑪. The subroutine KEYSTRIKE writes the character to the virtual display using UIS\$TEXT.

After the AST routine completes execution, control returns to the next statement in the main program. The next statement is a call to the SYS\$HIBER system service ⑥. The SYS\$HIBER allows the process to remain inactive until the next time the AST routine is executed, that is, when a key is struck.

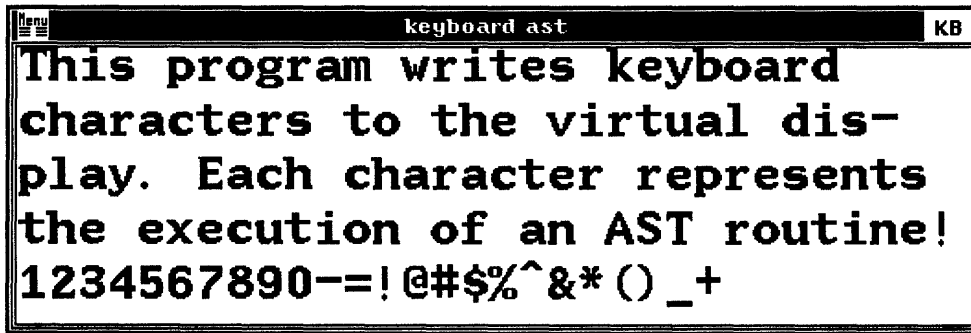
The AST routine KEYSTRIKE also verifies that the virtual and physical keyboards are connected ⑫.

Whenever column 60 is reached or the RETURN key is pressed ⑬, text output moves to the next line. The ASCII character code for the RETURN key is 13.

17.2.4 Calling Keyboard Routines

The program AST creates a viewport to which characters are written as shown in Figure 17-1.

Figure 17-1 Writing Characters to a Display Viewport



ZK-4561-85

17.2.5 Using AST Routines with Pointer Devices

Pointer routines allow the pointer to act as an input device to your application program. Typically, application programs use such data to keep track of the location of the pointer device in the virtual display, or the location of a specified rectangle in the virtual display. An effective way of using pointers in this manner is through AST routines.

17.2.5.1 Mouse

The mouse is a relative pointing device and can be used with AST routines to return status information about mouse location to the application.

17.2.5.2 Tablet

Another pointing device that you can use is the digitizer. The tablet consists of a puck or stylus and a tablet. Digitizer support may be used only with a tablet.

Digitizing with a Tablet

Digitizing with a tablet requires that you establish a region on the tablet—the data rectangle. The data rectangle is the area on the tablet in which digitizing is active. If you do not specify a data rectangle, the whole tablet is used.

Only one data digitizing region may be active at one time.

The pointer position on the tablet is available to the user's digitizing AST routine, if desired. If the pointer is within the data rectangle, then the user's AST routine is executed.

Only one image may own the tablet at any one time. When a process connects to the tablet, the system hardware cursor is turned off and the connected process receives all the input from the tablet device. The process must use a software cursor if it wishes to track the pointer in a window. The process owns the tablet until it makes a call to `UIS$ENABLE_TB` to disconnect itself from the tablet.

Mouse Operation

A mouse cannot be used as a data digitizer. The UIS routine will report an error, if you attempt to digitize with the mouse.

Terminating Data Digitizing

Only the process that issues the data digitizing request may change or cancel the request. If the process is deleted and the channel is deassigned, data digitizing is immediately canceled, if a request is still outstanding.

Only one data digitizing region may be active at a time. Attempts by other processes to initiate will fail if another process has already declared a digitizing region.

17.2.5.3 Step 1—Create an AST Routine

You must write a program that includes an AST subroutine that performs a task. Typically, AST subroutines perform inquiry functions and return pointer information such as location to the main program. See Table 12-1 for a list of pointer routines that return information about the pointer. You are not restricted to using AST routines in this manner. For example, you can use AST routines with pointers to create menus.

17.2.5.4 Step 2—Enable the AST Routine

The AST routine will execute whenever a specific run-time event occurs. However, you must enable this behavior by including an AST-enabling routine in the main program. The following table lists pointer AST-enabling routines.

Routine	Run-Time Event
<code>UIS\$SET_BUTTON_AST</code>	The button on the pointer device is depressed.
<code>UIS\$SET_TB_AST</code>	The digitizer is moved within a specified data region on the tablet.
<code>UIS\$SET_POINTER_AST</code>	The pointer is moved into a specified region of virtual display.

17.2.6 Programming Options

Many graphics applications use the pointer position and movement as a means of drawing objects on the display screen. Graphics routines can use this information to generate objects.

17-10 Asynchronous System Trap Routines

Pointer Movement

Many application programs need to know where the pointer is. For example, the program might need to perform some type of action whenever the pointer moves within certain regions of the virtual display. The AST-enabling routine `UIS$SET_POINTER_AST` can be used whenever pointer movement plays an important role in program execution.

Pointer Position

Your application may need to establish the position of the pointer in world coordinates. In addition, `UIS$SET_POINTER_POSITION` returns a status value.

Pointer Pattern

You can change the appearance of the pointer cursor using `UIS$SET_POINTER_PATTERN`. Normally, this cursor appears as an arrow on the display screen. The pointer cursor, or *pattern* represents bit settings within an array of 16 words. You can choose your own pointer pattern by assigning a value to each word in the array that will set the desired bits for the new pattern.

Optionally, you may request that the pointer also be bound to the region specified in the `UIS$SET_POINTER_PATTERN` call. When this region is unoccluded, the pointer pattern will not be allowed to exit after it has been positioned within the region. The cursor may leave the bound region, if it becomes occluded.

Tablet Information

Currently, two routines `UIS$GET_TB_INFO` and `UIS$GET_TB_POSITION` return information about tablet characteristics and position, respectively.

17.2.7 Program Development

Programming Objective

To change the default pointer pattern to a crosshair.

Programming Tasks

1. Declare the subroutine and the appropriate variables in the `COMMON` statement.
2. Create a virtual display.
3. Create a display window and viewport.
4. Enable pointer AST routine with `UIS$SET_POINTER_AST`.

5. Create a subroutine that defines the new cursor pattern.

```

PROGRAM PATTERN
IMPLICIT INTEGER(A-Z)
EXTERNAL FIGURE ❶
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON VD_ID,WD_ID

VD_ID=UIS$CREATE_DISPLAY(-1.0,-1.0,30.0,30.0,20.0,20.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','POINTER PATTERN') ❷

CALL UIS$SET_POINTER_AST(VD_ID,WD_ID,FIGURE,0) ❸

CALL SYS$HIBER()

END

SUBROUTINE FIGURE
IMPLICIT INTEGER(A-Z)
INTEGER*2 CURSOR(16) ❹
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON VD_ID,WD_ID

DATA CURSOR/7*896,65535,8*896/ ❺

CALL UIS$SET_POINTER_PATTERN(VD_ID,WD_ID,CURSOR,,8,8) ❻

RETURN
END

```

In this program, no world coordinates are specified for the display window ❷, so the display window maps the entire virtual display space. The dimensions of the display viewport are also not specified. As a result, the display viewport size defaults to the dimensions specified in `UIS$CREATE_DISPLAY`.

The subroutine `FIGURE` is called whenever the pointer lies within the specified area of the display window. In the main program the subroutine is declared as an external procedure ❶ in the main program. The AST-enabling routine for the pointer devices `UIS$SET_POINTER_AST` is called ❸. Because no rectangle is specified, the subroutine `FIGURE` is executed whenever the pointer is within the display window.

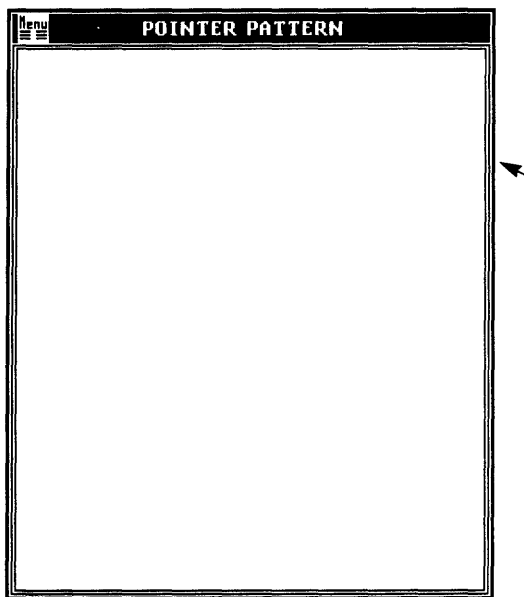
The array `CURSOR` is declared in the subroutine `FIGURE` ❹ and contains 16 elements. Each array element is declared as a word and is, therefore, 16 bits long. You should imagine the array as a 16 by 16-bit pattern, or matrix. Each array element ❺ is assigned a value which sets certain bits in the matrix to 1. The matrix represents the bitmap image of the new cursor pattern. The call to `UIS$SET_POINTER_PATTERN` references the new cursor pattern and the exact bit in the new cursor pattern used to calculate current pointer position ❻.

17-12 Asynchronous System Trap Routines

17.2.8 Calling UIS\$SET_POINTER_AST and UIS\$SET_POINTER_PATTERN

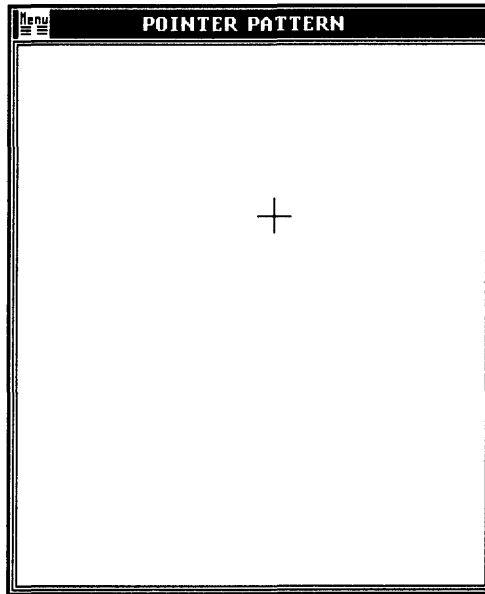
When you run the program PATTERN, the display viewport is created. The pointer lies outside the display viewport and the default pointer pattern is in effect as shown in Figure 17-2.

Figure 17-2 Default Pointer Pattern



ZK-4614-85

The process executing the main program is hibernating, that is, waiting for you to move the pointer. As you can see in Figure 17-3, when you move the pointer within the display window, the pointer pattern changes from an arrow to a cross.

Figure 17-3 New Pointer Pattern

ZK-4562-85

17.3 Manipulating Display Windows and Viewports

Default Shrinking Operation

The shrinking of viewports is performed using the Window Options Menu by default. When you choose the "Shrink to an Icon" menu item, `UIS$SHRINK_TO_ICON` is called. You can also expand icons to viewports with the user interface by placing the cursor in the icon and pressing the pointer button.

Default Resizing and Closing Operations

Resizing and closing display windows are performed using the Window Options Menu by default. When you choose the "Change the size" menu item, `UIS$RESIZE_WINDOW` is called and accepts the world coordinate values of the newly resized window.

Display windows are also closed using the Window Options Menu. When you choose the "Delete" menu item, `UIS$CLOSE_WINDOW` is called, which, in turn, calls `SYS$EXIT` system service. `SYS$EXIT` performs image rundown and deletes the process that owns the image.

17-14 Asynchronous System Trap Routines

17.3.1 Using AST Routines to Modify the Window Options Menu

Certain UIS routines can override the default actions listed in the Window Options Menu and enable user-written shrinking, expanding, resizing and closing AST routines that are activated whenever the “Shrink to an Icon”, “Change the size”, or “Delete” menu items are chosen. In this case, AST routines override the default shrinking, expansion, resizing, and closing UIS behavior.

17.3.1.1 Step 1—Create an AST Routine

In order to override one of the default actions listed in the Window Options Menu, you must write a program that includes an AST routine. When you execute the program and initiate the action through the user interface, the default action is no longer performed automatically.

You could code your AST routine so that it could perform any action. Most likely, you will modify the action of a menu item by adding additional actions to the default. If so, you must include in your AST routine a call to `UIS$RESIZE_WINDOW` in addition to code to perform any other special features you see fit. When the program executes, the AST routine will perform the resize as well as any other additional actions. The following table lists the task you wish to perform and the corresponding UIS routine that should be included in your subroutine.

Task	Routine
Close or delete a window	<code>UIS\$CLOSE_WINDOW</code>
Expand an icon ¹	<code>UIS\$EXPAND_ICON</code>
Resize a viewport	<code>UIS\$RESIZE_WINDOW</code>
Shrink a viewport	<code>UIS\$SHRINK_TO_ICON</code>

¹Not listed in the Window Options Menu.

17.3.1.2 Step 2—Enable the AST Routine

You want your AST routine to execute whenever you wish to override the default features listed in the Window Options Menu. In order to execute the AST routine, a run-time event must occur to trigger the AST routine. Therefore, you must include in your main program an appropriate AST-enabling routine. The following table lists window AST-enabling routines that trigger AST routine execution for various run-time events.

Routine	Run-Time Event
UIS\$SET_CLOSE_AST	The "Delete" menu item is chosen using the user interface.
UIS\$SET_EXPAND_ICON_AST	The pointer pattern is placed on an icon and the pointer button is depressed.
UIS\$SET_RESIZE_WINDOW_AST	The "Change the size" menu item is chosen using the user interface
UIS\$SET_SHRINK_TO_ICON_AST	The "Shrink to an icon" menu item is chosen using the user interface.

17.3.2 Programming Options

You can enable AST routine execution for the programming options listed below.

Shrinking Viewports to Icons

You can override the default display viewport shrinking operation by first calling the AST-enabling routine `UIS$SET_SHRINK_TO_ICON_AST` in your main program. Your AST routine will contain `UIS$SHRINK_TO_ICON` specifying icon attributes. Shrinking viewports to icons occurs as a five-step as follows:

1. The user initiates the shrinking operation using the user interface.
2. The viewport is moved offscreen using the invisible attribute.
3. The subroutine creates a small virtual display and viewport with no banner, the actual icon.
4. The subroutine using `UIS$SHRINK_TO_ICON` associates the icon name with the virtual display identifier of the offscreen viewport.

Expanding Icons to Display Viewports

You can override the default icon expansion operation by calling the AST-enabling routine `UIS$SET_EXPAND_ICON_AST` in your main program. Include `UIS$EXPAND_ICON` in your AST routine in order to specify viewport attributes.

Resizing Display Windows

You can override the default display window resize operation by first calling the AST-enabling routine `UIS$SET_RESIZE_AST` in your main program. Resizing occurs as a three-step process as follows:

1. The user initiates the resizing operation using the user interface.
2. The user interface returns values to the addresses specified in `UIS$SET_RESIZE_AST`.

17-16 Asynchronous System Trap Routines

3. The AST routine is called.

Your AST routine will include a call to `UIS$RESIZE_WINDOW`. A call to `UIS$RESIZE_WINDOW` can redefine the default resize behavior in the following ways:

- Absolute position — You can specify an absolute position, that is, a device coordinate position on the physical screen where the newly resized display viewport will be placed.
- Size — You can specify the dimensions of all newly resized display viewports. All subsequent display viewports are created with these dimensions.
- World coordinate space — You can specify the world coordinate space as the original display window. Typically, the coordinates that you specify here match the world coordinates of the original display window. However, this need not always be the case. If your original display window viewed a portion of the virtual display, you can view more or less of the virtual display depending on the world coordinate range you specify.

Closing Display Windows

To override the default close display window operation, you must first call the AST-enabling routine `UIS$SET_CLOSE_AST` in your main program.

The instructions that you include in your AST routine will override the default window closing behavior. Closing display windows occurs as a two-step process as follows:

1. The “Delete” menu item in the Window Option menu is chosen.
2. The AST routine is called.

17.3.3 Program Development

Programming Objective

To modify the display window shrinking, expanding, resizing, and closing operations listed in the Window Options Menu, whenever the “Shrink to icon”, “Change the size” or “Delete” menu item is chosen.

Programming Tasks

1. Declare the subroutines and the appropriate variables in the `COMMON` statement.
2. Create a virtual display.
3. Create a display window and viewport.
4. Draw two ellipses and a circle.

5. Enable viewport shrinking and icon expansion AST routines using UIS\$SET_SHRINK_TO_ICON_AST and UIS\$SET_EXPAND_ICON_AST.
6. Enable window resizing and closing AST routines using UIS\$SET_RESIZE_AST and UIS\$SET_CLOSE_AST.
7. Create viewport shrinking and icon expansion AST routines.
8. Create window resizing and closing AST routines.

```

PROGRAM OVERRIDE
IMPLICIT INTEGER(A-Z)
EXTERNAL RESIZER, SHRINKER, EXPANDER, CLOSER
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON VD_ID,VD_ID2,WD_ID,WD_ID2,NEW_ABS_X,NEW_ABS_Y

VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,50.0,50.0,10.0,10.0) ①
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','USER') ②

CALL UIS$ELLIPSE(VD_ID,0,10.0,20.0,5.0,15.0)

CALL UIS$SET_FONT(VD_ID,0,1,'UIS$FILL_PATTERNS')
CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_VERT6_2)
CALL UIS$CIRCLE(VD_ID,1,25.0,25.0,20.0)

CALL UIS$ELLIPSE(VD_ID,0,40.0,20.0,5.0,15.0)

CALL UIS$SET_SHRINK_TO_ICON_AST(WD_ID,SHRINKER) ③
CALL UIS$SET_EXPAND_ICON_AST(WD_ID,EXPANDER) ④
CALL UIS$SET_CLOSE_AST(WD_ID,CLOSER,0) ⑤

CALL UIS$SET_RESIZE_AST(VD_ID,WD_ID,RESIZER,0,NEW_ABS_X,NEW_ABS_Y,
2      NEW_WIDTH,NEW_HEIGHT,NEW_WC_X1,NEW_WC_Y1,NEW_WC_X2,
2      NEW_WC_Y2) ⑥

CALL SYS$HIBER() ⑦

TYPE *,'DISPLAY WINDOW HAS BEEN SUCCESSFULLY CLOSED' ⑧

END

SUBROUTINE RESIZER ⑨
IMPLICIT INTEGER(A-Z)
COMMON VD_ID,VD_ID2,WD_ID,wd_id2,NEW_ABS_X,NEW_ABS_Y
CALL UIS$RESIZE_WINDOW(VD_ID,WD_ID,NEW_ABS_X,NEW_ABS_Y,.....) ⑩

RETURN
END

```

17-18 Asynchronous System Trap Routines

```
SUBROUTINE SHRINKER 11
IMPLICIT INTEGER(A-Z)
INCLUDE 'SYS$LIBRARY:UISENTRY'
INCLUDE 'SYS$LIBRARY:UISUSRDEF'
COMMON VD_ID,VD_ID2,WD_ID,WD_ID2,NEW_ABS_X,NEW_ABS_Y

STRUCTURE/AWAY/ 12
INTEGER*4 CODE1
INTEGER*4 ATTR1
INTEGER*4 CODE2
INTEGER*4 ATTR2
INTEGER*4 END_LIST
END STRUCTURE

RECORD/AWAY/WINDOW 13

WINDOW.CODE1=WDPL$C_PLACEMENT 14
WINDOW.ATTR1=WDPL$M_INVISIBLE 15
WINDOW.CODE2=WDPL$C_END_OF_LIST 16
CALL UIS$MOVE_VIEWPORT(WD_ID,WINDOW) 17

WINDOW.CODE1=WDPL$C_ATTRIBUTES 18
WINDOW.ATTR1=WDPL$M_NOBANNER 19
WINDOW.CODE2=WDPL$C_END_OF_LIST 20

VD_ID2=UIS$CREATE_DISPLAY(0.0,0.0,5.0,5.0,2.54,2.54) 21
WD_ID2=UIS$CREATE_WINDOW(VD_ID2,'SYS$WORKSTATION',, 22
2,,,,,WINDOW)
CALL UIS$SET_FONT(VD_ID2,0,2,'MY_FONT_5')
CALL UIS$TEXT(VD_ID2,2,'USER',0.5,3.5) 23

ICON_FLAGS=UIS$M_ICON_DEF_BODY 24
CALL UIS$SHRINK_TO_ICON(WD_ID,WD_ID2,ICON_FLAGS) 25

RETURN
END

SUBROUTINE EXPANDER
IMPLICIT INTEGER(A-Z)
COMMON VD_ID,VD_ID2,WD_ID,WD_ID2,NEW_ABS_X,NEW_ABS_Y

CALL UIS$EXPAND_ICON(WD_ID,WD_ID2)

RETURN
END

SUBROUTINE CLOSER 26
IMPLICIT INTEGER(A-Z)
COMMON VD_ID,VD_ID2,WD_ID,WD_ID2,NEW_ABS_X,NEW_ABS_Y

CALL UIS$ERASE(VD_ID)
CALL UIS$DELETE_WINDOW(WD_ID) 27
```

```

CALL UIS$DELETE_DISPLAY(VD_ID)           28
CALL SYS$WAKE(,)                         29
RETURN
END

```

The main program `OVERRIDE` creates a virtual display ❶ and a display window ❷. The world coordinate space of the display window is a portion of the virtual display, the display window contains only those objects in the virtual display that lie within it.

A circle is drawn between two ellipses in the virtual display and appears in the display window and its associated display viewport.

Four AST-enabling routines, `UIS$SET_SHRINK_TO_ICON_AST`, `UIS$SET_EXPAND_ICON_AST`, `UIS$SET_CLOSE_AST` and `UIS$SET_RESIZE_AST`, ❸ ❹ ❺ ❻ are called. The main program executes until the call to `SYS$HIBER` is reached ❼.

The Window Options Menu is invoked from the `MENU` icon in the viewport `WINDOW` using the pointer. Assume that the menu item "Change the size" is chosen. Perform the following procedure:

1. Move the pointer to one of the flashing dots on the border of the viewport.
2. Press the button and the border of the display viewport is highlighted.
3. Hold the button down and move the pointer until the stretchy box is the desired size and release the pointer button.

The call to `UIS$RESIZE_WINDOW` ❽ in the subroutine `RESIZER` ❾ modifies the default resize behavior. `UIS$RESIZE_WINDOW` specifies the world coordinates of the existing virtual display as the world coordinates for all newly resized display windows. Therefore, a newly resized window always displays the entire virtual display space. If the aspect ratios of the virtual display and the resized display viewport are not equal, graphic objects are scaled.

The subroutine `SHRINKER` ❿ modifies the default shrinking behavior. The window attributes data structure `AWAY` is created ❿. A record `WINDOW` is defined to have the structure of `AWAY` ⓫. The fields of record `WINDOW` are assigned values ⓬ ⓭ ⓮. Note the use of the invisible placement attribute. A call to `UIS$MOVE_VIEWPORT` ⓯ references the display window identifier of the existing viewport and the current window attributes. The viewport is moved offscreen.

New window attribute values are assigned ⓰ ⓱ ⓲ to the fields of the record `WINDOW`.

17-20 Asynchronous System Trap Routines

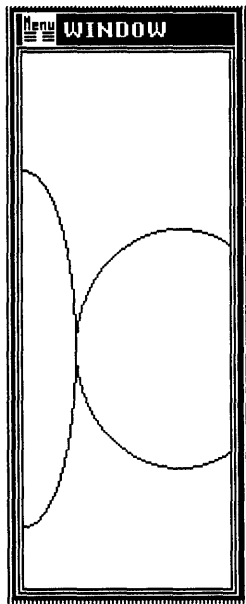
A virtual display and display window are created for the icon 21 22. UIS\$TEXT draws the character string in the icon 23. The flag UIS\$M_ICON_DEF_BODY sets the appropriate in the mask **icon_flags** 24. When this bit is set, the area of the icon becomes a button AST region (for later icon expansion). UIS\$SHRINK_TO_ICON 25 associates the display window identifiers of the existing viewport and the icon.

The subroutine CLOSER 26 overrides the default window closing behavior by deleting the display window 27, display viewport, and the virtual display 28. The process that owns the main program is awakened 29. The main program continues execution with the next statement after the call to SYS\$HIBER 8, types the message "Display window has been successfully closed", and terminates.

17.3.4 Calling UIS\$SET_RESIZE_AST

When the main program executes, a display window and its associated display viewport appear on the display screen as shown in Figure 17-4.

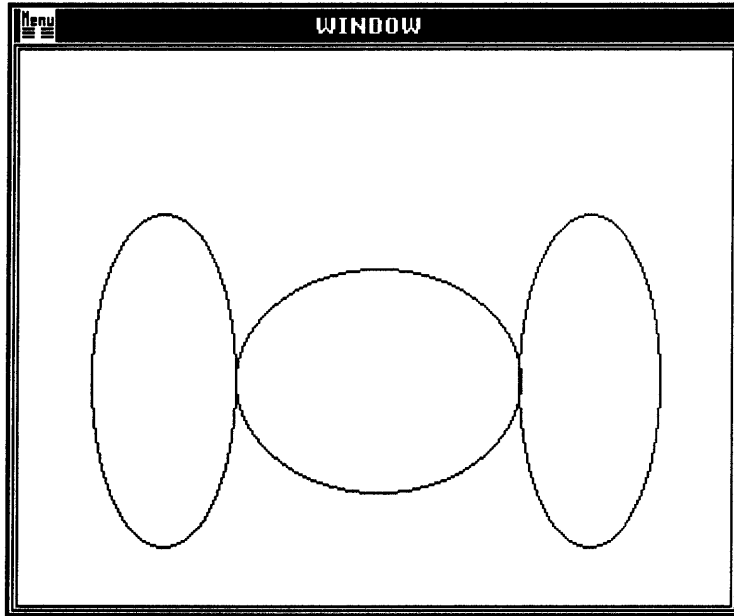
Figure 17-4 Unresized Window and Viewport



ZK-4563-85

The menu item "Change the size" is selected and the display window and viewport are resized as shown in Figure 17-5.

Figure 17-5 Resized Window and Viewport



ZK-4564-85

17.3.5 Calling UIS\$SET_SHRINK_TO_ICON_LAST

The menu item "Shrink to icon" is selected and the display viewport is replaced with a user-defined icon as shown in Figure 17-6.

Figure 17-6 Icon



ZK-5484-86

17-22 Asynchronous System Trap Routines

17.3.6 Calling `UIS$SET_CLOSE_AST`

When the menu item "Delete" is chosen, the display viewport, window, and virtual display are deleted and the message "Display window has been successfully closed" is written to the terminal emulation window.

PART III UIS Routines

Chapter 18

UIS Routine Descriptions

18.1 Overview

Each UIS and UISDC routine in Parts III and IV of this book is documented using a structured format. This section discusses the main headings of this format, the information that is presented under each heading, and the format used to present the information.

The purpose of this section, therefore, is to explain where to find information and how to read it correctly, not how to use it.

Some main headings in the routine template contain information that requires no further explanation beyond what is given in Table 18–1. However, the following main headings contain information that does require additional discussion; this discussion takes place in the remaining subsections of this section.

- Format Heading
- Returns Heading
- Arguments Heading

The following table lists the main headings in the UIS routines template.

Table 18–1 Main Headings in the Routine Template

Main Heading	Description
Routine Name	Required. The routine entry point name is usually, though not always, followed by the English name of the routine.
Routine Overview	Required. The routine overview appears directly below the routine name; the overview explains, usually in one or two sentences, what the routine does.

18-2 UIS Routine Descriptions

Table 18-1 (Cont.) Main Headings in the Routine Template

Main Heading	Description
Format	Required. The format heading follows the routine overview. The format gives the routine entry point name and the routine argument list.
Returns	Required. The returns heading follows the routine format. It explains what information is returned by the routine.
Arguments	Required. The arguments heading follows the returns heading. Detailed information about each argument is provided under the arguments heading. If a routine takes no arguments, the word "None" appears.
Description	<p>Optional. The description heading follows the arguments heading. The description section contains information about specific actions taken by the routine: interaction between routine arguments, if any; operation of the routine within the context of VAX/VMS; user privileges needed to call the routine, if any; system resources used by the routine; and user quotas that may affect the operation of the routine.</p> <p>Note that any restrictions on the use of the routine are always discussed first in the description section; for example, any required user privileges or necessary system resources are explained first.</p> <p>For some simple routines, a description section is not necessary because the routine overview carries the needed information.</p>
Examples	<p>Optional. The examples heading appears following the description heading. The examples section contains programming examples that illustrate use of the routine. Following the example, an explanation of the example is given.</p> <p>All examples have been tested and should run when compiled (or assembled) and linked.</p>

Table 18-1 (Cont.) Main Headings in the Routine Template

Main Heading	Description
Screen Output	Optional. The screen output heading contains either an actual display produced by the routine or information that the routine would normally return to the program. Please note that in many instances screen output contains annotations that serve only to explain the information returned. For example, UIS\$GET_POSITION returns information about the current text position along the actual path. This information is displayed and described as an example of the kind of data that can be returned. In many cases such as the inquiry routines, the displayed information is formatted with headings and annotations for the purposes of presentation in this manual only.
Illustration	Optional. The illustration heading contains artwork that describes how to use the routine, how the routine functions, or what kind of information to expect from it. The illustrations <i>may or may not</i> be annotated.

18.1.1 Format Heading

The following types of information can be present in the format heading.

- Procedure call format
- Explanatory text

The procedure call format ensures that a routine call conforms to the procedure call mechanism described in the VAX Procedure Calling and Condition Handling Standard; for example, an entry mask is created, registers are saved, and so on.

Procedure call formats can appear in many forms. Four examples have been provided to illustrate the meaning of syntactical elements such as brackets and commas. General rules of syntax governing how to use procedure call formats are shown in Table 18-2.

Example 1 This example illustrates the standard representation of optional arguments and best describes the use of commas as delimiters. Arguments enclosed within square brackets are optional, but if an optional argument other than a trailing optional argument is omitted, you must include a comma as a delimiter for the omitted argument.

```
ENTRY-POINT-NAME  arg1 [, [arg2 [,arg3]]
```

Typically, VAX RMS system routines use this format where at most three arguments appear in the argument list.

18-4 UIS Routine Descriptions

Example 2 When the argument list contains three or more optional arguments, the syntax does not provide enough information. If the optional arguments **arg3** and **arg4** are omitted and the trailing argument **arg5** is specified, commas **must** be used to delimit the positions of the omitted arguments.

```
ENTRY-POINT-NAME    arg1 ,arg2 ,[arg3] ,nullarg [,arg4] [,arg5]
```

Typically, VAX/VMS system services, utility routines, and VAX Run-Time Library routines contain call formats with more than three arguments.

Example 3 In the following call format example, the trailing four arguments are optional as a group, that is, either you specify **arg2**, **arg3**, **arg4**, and **arg5** or none of them. Therefore, if the optional arguments are not specified, commas need not be used to delimit unoccupied positions.

However, if a hypothetical required argument or a separate optional argument were specified after **arg5**, commas must be used when **arg2**, **arg3**, **arg4**, and **arg5** are omitted.

```
ENTRY-POINT-NAME    arg1 [,arg2 ,arg3 ,arg4 ,arg5]
```

Example 4 In the following example, you may specify **arg2** and omit **arg3**. However whenever you specify **arg3**, you **must** specify **arg2**

```
ENTRY-POINT-NAME    arg1 [,arg2 [,arg3]]
```

Explanatory Text

Explanatory text may follow one or both of the above formats. This text is present only when needed to clarify the format. For example, the call format indicates that arguments are optional by enclosing them in brackets ([]). However, brackets alone cannot convey all the important information that may apply to optional arguments. For example, in some routines that have many optional arguments, if one optional argument is selected, another optional argument must also be selected. In such cases, text following the format clarifies this fact.

Table 18-2 General Rules of Syntax

Element	Syntax Rule
Entry point names	Entry point names are always shown in uppercase characters.
Argument names	Argument names are always shown in lowercase characters.
Spaces	One or more spaces are used between the entry point name and the first argument, and between each argument.
Braces	Braces surround two or more arguments. You must choose one of the arguments.
Brackets ([])	Brackets surround optional arguments. Note that commas too can be optional (see the comma element).

Table 18-2 (Cont.) General Rules of Syntax

Element	Syntax Rule
Commas	Between arguments, the comma always follows the space. If the argument is optional, the comma may appear inside the brackets or outside the brackets, depending on the position of the argument in the list and on whether surrounding arguments are optional or required.
Null arguments	<p>A null argument is a place-holding argument. It is used for either of the following reasons: (1) to hold a place in the argument list for an argument that has not yet been implemented by DIGITAL but may be in the future or (2) to mark the position of an argument that was used in earlier versions of the routine but is not used in the latest version (upward compatibility is thereby ensured because arguments that follow the null argument in the argument list keep their original positions). A null argument is always given the name nullarg.</p> <p>In the argument list constructed on the stack when a procedure is called, both null arguments and omitted optional arguments are represented by longword argument list entries containing the value 0. The programming language syntax required to produce argument list entries containing 0 differ from language to language, so see your language user's guide for language-specific syntax.</p>

18.1.2 Returns Heading

Under the returns heading appears information that describes what information, if any, is returned by the routine to the caller. Programs written in VAX MACRO return information in R0. The information that is returned is a longword value.

The high-level language programmer receives status information in the return (or status) variable he or she uses when making the call. The run-time environment established for the high-level language program allows the status information in R0 to be moved automatically to the user's return variable. The information that is returned is always a longword value.

18-6 UIS Routine Descriptions

18.1.3 Arguments Heading

Under the arguments heading appears detailed information about each argument listed in the call format. Arguments are described in the order in which they appear in the call format. If the routine has no arguments, the term “none” appears.

The following format is used to describe each argument.

argument-name

VMS Usage: argument-VMS-data-type

type: argument-data-type

access: argument-access

mechanism: argument-passing-mechanism

One paragraph of structured text, followed by other paragraphs of text, as needed.

18.2 Functional Organization of UIS Routines

The UIS routines perform many functions within an application program. Besides, creating the graphic objects that you see on the display screen, there are routines that manage the input devices and routines that return information to the program to name a few.

Figure 18-1 lists each UIS routine by functional category.

Figure 18-1 Functional Categories of UIS Routines

AST-Enabling Routines

UIS\$SET_ADDOPT_AST
 UIS\$SET_BUTTON_AST
 UIS\$SET_CLOSE_AST
 UIS\$SET_EXPAND_ICON_AST
 UIS\$SET_GAIN_KB_AST
 UIS\$SET_KB_AST
 UIS\$SET_LOSE_KB_AST
 UIS\$SET_MOVE_INFO_AST
 UIS\$SET_POINTER_AST
 UIS\$SET_RESIZE_AST
 UIS\$SET_SHRINK_TO_ICON_AST
 UIS\$SET_TB_AST

Attribute Routines

UIS\$SET_ARC_TYPE
 UIS\$SET_BACKGROUND_INDEX
 UIS\$SET_CHAR_ROTATION
 UIS\$SET_CHAR_SIZE
 UIS\$SET_CHAR_SLANT
 UIS\$SET_CHAR_SPACING
 UIS\$SET_CLIP
 UIS\$SET_FILL_PATTERN
 UIS\$SET_FONT
 UIS\$SET_LINE_STYLE
 UIS\$SET_LINE_WIDTH
 UIS\$SET_TEXT_FORMATTING
 UIS\$SET_TEXT_MARGINS
 UIS\$SET_TEXT_PATH
 UIS\$SET_TEXT_SLOPE
 UIS\$SET_WRITING_INDEX
 UIS\$SET_WRITING_MODE

Color Routines

UIS\$CREATE_COLOR_MAP
 UIS\$CREATE_COLOR_MAP_SEG
 UIS\$DELETE_COLOR_MAP
 UIS\$DELETE_COLOR_MAP_SEG
 UIS\$HLS_TO_RGB
 UIS\$HSV_TO_RGB
 UIS\$RESTORE_CMS_COLORS
 UIS\$RGB_TO_HLS
 UIS\$RGB_TO_HSV
 UIS\$SET_COLOR
 UIS\$SET_COLORS
 UIS\$SET_INTENSITIES
 UIS\$SET_INTENSITY

Display List Routines

UIS\$BEGIN_SEGMENT
 UIS\$COPY_OBJECT
 UIS\$DELETE_OBJECT
 UIS\$DELETE_PRIVATE
 UIS\$DISABLE_DISPLAY_LIST
 UIS\$ENABLE_DISPLAY_LIST
 UIS\$END_SEGMENT
 UIS\$ERASE
 UIS\$EXECUTE
 UIS\$EXECUTE_DISPLAY
 UIS\$EXTRACT_HEADER
 UIS\$EXTRACT_OBJECT
 UIS\$EXTRACT_PRIVATE
 UIS\$EXTRACT_REGION
 UIS\$EXTRACT_TRAILER
 UIS\$FIND_PRIMITIVE
 UIS\$FIND_SEGMENT
 UIS\$INSERT_OBJECT
 UIS\$MOVE_AREA
 UIS\$PRIVATE
 UIS\$SET_INSERTION_POSITION
 UIS\$TRANSFORM_OBJECT

Graphics Routines

UIS\$CIRCLE
 UIS\$ELLIPSE
 UIS\$IMAGE
 UIS\$LINE
 UIS\$LINE_ARRAY
 UIS\$PLOT
 UIS\$PLOT_ARRAY

Inquiry Routines

UIS\$GET_ABS_POINTER_POS
 UIS\$GET_ALIGNED_POSITION
 UIS\$GET_ARC_TYPE
 UIS\$GET_BACKGROUND_INDEX
 UIS\$GET_BUTTONS
 UIS\$GET_CHAR_ROTATION
 UIS\$GET_CHAR_SIZE
 UIS\$GET_CHAR_SLANT
 UIS\$GET_CHAR_SPACING
 UIS\$GET_CLIP
 UIS\$GET_COLOR
 UIS\$GET_COLORS
 UIS\$GET_CURRENT_OBJECT

18-8 UIS Routine Descriptions

Figure 18-1 (Cont.) Functional Categories of UIS Routines

Inquiry Routines (cont.)

UIS\$GET_DISPLAY_SIZE
UIS\$GET_FILL_PATTERN
UIS\$GET_FONT
UIS\$GET_FONT_ATTRIBUTES
UIS\$GET_FONT_SIZE
UIS\$GET_HW_COLOR_INFO
UIS\$GET_INTENSITIES
UIS\$GET_INTENSITY
UIS\$GET_KB_ATTRIBUTES
UIS\$GET_LINE_STYLE
UIS\$GET_LINE_WIDTH
UIS\$GET_NEXT_OBJECT
UIS\$GET_OBJECT_ATTRIBUTES
UIS\$GET_PARENT_SEGMENT
UIS\$GET_POINTER_POSITION
UIS\$GET_POSITION
UIS\$GET_PREVIOUS_OBJECT
UIS\$GET_ROOT_SEGMENT
UIS\$GET_TB_INFO
UIS\$GET_TB_POSITION
UIS\$GET_TEXT_FORMATTING
UIS\$GET_TEXT_MARGINS
UIS\$GET_TEXT_PATH
UIS\$GET_TEXT_SLOPE
UIS\$GET_VCM_ID
UIS\$GET_VIEWPORT_ICON
UIS\$GET_VIEWPORT_POSITION
UIS\$GET_VIEWPORT_SIZE
UIS\$GET_VISIBILITY
UIS\$GET_WINDOW_ATTRIBUTES
UIS\$GET_WINDOW_SIZE
UIS\$GET_WRITING_INDEX
UIS\$GET_WRITING_MODE
UIS\$GET_WS_COLOR
UIS\$GET_WS_INTENSITY

Keyboard Routines

UIS\$CREATE_KB
UIS\$DELETE_KB
UIS\$DISABLE_KB
UIS\$DISABLE_VIEWPORT_KB
UIS\$ENABLE_KB
UIS\$ENABLE_VIEWPORT_KB

Keyboard Routines (cont.)

UIS\$READ_CHAR
UIS\$SET_KB_ATTRIBUTES
UIS\$SET_KB_COMPOSE2
UIS\$SET_KB_COMPOSE3
UIS\$SET_KB_KEYTABLE
UIS\$TEST_KB

Pointer Routines

UIS\$CREATE_TB
UIS\$DELETE_TB
UIS\$DISABLE_TB
UIS\$ENABLE_TB
UIS\$SET_POINTER_PATTERN
UIS\$SET_POINTER_POSITION

Sound Routines

UIS\$SOUND_BELL
UIS\$SOUND_CLICK

Text Routines

UIS\$MEASURE_TEXT
UIS\$NEW_TEXT_LINE
UIS\$SET_ALIGNED_POSITION
UIS\$SET_POSITION
UIS\$TEXT

Windowing Routines

UIS\$CLOSE_WINDOW
UIS\$CREATE_DISPLAY
UIS\$CREATE_TERMINAL
UIS\$CREATE_TRANSFORMATION
UIS\$CREATE_WINDOW
UIS\$DELETE_DISPLAY
UIS\$DELETE_TRANSFORMATION
UIS\$DELETE_WINDOW
UIS\$EXPAND_ICON
UIS\$MOVE_VIEWPORT
UIS\$MOVE_WINDOW
UIS\$POP_VIEWPORT
UIS\$PUSH_VIEWPORT
UIS\$RESIZE_WINDOW
UIS\$SHRINK_TO_ICON

UIS\$BEGIN_SEGMENT

Begins a new segment in the virtual display.

Format

seg_id=UIS\$BEGIN_SEGMENT *vd_id*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the segment identifier in the variable *seg_id* or R0 (VAX MACRO). The segment identifier uniquely identifies a segment and is used as an argument in other routines.

UIS\$BEGIN_SEGMENT signals all errors; no condition values are returned.

Argument

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

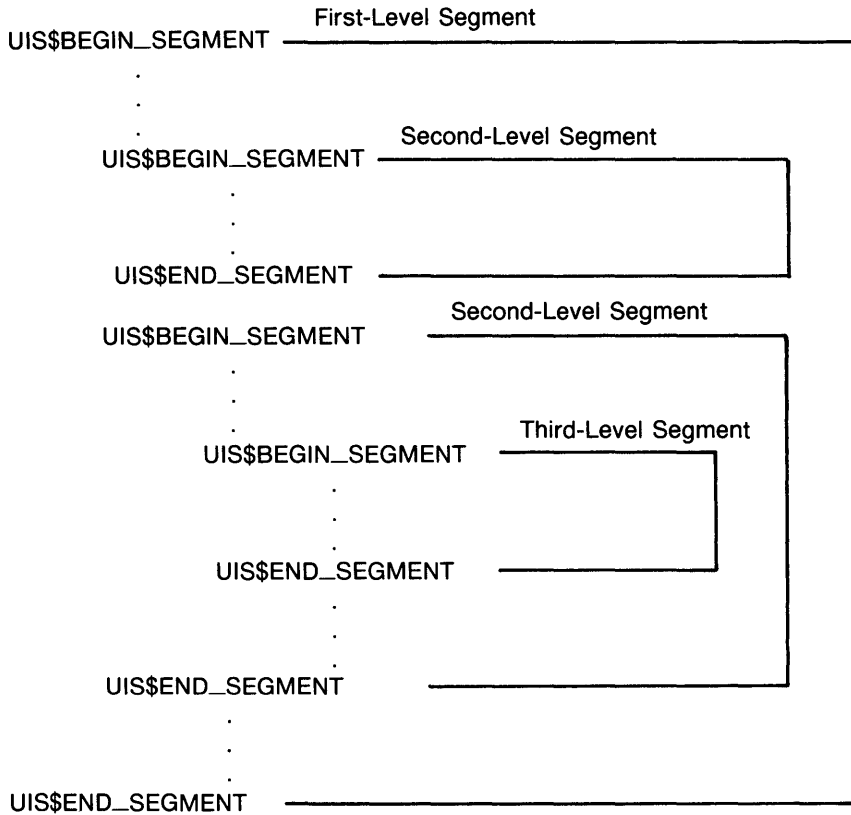
Description

All values of attribute blocks 0 to 255 are propagated to the new segment, but all changes to attribute blocks in this segment will be local to this segment only and not the parent.

You can also nest segments.

18-10 **UIS Routine Descriptions**
UIS\$BEGIN_SEGMENT

Illustration



UIS\$CIRCLE

Draws an arc along the circumference of a circle.

Format

UIS\$CIRCLE *vd_id, atb, center_x, center_y, xradius*
[,start_deg ,end_deg]

Returns

UIS\$CIRCLE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

atb

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Attribute block number. The ***atb*** argument is the address of a longword integer that specifies an attribute block that controls the appearance of the circle or arc.

center_x, center_y

VMS Usage: **floating_point**
 type: **f_floating**
 access: **read only**
 mechanism: **by reference**

Center position x and y world coordinates. The ***center_x*** and ***center_y*** arguments are the addresses of f_floating point numbers that define a point in the virtual display that is the center of the arc or circle.

18-12 UIS Routine Descriptions

UIS\$CIRCLE

xradius

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Radius of the circle specified as an x world coordinate width. The **xradius** argument is the address of an **f_floating** point number that defines the distance from the center of the circle to the circumference of the circle.

start_deg, end_deg

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Degree at which the arc starts and ends. The **start_deg** and **end_deg** arguments are the addresses of **f_floating** point numbers that define the starting and ending point on the circumference of the circle where the arc or circle will be drawn. Degrees are measured clockwise from the top of the circle. If these arguments are not specified, *0.0* degrees and *360.0* degrees are assumed, respectively.

Description

UIS\$CIRCLE draws an arc specified by a center position and a radius for the range of the degrees specified.

The arc can be closed by drawing one or more lines between the endpoints. The arc type associated with the attribute block specifies the way in which the arc is closed. The arc is not closed off by default. See UIS\$SET_ARC_TYPE for details.

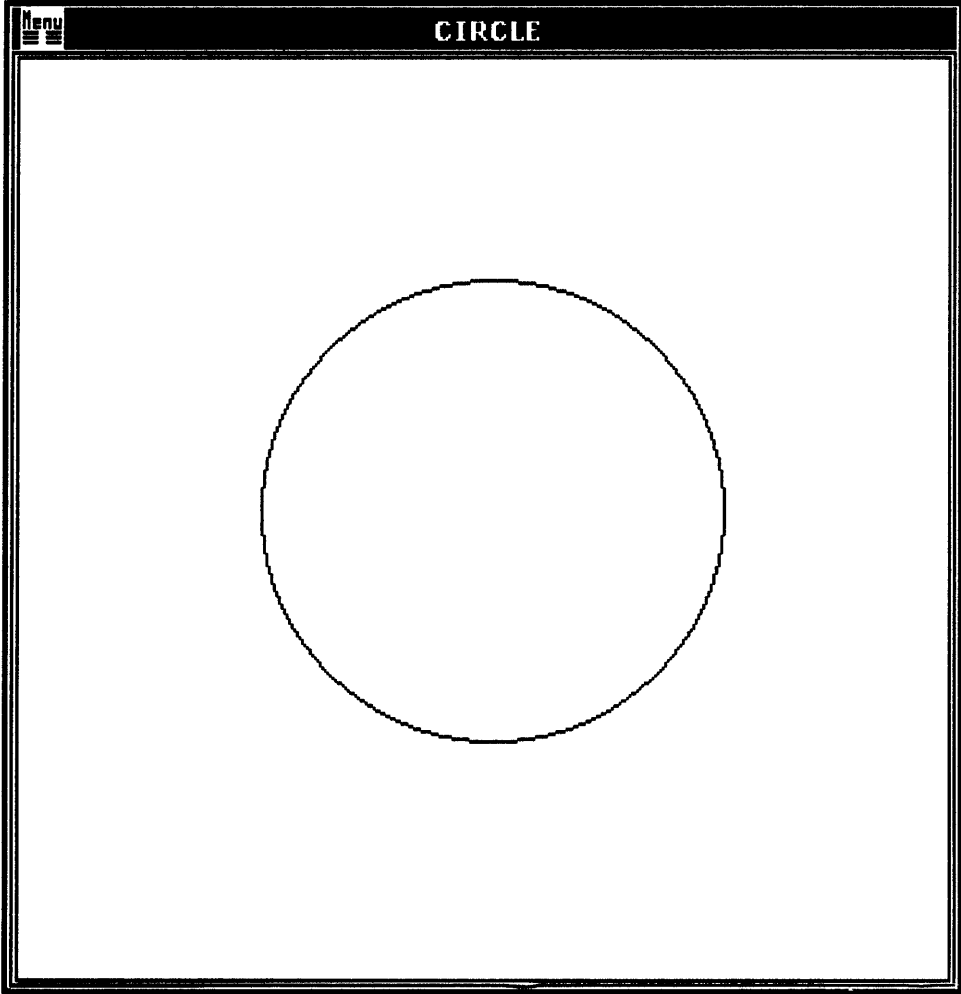
The points are drawn with the current line pattern and width, and filled with the current fill pattern if enabled.

UIS\$CIRCLE does not support the following combination of attributes:

- Line width not equal to 1 and line style not equal to $FFFFFFFF_{16}$
- Line width not equal to 1 and complement writing mode

Circles are distorted by differences between the aspect ratios of the display window and display viewport.

Screen Output



18-14 **UIS Routine Descriptions**
UIS\$CLOSE_WINDOW

UIS\$CLOSE_WINDOW

Calls the system service SYS\$EXIT to exit the current image.

Format

UIS\$CLOSE_WINDOW *wd_id*

Returns

UIS\$CLOSE_WINDOW signals all errors; no condition values are returned.

Argument

wd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

Description

UIS\$CLOSE_WINDOW is invoked as the default action taken by the "Delete" menu item in the Window Options Menu. See UIS\$SET_CLOSE_AST for information about overriding this routine.

UIS\$COPY_OBJECT

Copies the specified object and its private data within the virtual display. It may also transform the coordinates or attributes or both of the specified object. The original object remains unchanged in the virtual display.

Format

copy_id=UIS\$COPY_OBJECT { *obj_id*
seg_id } [,*matrix*] [,*atb*]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the copy identifier in the variable *copy_id* or R0 (VAX MACRO). The copy identifier uniquely identifies a newly copied object.

UIS\$COPY_OBJECT signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies an object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

18-16 UIS Routine Descriptions UIS\$COPY_OBJECT

matrix

VMS Usage: **vector_longword_signed**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Transformation matrix. The **matrix** argument is the address of a 2 x 3 matrix of longwords containing scaling, translation, and/or rotation data.

Structure of a VAX FORTRAN Two-Dimensional Array

A two-dimensional array declared as ARRAY(2,3) has the following structure.

1,1	1,2	1,3
2,1	2,2	2,3

ZK-5492-86

Different languages allocate memory for array elements in different orders. This description assumes the order used by VAX FORTRAN. If you call UIS\$COPY_OBJECT from another language, make sure that the array elements are in the same order.

Memory addresses of array elements range from lowest to highest in the following order: (1,1),(2,1), (1,2),(2,2),(1,3), and (2,3). The order of array element is shown in the following figure.

1	3	5
2	4	6

ZK-5493-86

Pairs of array elements govern how displayed objects are scaled, rotated, and translated. UIS computes the transformed coordinates in the following manner.

$$x_1 = A(1,1)*x + A(1,2)*y + A(1,3)$$

$$y_1 = A(2,1)*x + A(2,2)*y + A(2,3)$$

Translation

When translation alone is performed, the following array elements are assigned values. D_x and D_y represent distances between the original coordinates and the new coordinates.

1	0	D_x
0	1	D_y

ZK-5494-86

Scaling

When scaling alone is performed, the following array elements are assigned values.

S_x	0	0
0	S_y	0

ZK-5495-86

Rotation

When rotation alone is performed, the following array elements are assigned values, where "@" is the desired angle of rotation measured clockwise. The values returned from the VAX FORTRAN SIN and COS functions are stored in the appropriate array elements.

$\cos (@)$	$\sin (@)$	0
$-\sin (@)$	$\cos (@)$	0

ZK-5496-86

An unlimited number of transformations can be performed at one time by simply multiplying the matrices together into a single matrix using matrix multiplication.

18-18 UIS Routine Descriptions

UIS\$COPY_OBJECT

In order to multiply two matrices together, you must add a row to the bottom of each matrix.

0	0	1
---	---	---

ZK-5461-86

After the multiplication is performed, remove the last row of the result.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

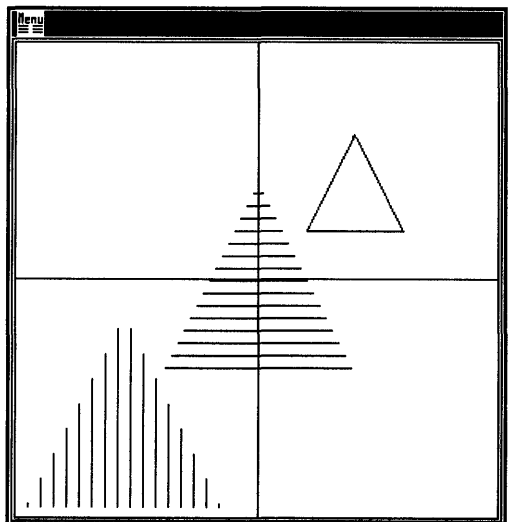
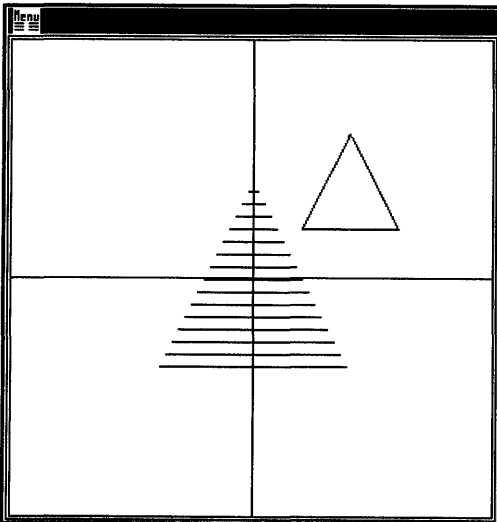
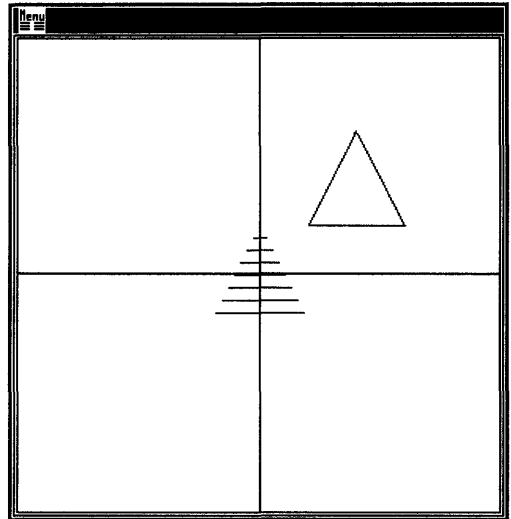
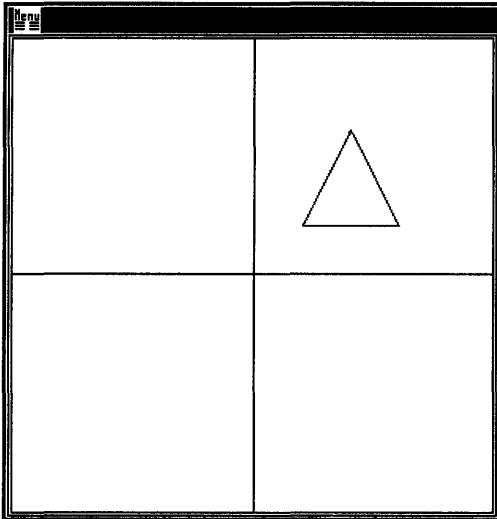
Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block whose attribute settings override current segment attributes.

Description

Either the coordinates can be transformed, or the attributes can be overridden or both.

After a transformation, occluded objects may not appear correctly on the display screen. This can be corrected by calling UIS\$EXECUTE to correctly refresh the display screen.

Screen Output



UIS\$CREATE_COLOR_MAP

Creates a virtual color map of the specified size and with the specified attributes.

Format

vcm_id=**UIS\$CREATE_COLOR_MAP** *vcm_size*
 [*,vcm_name*]
 [*,vcm_attributes*]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the virtual color map identifier in the variable *vcm_id* or R0 (VAX MACRO). The virtual color map identifier uniquely identifies the virtual color map and must be specified in **UIS\$CREATE_DISPLAY**. It is also used as an argument in other color routines.

UIS\$CREATE_COLOR_MAP signals all errors; no condition values are returned.

Arguments

vcm_size
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Size of the virtual color map. The ***vcm_size*** argument is the address of a longword that defines the number of entries in the virtual color map.

vcm_name
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Name of the virtual color map. The **vcm_name** argument is the address of a string descriptor of the name of the virtual color map. Specify the name of an existing shareable color map. If your application is creating the shareable color map, specify a valid color map name.

The virtual color map name should not exceed 15 characters.

vcm_attributes

VMS Usage: **item_list_pair**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by descriptor**

Virtual color map attributes. The **vcm_attributes** argument is the address of data structure of longword pairs that specify virtual color attributes.

The following figure describes the structure of this argument.

Attribute code (VCMAL\$C_XXXX)
Longword value for attribute specified in previous longword
2nd attribute code
2nd attribute value
• • •
End of list = 0 (VCMAL\$C_END_OF_LIST)

ZK-5367-86

All of the following virtual color map attributes are optional.

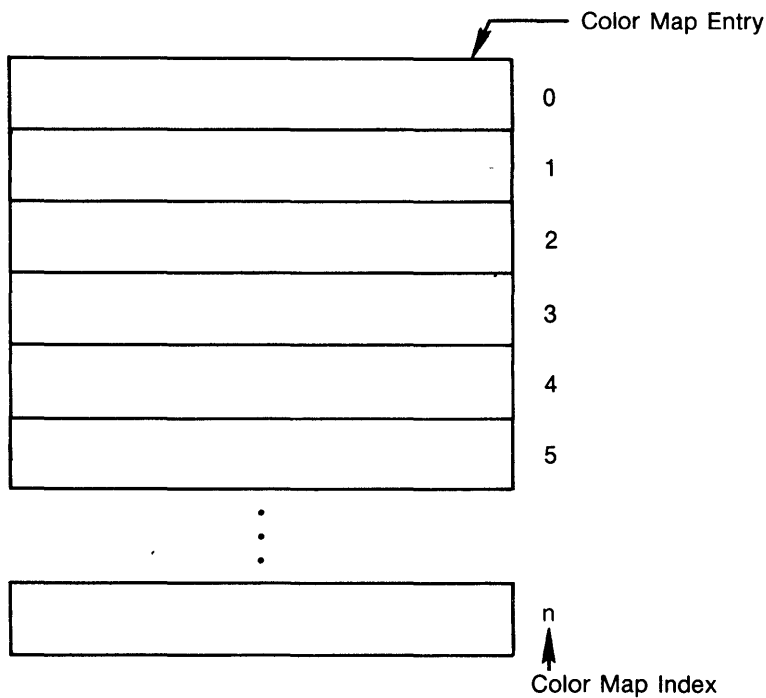
18-22 UIS Routine Descriptions
UIS\$CREATE_COLOR_MAP

Attributes	Function
VCMAL\$C_ATTRIBUTES	General attributes
VCMAL\$M_RESIDENT	Set for resident virtual color map
VCMAL\$M_SHARE	Set for shareable virtual color map
VCMAL\$M_SYSTEM ^{1,2}	Set for system shareable virtual color map
VCMAL\$M_NO_BIND	Set to disable automatic hardware color map binding

¹VCMAL\$M_SHARE must also be set.

²SYSGBL privilege is required.

Illustration



UIS\$CREATE_COLOR_MAP_SEG

Allocates one or more hardware color map indices and binds them to a virtual color map.

Format

```
cms_id=UIS$CREATE_COLOR_MAP_SEG vcm_id  
                                     [,devnam]  
                                     [,place_mode]  
                                     [,place_data]
```

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the color map segment identifier in the variable *cms_id* or R0 (VAX MACRO). The color map segment identifier uniquely identifies the color map segment and is used as an argument in other routines.

UIS\$CREATE_COLOR_MAP_SEG signals all errors; no condition values are returned.

Arguments

vcm_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual color map identifier. The ***vcm_id*** argument is the address of a longword that uniquely identifies the virtual color map. See UIS\$CREATE_COLOR_MAP for more information about the ***vcm_id*** argument.

NOTE: This routine can only be used **once** for each virtual color map identifier.

18-24 UIS Routine Descriptions

UIS\$CREATE_COLOR_MAP_SEG

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify the device name SYS\$WORKSTATION in the **devnam** argument.

place_mode

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Placement mode. The **place_mode** argument is the address of a longword that specifies the placement mode, that is, which hardware color map entries can be allocated. The following table lists valid placement modes.

Symbol	Function
UIS\$C_GENERAL	General placement—Allocates any available entries in the hardware color map.
UIS\$C_COLOR_EXACT	Exact placement—Allocates map entries starting at the specified entry and aligned on a natural entry boundary. Given the size of the virtual color map, UIS computes a working size that is the smallest power of 2 greater than or equal to the requested size. The natural alignment of a map is a starting index that is a multiple of the working size. For example, a six-entry color map could be placed at indices 0, 8, 16 and so on.
UIS\$C_COLOR_BASED	Based placement (default)—Allocates entries such that writing modes using Boolean logic operations on pixel values can correctly display color intersections.

place_data

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Placement data. The **place_data** argument is the address of a longword that contains the first index to be allocated. The **placement_data** argument is used with exact placement mode.

Description

For hardware supporting bit plane write masks, the segment will be based at an index that is a power of 2 and that write operation will be performed using the appropriate mask. The virtual color map entry index specified in the **place_data** argument indicates the binding between the virtual color map and the hardware color map entries allocated by UIS\$CREATE_COLOR_MAP_SEG. The default value is 0, that is, the first allocated map entry is bound to virtual color map entry 0, the second allocated map entry is bound to virtual color map entry 1, and so on.

If the appropriate entries cannot be allocated, an error is signaled. In addition to failure due to resource depletion, a call to UIS\$CREATE_COLOR_MAP_SEG can fail because UIS has already issued this call for the application. This occurs if the flag VCMAL\$M_NO_BIND is not set when the virtual color map is created, and internal processing required a binding to hardware resources. For example, UIS\$CREATE_WINDOW allocates and binds hardware color map resources when creating a display viewport.

Conversely, if VCMAL\$M_NO_BIND is set, but UIS\$CREATE_COLOR_MAP_SEG was not called, calls to some UIS routines such as UIS\$SET_COLOR and UIS\$SET_INTENSITY may fail.

NOTE: The recommended procedure for using this routine is as follows:

1. Specify the flag VCMAL\$M_NO_BIND when creating the virtual color map with UIS\$CREATE_COLOR_MAP.
2. Invoke UIS\$CREATE_COLOR_MAP_SEG before calling any other UIS routine.
3. Initialize the color map using UIS\$SET_COLORS. By definition all colors are black.

Width and height of the display viewport. The **width** and **height** arguments are the addresses of f-floating point numbers that define both the width and height of the display viewport in centimeters.

vcm_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual color map identifier. The **vcm_id** argument is the address of a longword that uniquely identifies the virtual color map. See UIS\$CREATE_COLOR_MAP for more information about the **vcm_id** argument.

If **vcm_id** is not specified, a two-entry virtual color map is created for the virtual display by default.

Description

To avoid distorting the resulting graphic image, the aspect ratio of the world coordinate range of the display window must be equal to the aspect ratio of the display viewport. See UIS\$CREATE_WINDOW for more information about aspect ratios.

UIS\$CREATE_KB

Creates a virtual keyboard on the specified device.

Format

kb_id=**UIS\$CREATE_KB** *devnam*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the virtual keyboard identifier in the variable *kb_id* or R0 (VAX MACRO). The virtual keyboard identifier uniquely identifies the virtual keyboard. The variable *kb_id* is used as an argument in other routines.

UIS\$CREATE_KB signals all errors; no condition values are returned.

Argument

devnam
VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name string. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify the logical name SYS\$WORKSTATION as the device name string.

Description

UIS\$CREATE_KB generates a value for the **kb_id** argument which is referenced in subsequent routines that use **kb_id** as a parameter.

Example

```

.
.
.
VD_ID=UIS$CREATE_DISPLAY(-5.0,-5.0,50.0,45.0,15.0,15.0)
KB_ID=UIS$CREATE_KB('SYS$WORKSTATION')    ❶
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','VIEWPORT TITLE',
2     10.0,10.0,25.0,25.0)
CALL UIS$ENABLE_VIEWPORT_KB(KB_ID,WD_ID)    ❷
.
.
.
CALL UIS$DISABLE_VIEWPORT_KB(WD_ID)    ❸

```

The preceding example creates a virtual keyboard ❶ and binds the virtual keyboard to a display window ❷. In order to use the virtual keyboard and its characteristics with the desired viewport, you must assign the physical keyboard to the desired virtual keyboard and viewport. Press the F5 or CYCLE key until the KB icon in the appropriate viewport is highlighted.

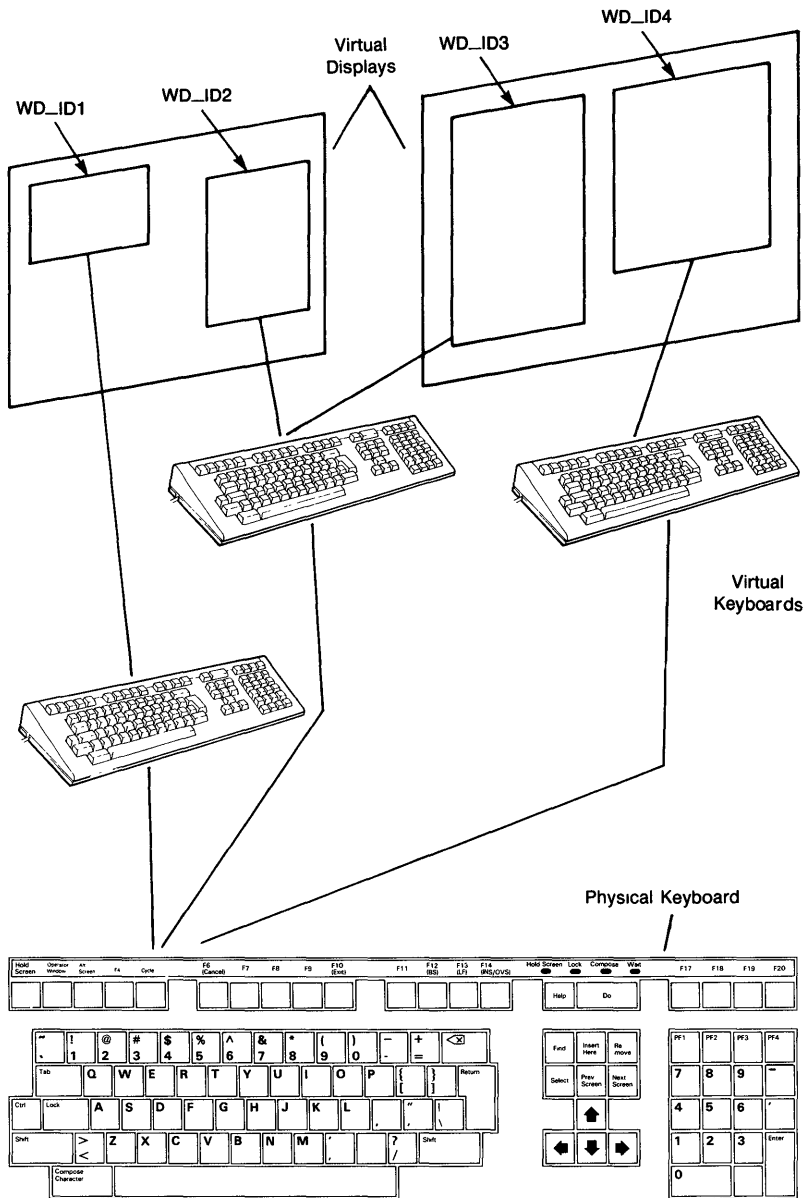
The call to UIS\$DISABLE_VIEWPORT_KB ❸ explicitly disables the binding between the virtual keyboard and the display window. Also, the ability to assign the physical keyboard to the appropriate virtual keyboard, that is, to *cycle* from viewport to viewport, is disabled.

If UIS\$ENABLE_KB were called after UIS\$ENABLE_VIEWPORT_KB, the KB icon would have been highlighted as soon as the program executed.

18-30 UIS Routine Descriptions

UIS\$CREATE_KB

Illustration



UIS\$CREATE_TB

Creates a tablet digitizer identifier that allows you to connect your process to the tablet.

Format

tb_id=UIS\$CREATE_TB *devnam*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Longword value returned as the tablet identifier in the variable *tb_id* or R0 (VAX MACRO). The tablet identifier uniquely identifies the tablet device and can be used in other routines where appropriate.

UIS\$CREATE_TB signals all errors; no condition values are returned.

Argument

devnam
VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify SYS\$WORKSTATION as the default device name character string.

Description

UIS\$CREATE_TB creates a tablet digitizer identifier. When you want to connect to the tablet, you must specify this identifier in a call to UIS\$ENABLE_TB.

code, while the second longword holds an attribute value (which can be real or integer). The constant `WDPL$C_END_OF_LIST` terminates the list.

The window attributes list has the same format as defined in the `UIS$CREATE_WINDOW` service. If your application program is written in FORTRAN, use the `RECORD` data type to construct the attribute list. Refer to `UIS$CREATE_WINDOW` for a description of the attribute list.

devnam

VMS Usage: **device_name**
type: **character string**
access: **write only**
mechanism: **by descriptor**

New terminal emulation device name. The **devnam** argument is the address of a character string descriptor of a location that receives the new terminal emulation device name string.

devlen

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length of the terminal emulation device name string. The **devlen** argument is the address of a word that receives the length of the terminal device name character string.

Description

`UIS$CREATE_TERMINAL` creates a pseudodevice in the VMS database and returns the device name string for the device. The window may not appear on the screen until a channel is assigned to the device using the `SYS$ASSIGN` system service and the first write to the device is performed.

The pseudodevice is created without any initial owner. Once a channel is assigned to the device, it is owned by that process, which is usually the same process that issued the `UIS$CREATE_TERMINAL` call. After all channels have been deassigned, the pseudodevice will be removed automatically from the system. If a permanent pseudodevice is required, then the application should specify a process that maintains a permanent channel to the device.

UIS\$CREATE_TRANSFORMATION

Creates a two-dimensional world coordinate transformation into an existing virtual display's coordinate space. It provides for two-dimensional translation and scaling, but not rotation.

Format

tr_id=**UIS\$CREATE_TRANSFORMATION** *vd_id*, *x*₁, *y*₁,
*x*₂, *y*₂ [*vd**x*₁,
*vd**y*₁, *vd**x*₂, *vd**y*₂]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the transformation identifier in the variable *tr_id* or R0 (VAX MACRO). The transformation identifier uniquely identifies a transformation coordinate space. See the "DESCRIPTION" section below for more information about *tr_id*.

UIS\$CREATE_TRANSFORMATION signals all errors; no condition values are returned.

Arguments

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

*x*₁, *y*₁, *x*₂, *y*₂
VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the new coordinate space. The x_1 and y_1 arguments and the x_2 and y_2 arguments are the addresses of `f_floating` point numbers that define the lower-left corner and upper-right corner of the new transformation coordinate space, respectively.

vd_x₁, vdy₁, vd_x₂, vdy₂

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

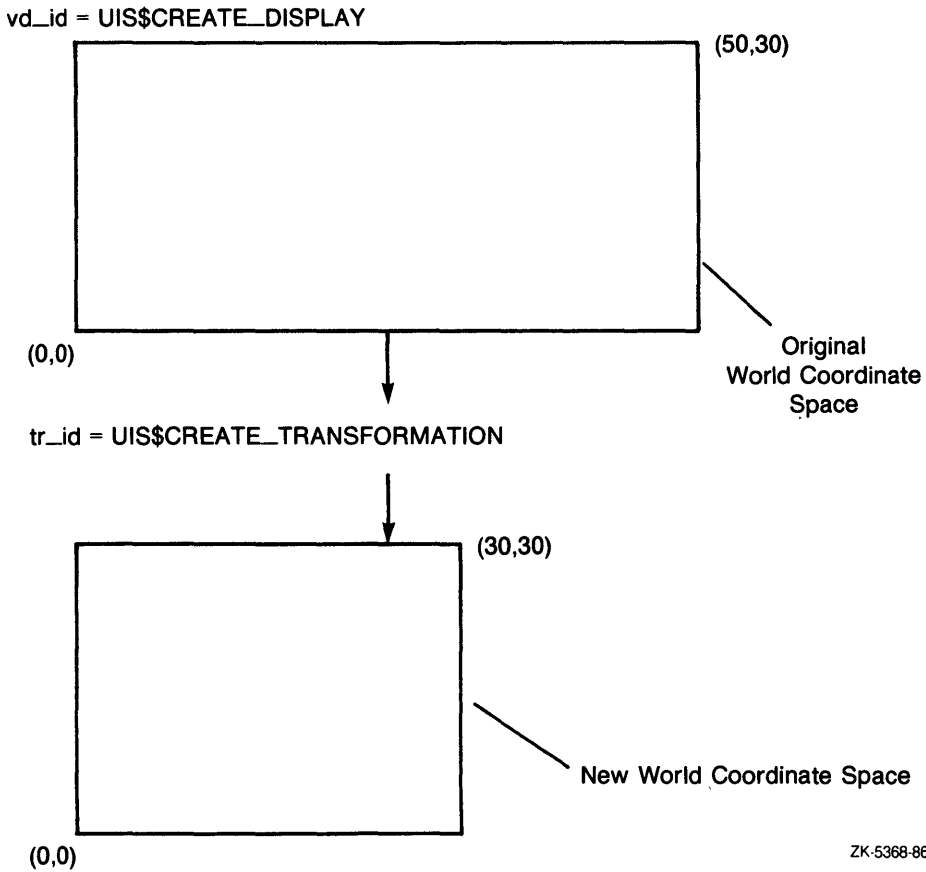
World coordinates of the original virtual display space. The ***vd_x₁*** and ***vdy₁*** arguments are the addresses of `f_floating` point numbers that define the lower-left corner of the corresponding virtual display space. The ***vd_x₂*** and ***vdy₂*** arguments are the addresses of `f_floating` point numbers that define the upper-right corner of the corresponding virtual display space. If these optional arguments are not specified, the world coordinates specified in `UIS$CREATE_DISPLAY` are used.

Description

Once the transformation is created, it can be used in any routine that accepts a ***vd_id*** argument except `UIS$DELETE_DISPLAY` by substituting the ***tr_id*** argument instead. When the ***tr_id*** value is used, it indicates the same virtual display but that the coordinates are mapped relative to the transformation coordinate space, and not the original virtual display coordinate space. Each routine automatically performs the transformation.

18-36 **UIS Routine Descriptions**
UIS\$CREATE_TRANSFORMATION

Illustration



UIS\$CREATE_WINDOW

Creates a display window and an associated display viewport. See UIS\$GET_WINDOW_ATTRIBUTES for information about window attributes.

Format

wd_id=UIS\$CREATE_WINDOW *vd_id*, *devnam* [*title*] [*x*₁,
*y*₁, *x*₂, *y*₂] [*width*, *height*]
[*attributes*]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the display window identifier in the variable *wd_id* or R0 (VAX MACRO). The display window identifier uniquely identifies the display window and is used as an argument in other routines.

UIS\$CREATE_WINDOW signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword value that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

18-38 UIS Routine Descriptions

UIS\$CREATE_WINDOW

Device name. The **devnam** argument is the address of a character string descriptor of the display device on which the display viewport is created. Specify the logical name SYS\$WORKSTATION as the name of the device.

title

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Banner title. The **title** argument is the address of a descriptor of the character string to be inserted into the banner of the display viewport. If the argument **title** is not specified, the display banner is created without a title.

x₁, y₁ x₂, y₂

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the display window. The **x₁, y₁** and **x₂, y₂** arguments are addresses of **f_floating** point numbers that define the lower-left corner and upper-right corner of the display window rectangle. The display window rectangle defines the visible portion of the virtual display. The world coordinate space of the display window rectangle is mapped to the display screen as the display viewport.

If these coordinates are not specified, the entire world coordinate space specified in the UIS\$CREATE_DISPLAY routine is used.

width, height

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Initial dimensions of the display viewport. The **width** and **height** arguments are addresses of **f_floating** point numbers that define the width and height of the display viewport in centimeters. If the **width** and **height** arguments of the display viewport specified in UIS\$CREATE_WINDOW are different from the **width** and **height** arguments specified in the UIS\$CREATE_DISPLAY routine, the default values of UIS\$CREATE_DISPLAY are overridden and scaling occurs.

If the world coordinates of the display window **are** specified and the **width** and **height** arguments are **not** specified, the default dimensions of the display viewport are calculated from the ratios of the world coordinate values and the width and height specified in UIS\$CREATE_DISPLAY. See

the Description section for more information about calculating the default display viewport dimensions.

Display viewports that are too large to fit on the screen are automatically proportionally scaled in size.

attributes

VMS Usage: **item_list_pair**
 type: **longword integer (signed) or f_floating**
 access: **read only**
 mechanism: **by reference**

Display viewport attribute list. The **attributes** argument is the address of a data structure that contains longword pairs, or *doublets*. The first longword stores an attribute ID code and the second longword holds the attribute value (which can be real or integer). The constant **WDPL\$C_END_OF_LIST** terminates this list. FORTRAN application programs should create a record using the RECORD statement to construct this list. It has the following format.

Attribute ID code (WDPL\$C__xxx)
Longword value for attribute identified in previous longword
2nd attribute ID code
2nd attribute value
• • •
End of list = 0 (WDPL\$C__END__OF__LIST)

ZK-4581-85

Window attributes are optional and control window placement and attributes.

18-40 **UIS Routine Descriptions**
UIS\$CREATE_WINDOW

Attribute	Description
WDPL\$C_ABS_POS_X	<p>Exact x placement on the screen.</p> <p>This attribute defines the x origin of the viewport relative to the lower-left corner of the screen. The value is expressed as an f_floating point number of centimeters. Note that the actual point WDPL\$C_ABS_POS_X defines is the lower left corner of the display viewport without the border. Along with WDPL\$C_ABS_POS_Y, this provides the ability to place exactly a new viewport at a specific position anywhere on the workstation screen.</p>
WDPL\$C_ABS_POS_Y	<p>Exact y placement on the screen.</p> <p>This attribute defines the y origin of the viewport relative to the lower-left corner of the screen. The value is expressed as an f_floating point number of centimeters. Note that the actual point WDPL\$C_ABS_POS_Y defines is the lower-left corner of the display viewport without the border. Along with WDPL\$C_ABS_POS_X, this attribute provides the ability to place exactly a new viewport at a specific position anywhere on the workstation screen.</p>
WDPL\$C_PLACEMENT	<p>Display viewport placement flags.</p> <p>The attribute list is a longword bit vector providing viewport placement information. The preference masks (top, bottom, left, and right) may be combined by setting more than one bit in the bit vector. If the screen becomes crowded, the system may override the preference masks.</p> <ul style="list-style-type: none">• WDPL\$M_TOP — The display viewport is placed near the top of the physical display• WDPL\$M_BOTTOM — The display viewport is placed near the bottom of the physical display• WDPL\$M_LEFT — The display viewport is placed near the left side of the physical display• WDPL\$M_RIGHT — The display viewport is placed near the right side of the physical display• WDPL\$M_CENTER — The display viewport is centered over the position specified by WDPL\$C_ABS_POS_X and WDPL\$C_ABS_POS_Y.

Attribute	Description
	<ul style="list-style-type: none"> • WDPL\$M_INVISIBLE — The display viewport is created invisibly, that is, off the screen and, hence, cannot be seen. • Other bits — The remaining bits are reserved to DIGITAL and must be set to zero.
WDPL\$C_ATTRIBUTES	<p>Display viewport attributes.</p> <p>This data structure argument causes the display viewport to be created with one or more of the following attributes. These attributes are specified as bits in a longword mask.</p> <ul style="list-style-type: none"> • WDPL\$M_ALIGNED—The left inner edge of the display viewport is to be aligned on byte boundaries. Applications, such as the VT220 terminal emulator, can use WDPL\$M_ALIGNED to take advantage of text drawing performance optimizations when 8-bit characters are written on byte boundaries. • WDPL\$M_NOBANNER—The display viewport is created without a banner. If a banner title was specified, it is ignored. • WDPL\$M_NOBORDER—The display viewport is created without a border. When you specify WDPL\$M_NOBORDER, the attribute WDPL\$M_NOBANNER is implied. A viewport created without a border cannot be moved with the user interface. • WDPL\$M_NOKB_ICON—The display viewport banner is created without a KB icon. Specify this attribute, if you are sure the application will never require a KB icon or if you wish to add more space in the banner for the title. Otherwise, UIS saves an extra quarter of an inch in the banner for the KB icon. • WDPL\$M_NOMENU_ICON—The display viewport banner is created without a menu icon. Therefore, the Window Options Menu cannot be activated. • Other bits—The remaining bits are reserved to DIGITAL and must be zero.

18-42 **UIS Routine Descriptions**
UIS\$CREATE_WINDOW

Attribute	Description
WDPL\$C_END_OF_LIST	Terminates attributes list. This must be the last longword in the attribute list. It does not require an associated longword value.

Description

UIS\$CREATE_WINDOW defines a portion of the virtual display that lies within the display window and that is mapped to the display screen as the display viewport.

Default Dimensions of the Display Viewport

Whenever the world coordinates of the display window are defined, but the dimensions of the display viewport are not specified, the system calculates the default dimensions of the display viewport using the appropriate arguments from each routine as shown in the following figure. The size of the display viewport is based on the **width** and **height** arguments in UIS\$CREATE_DISPLAY in the following manner:

$$\begin{array}{rcl}
 \text{UIS\$CREATE_DISPLAY} & & \text{UIS\$CREATE_WINDOW} \\
 \\
 \frac{\text{width}}{x_2 - x_1} & = & \frac{\text{new_width}}{x_2 - x_1} \\
 \\
 \frac{\text{height}}{y_2 - y_1} & = & \frac{\text{new_height}}{y_2 - y_1}
 \end{array}$$

ZK-5462-86

The variables new_width and new_height represent unknown quantities, the default dimensions of the display viewport. All other variables are the parameters used in the respective routine calls.

For example, the viewport that is created in the following example is 4 centimeters wide and 2 centimeters high.

```

vd_id=UIS$CREATE_DISPLAY(0.0,0.0,1.0,1.0,8.0,4.0)
wd_id=UIS$CREATE_WINDOW(vd_id,'SYS$WORKSTATION','TEST WINDOW',
0.0,0.0,0.5,0.5)

```

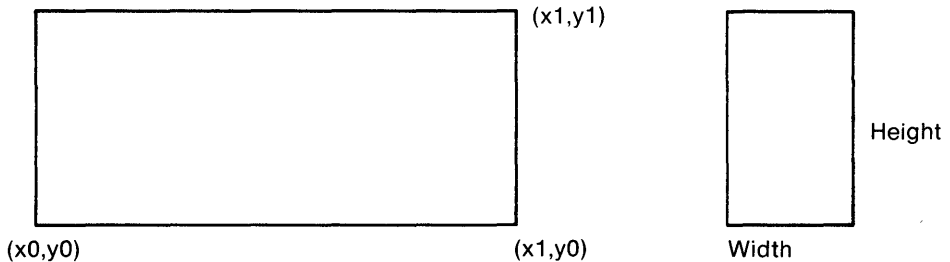
Otherwise, these values can be overridden with the optional **width** and **height** arguments in UIS\$CREATE_WINDOW.

Display Viewport Creation

Display viewports are always created completely on or off the display screen.

Distortion of Graphic Objects

To avoid distortion of graphic objects, the aspect ratios of the display window and the display viewport must be equal.



ZK-4582-85

In the preceding illustration, the aspect ratio of the display window on the left does not appear to be equal to the aspect ratio of the viewport on the right.

You can compare aspect ratios using the following equation.

$$\frac{|y_1 - y_0|}{|x_1 - x_0|} = \frac{\text{viewport height}}{\text{viewport width}}$$

ZK-4579-85

The aspect ratio of the display viewport is the absolute value of the height divided by the absolute value of the width.

18-44 UIS Routine Descriptions

UIS\$CREATE_WINDOW

Example

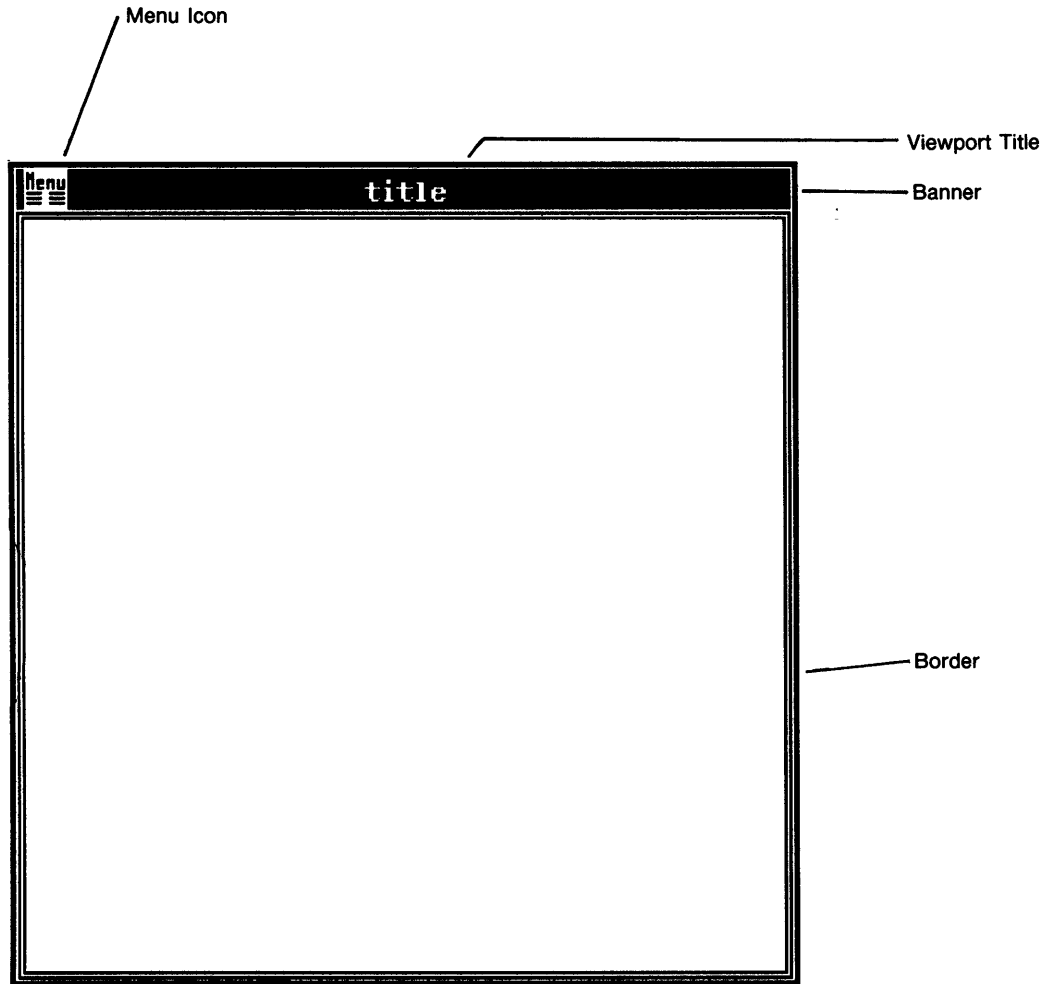
```
PROGRAM EXAMPLE_A
.
.
.
STRUCTURE/STRUCT/      ❶
  INTEGER*4 CODE_1
  REAL*4 ATTRIB_1
  INTEGER*4 CODE_2
  REAL*4 ATTRIB_2
  INTEGER*4 CODE_3
  INTEGER*4 ATTRIB_3
  INTEGER*4 END
END STRUCTURE

RECORD/STRUCT/WINDOW   ❷

WINDOW.CODE_1=WDPL$C_ABS_POS_X
WINDOW.ATTRIB_1=10.5
WINDOW.CODE_2=WDPL$C_ABS_POS_Y
WINDOW.ATTRIB_2=13.25
WINDOW.CODE_3=WDPL$C_ATTRIBUTES
WINDOW.ATTRIB_3=WDPL$M_NOKB_ICON .OR. WDPL$M_NOMENU_ICON
WINDOW.END=WDPL$C_END_OF_LIST
.
.
.
VD_ID=UIS$CREATE_DISPLAY(-10.0,-10.0,35.5,35.5,16.0,16.0)  ❸
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','LOOK',2.0,2.0,28.0,28.0
2      20.0,20.0,WINDOW)  ❹
.
.
.
```

The preceding example describes how to construct the data structure argument used in `UIS$CREATE_WINDOW` to enable viewport placement and characteristics ❶ ❷. In addition, the example illustrates the minimum number of calls used to create a display window ❸ ❹.

Screen Output



UIS\$DELETE_COLOR_MAP

Deletes a virtual color map.

Format

UIS\$DELETE_COLOR_MAP *vcm_id*

Returns

UIS\$DELETE_COLOR_MAP signals all errors; no condition values are returned.

Argument

vcm_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual color map identifier. The **vcm_id** argument is the address of a longword that uniquely identifies the virtual color map. See UIS\$CREATE_COLOR_MAP for more information about the **vcm_id** argument.

Description

An attempt to delete an active virtual color map, that is, a virtual color map associated with one or more virtual displays, signals an error.

All virtual displays that reference the virtual color map should be deleted first using UIS\$DELETE_DISPLAY.

UIS\$DELETE_COLOR_MAP_SEG

Deletes the specified color map segment.

Format

UIS\$DELETE_COLOR_MAP_SEG *cms_id*

Returns

UIS\$DELETE_COLOR_MAP_SEG signals all errors; no condition values are returned.

Argument

cms_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Color map segment identifier. The ***cms_id*** argument is the address of a longword that uniquely identifies the color map segment to be deleted. See UIS\$CREATE_COLOR_MAP_SEG for more information about the ***cms_id*** argument.

Description

Color map segment deletion has no effect on the colors being mapped by the hardware color map. The deletion of color map segments marks the corresponding entries as available for allocation.

An attempt to delete an active color map segment, that is, a color map segment referenced by a virtual color map, signals an error.

The virtual color map should be deleted first using UIS\$DELETE_COLOR_MAP.

18-48 **UIS Routine Descriptions**
UIS\$DELETE_DISPLAY

UIS\$DELETE_DISPLAY

Deletes the virtual display, all associated windows, and viewports.

Format

UIS\$DELETE_DISPLAY *vd_id*

Returns

UIS\$DELETE_DISPLAY signals all errors; no condition values are returned.

Argument

vd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

Description

You cannot substitute the **tr_id** argument for the virtual display identifier in this routine.

UIS\$DELETE_KB

Deletes a virtual keyboard. If the specified virtual keyboard is bound to a window or to the physical keyboard, those bindings are terminated.

Format

UIS\$DELETE_KB *kb_id*

Returns

UIS\$DELETE_KB signals all errors; no condition values are returned.

Argument

kb_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the **kb_id** argument.

Description

UIS\$DELETE_KB may be used to delete a virtual keyboard at any time within a program.

UIS\$DELETE_OBJECT

Deletes the specified object from the virtual display.

Format

UIS\$DELETE_OBJECT { *obj_id* }
 { *seg_id* }

Returns

UIS\$DELETE_OBJECT signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies the object to be deleted.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

Description

The screen is updated immediately to reflect the new state of the virtual display. If it is impossible to modify only those portions which have changed, then the entire display may be replotted. Occluded objects are always refreshed.

UIS\$DELETE_PRIVATE

Deletes the private data associated with the object.

Format

UIS\$DELETE_PRIVATE { *obj_id* }
 { *seg_id* }

Returns

UIS\$DELETE_PRIVATE signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies an object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

Description

If more than one private data item exists, all private data items are deleted.

UIS\$DELETE_TB

Deletes the tablet digitizer identifier and disconnects the application from the tablet.

Format

UIS\$DELETE_TB *tb_id*

Returns

UIS\$DELETE_TB signals all errors; no condition values are returned.

Argument

tb_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Tablet identifier. The **tb_id** argument is the address of a longword that uniquely identifies the tablet device. See UIS\$CREATE_TB for more information about the **tb_id** argument.

Description

UIS\$DELETE_TB deletes a tablet digitizing identifier. When your process has completed digitizing, you should call this routine to delete the identifier.

UIS\$DELETE_TRANSFORMATION

Deletes a world coordinate transformation of a virtual display. The corresponding virtual display is not affected.

Format

UIS\$DELETE_TRANSFORMATION *tr_id*

Returns

UIS\$DELETE_TRANSFORMATION signals all errors; no condition values are returned.

Argument

tr_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Transformation identifier. The **tr_id** argument is the address of a longword that uniquely identifies the transformation to be deleted. See UIS\$CREATE_TRANSFORMATION for more information about the **tr_id** argument.

UIS\$DELETE_WINDOW

Deletes an existing display window and viewport.

Format

UIS\$DELETE_WINDOW *wd_id*

Returns

UIS\$DELETE_WINDOW signals all errors; no condition values are returned.

Argument

wd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window to be deleted. See UIS\$CREATE_WINDOW for more information about the argument **wd_id**.

Description

UIS\$DELETE_WINDOW deletes the display window specified by the **wd_id** argument. The associated viewport is removed from the screen. The virtual display associated with this display window is neither modified nor destroyed during the execution of this service.

UIS\$DISABLE_DISPLAY_LIST

Disables specified display list functions.

Format

UIS\$DISABLE_DISPLAY_LIST *vd_id* [,*display_flags*]

Returns

UIS\$DISABLE_DISPLAY_LIST signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display whose display list should be disabled. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

display_flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display list flags. The **display_flags** argument is address of a longword mask that controls display screen and display list updates.

The following table describes the flags and masks.

18-56 **UIS Routine Descriptions**
UIS\$DISABLE_DISPLAY_LIST

Flag	Description
UIS\$M_DL_ENHANCE_LIST	Controls making additions to the display list. When disabled, no new display list entries are made. This flag is set by default when a virtual display is created.
UIS\$M_DL_MODIFY_LIST	Controls display list modifications. When disabled, no display list editing is allowed. This flag is set by default when a virtual display is created.
UIS\$M_DL_UPDATE_WINDOW	Controls drawing. When disabled, no drawing or update occurs. This flag is set by default when a virtual display is created.

The following table lists UIS routines that check the flags.

Flag	UIS Routine
UIS\$M_DL_MODIFY_LIST ¹	UIS\$COPY_OBJECT UIS\$DELETE_OBJECT UIS\$ERASE UIS\$INSERT_OBJECT UIS\$MOVE_AREA UIS\$TRANSFORM_OBJECT
UIS\$M_DL_ENHANCE_LIST ¹	UIS\$CIRCLE UIS\$ELLIPSE UIS\$EXECUTE UIS\$EXECUTE_DISPLAY UIS\$IMAGE UIS\$LINE UIS\$LINE_ARRAY UIS\$PLOT UIS\$PLOT_ARRAY UIS\$TEXT

¹All routines listed under UIS\$M_DL_ENHANCE_LIST and UIS\$M_DL_MODIFY_LIST will also check the state of UIS\$M_DL_UPDATE_WINDOW before doing any screen updates.

If a bit is set in the mask, the corresponding function is disabled. If the bit is 0, the corresponding function is not changed. See UIS\$ENABLE_DISPLAY_LIST for information on how to enable functions.

If **display_flags** is not specified, UIS\$M_DL_ENHANCE_LIST is disabled.

Description

UIS\$DISABLE_DISPLAY_LIST is useful in applications such as animation. In such a case, display list additions are neither necessary nor desired because of the additional overhead.

Example

At some point in your application you may wish to perform several modifications to the display list without seeing the screen change.

```
.  
. .  
. .  
display_flags= UIS$M_DL_UPDATE_WINDOW  
. .  
. .  
CALL UIS$DISABLE_DISPLAY_LIST(VD_ID, DISPLAY_FLAGS)  
. .  
. .  
Insert your modifications here  
. .  
. .  
CALL UIS$ENABLE_DISPLAY_LIST(VD_ID, DISPLAY_FLAGS)  
CALL UIS$EXECUTE(VD_ID)    ! Erases and redraws the virtual display
```

UIS\$DISABLE_KB

Disconnects the physical keyboard from the specified virtual keyboard. See the example in `UIS$CREATE_KB` for more information.

Format

UIS\$DISABLE_KB *kb_id*

Returns

`UIS$DISABLE_KB` signals all errors; no condition values are returned.

Argument

kb_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword that uniquely identifies the virtual keyboard to be disabled. See `UIS$CREATE_KB` for more information about the **kb_id** argument.

UIS\$DISABLE_TB

Disconnects the digitizing tablet.

Format

UIS\$DISABLE_TB *tb_id*

Returns

UIS\$DISABLE_TB signals all errors; no condition values are returned.

Argument

tb_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Tablet identifier. The **tb_id** argument is the address of longword that uniquely identifies the tablet device. See UIS\$CREATE_TB for more information about the **tb_id** argument.

Description

UIS\$DISABLE_TB disconnects your process from the tablet. This routine reenables the system pointer and frees the tablet for use by another process.

UIS\$DISABLE_VIEWPORT_KB

Prevents the user from assigning the physical keyboard to a viewport. See the example in `UIS$CREATE_KB` for more information.

Format

UIS\$DISABLE_VIEWPORT_KB *wd_id*

Returns

`UIS$DISABLE_VIEWPORT_KB` signals all errors; no condition values are returned.

Argument

wd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Display window identifier. The `wd_id` argument is the address of a longword that uniquely identifies a display window. The associated viewport of the display window is disabled. See `UIS$CREATE_WINDOW` for more information about the `wd_id` argument.

Description

`UIS$DISABLE_VIEWPORT_KB` removes the display window from the assignment list. You can no longer use the `[CYCLE]` key to make the viewport active. Use `UIS$ENABLE_VIEWPORT_KB` or `UIS$ENABLE_KB` to place the display window on the assignment list.

UIS\$ELLIPSE

Draws an arc along the circumference of an ellipse.

Format

UIS\$ELLIPSE *vd_id, atb, center_x, center_y, xradius, yradius [,start_deg, end_deg]*

Returns

UIS\$ELLIPSE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies the attribute block that will modify the ellipse. If you specify 0 in the **atb** argument, the default settings of attribute block 0 are used.

center_x, center_y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

UIS\$ELLIPSE

Center position *x* and *y* world coordinates. The **center_x** and **center_y** arguments are the addresses of *f_floating* point numbers that define a point in the virtual display that is the center of the ellipse or arc.

xradius

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Radius of the ellipse specified as an *x* world coordinate width. The **xradius** argument is the address of an *f_floating* point number that defines the distance from the center of the ellipse to the circumference of the ellipse or arc.

yradius

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Radius of the ellipse specified as a *y* world coordinate width. The **yradius** argument is the address of an *f_floating* point number that defines the distance from the center of the ellipse to the circumference of the ellipse or arc.

start_deg, end_deg

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Degree at which the arc starts and ends. The **start_deg** and **end_deg** arguments are the addresses of *f_floating* numbers that define the starting point and ending point in degrees on the circumference of the ellipse where the arc or ellipse will be drawn. Degrees are measured clockwise from the top of the ellipse. If these arguments are not specified, *0.0* and *360.0* degrees are assumed. If both arguments are not specified, a complete ellipse is drawn.

Description

UIS\$ELLIPSE uses center position coordinates and *x* and *y* radii to construct an ellipse. Along the circumference of this ellipse, UIS\$ELLIPSE draws an arc for a specified range of degrees.

UIS\$ELLIPSE

The arc is closed by drawing one or more lines between the endpoints. The type of arc associated with the attribute block specifies the way in which the arc is closed. See the UIS\$SET_ARC_TYPE routine.

The points are drawn with the current line pattern and width, and filled with the current fill pattern, if enabled.

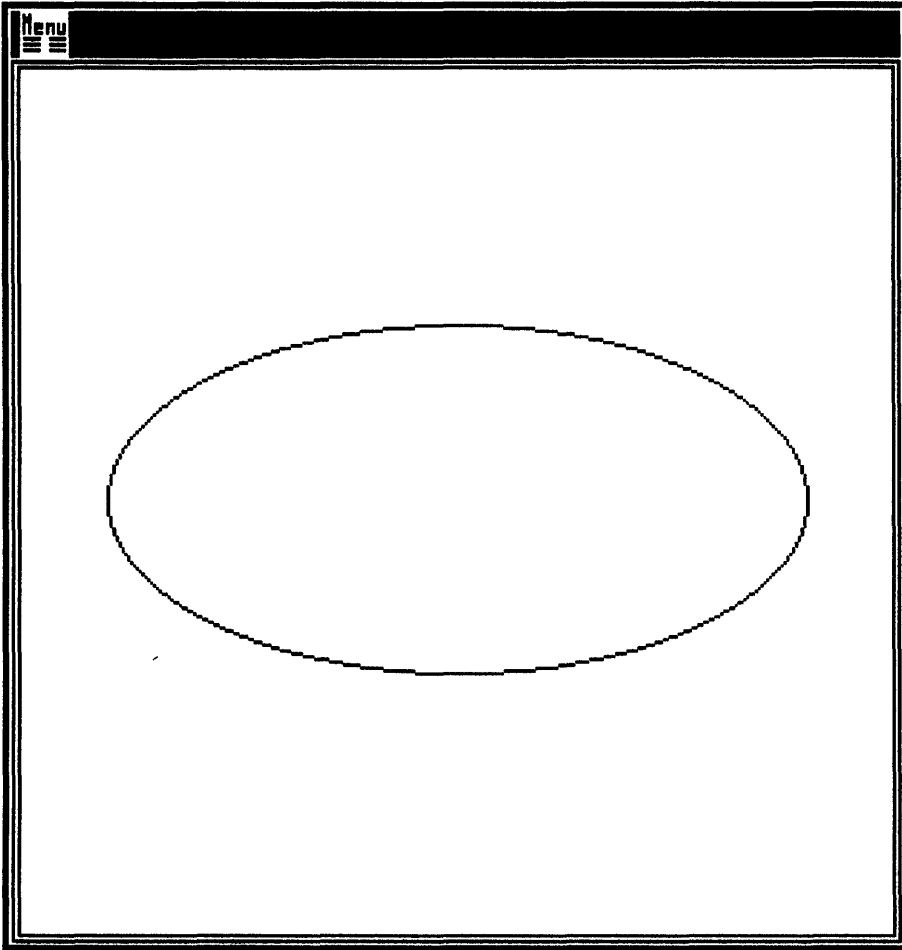
UIS\$ELLIPSE does not support the following combination of attributes:

- Line width not equal to 1 and line style not equal to *FFFFFFFF*₁₆
- Line width not equal to 1 and complement writing mode

Ellipses are distorted by differences between the aspect ratios of the virtual display and display window.

18-64 UIS Routine Descriptions
 UIS\$ELLIPSE

Screen Output



ZK-5418-86

UIS\$ENABLE_DISPLAY_LIST

Reenables automatic additions to the display list.

Format

UIS\$ENABLE_DISPLAY_LIST *vd_id* [*,display_flags*]

Returns

UIS\$ENABLE_DISPLAY_LIST signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display whose display list is to be enabled. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

display_flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display list flags. The *display_flags* argument is the address of a longword mask that controls display screen and display list updates.

The following table describes the flags and masks.

18-66 **UIS Routine Descriptions**
UIS\$ENABLE_DISPLAY_LIST

Flag	Description
UIS\$M_DL_ENHANCE_LIST	Controls making additions to the display list. When disabled, no new display list entries are made. This flag is set by default when a virtual display is created.
UIS\$M_DL_MODIFY_LIST	Controls display list modifications. When disabled, no display list editing is allowed. This flag is set by default when a virtual display is created.
UIS\$M_DL_UPDATE_WINDOW	Controls drawing. When disabled, no drawing or update occurs. This flag is set by default when a virtual display is created.

The following table lists UIS routines that check the flags.

Flag	UIS Routine
UIS\$M_DL_MODIFY_LIST ¹	UIS\$COPY_OBJECT UIS\$DELETE_OBJECT UIS\$ERASE UIS\$INSERT_OBJECT UIS\$MOVE_AREA UIS\$TRANSFORM_OBJECT
UIS\$M_DL_ENHANCE_LIST ¹	UIS\$CIRCLE UIS\$ELLIPSE UIS\$EXECUTE UIS\$EXECUTE_DISPLAY UIS\$IMAGE UIS\$LINE UIS\$LINE_ARRAY UIS\$PLOT UIS\$PLOT_ARRAY UIS\$TEXT

¹All routines listed under UIS\$M_DL_ENHANCE_LIST and UIS\$M_DL_MODIFY_LIST will also check the state of UIS\$M_DL_UPDATE_WINDOW before doing any screen updates.

If a bit is set in the mask, the corresponding function is disabled. If the bit is 0, the corresponding function is not changed.

If **display_flags** is not specified, UIS\$M_DL_ENHANCE_LIST is disabled.

Example

At some point in your application you may wish to perform several modifications to the display list without seeing the screen change.

```
.  
. .  
display_flags= UIS$M_DL_UPDATE_WINDOW  
. .  
CALL UIS$DISABLE_DISPLAY_LIST(VD_ID, DISPLAY_FLAGS)  
. .  
Insert your modifications here  
. .  
CALL UIS$ENABLE_DISPLAY_LIST(VD_ID, DISPLAY_FLAGS)  
CALL UIS$EXECUTE(VD_ID)    ! Erases and redraws the virtual display
```

UIS\$ENABLE_KB

Connects the physical keyboard to the specified virtual keyboard. See the example in `UIS$CREATE_KB` for more information.

Format

UIS\$ENABLE_KB *kb_id* [*,wd_id*]

Returns

`UIS$ENABLE_KB` signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The ***kb_id*** argument is the address of a longword that uniquely identifies the virtual keyboard to be connected to a physical keyboard. See `UIS$CREATE_KB` for more information about the ***kb_id*** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies the display window whose KB icon should be highlighted. See `UIS$CREATE_WINDOW` for more information about the ***wd_id*** argument.

Description

Because it is desirable to leave control of the keyboard to the user, it is recommended that you use the UIS\$ENABLE_KB as little as possible. However, there are times when you may want to use it.

- When you are starting up a new application. In this case, the user may want the workstation keyboard to be implicitly connected to a new application.
- When the physical keyboard is already connected to the application (as determined by the UIS\$TEST_KB routine). In this case, the application may wish to facilitate movement of the keyboard between its windows.

Note that these are not restrictions imposed by the workstation software.

UIS\$ENABLE_TB

Assigns the tablet to the calling process.

Format

UIS\$ENABLE_TB *tb_id*

Returns

UIS\$ENABLE_TB signals all errors; no condition values are returned.

Argument

tb_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Tablet identifier. The **tb_id** argument is the address of a longword that uniquely identifies a tablet device. See UIS\$CREATE_TB for more information about the **tb_id** argument.

Description

Only one application may own the tablet at one time. When a process connects to the tablet, the system hardware cursor is turned off and the connected process receives all the input from the tablet device. The process owns the tablet until it calls UIS\$DISABLE_TB to disconnect itself from the tablet.

The process must use a software cursor to track the pointer in a display window.

UIS\$ENABLE_VIEWPORT_KB

Allows the user to assign a virtual keyboard to the physical keyboard and signals binding through the KB icon in the viewport banner. See the example in UIS\$CREATE_KB for more information.

Format

UIS\$ENABLE_VIEWPORT_KB *kb_id, wd_id*

Returns

UIS\$ENABLE_VIEWPORT_KB signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword that uniquely identifies the virtual keyboard. See UIS\$CREATE_KB for more information about the **kb_id** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

Description

UIS\$ENABLE_VIEWPORT_KB makes the display window as a KB handle. The viewport contains a nonhighlighted KB icon.

UIS\$END_SEGMENT

Ends a current segment in a virtual display.

Format

UIS\$END_SEGMENT *vd_id*

Returns

UIS\$END_SEGMENT signals all errors; no condition values are returned.

Argument

vd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

Description

Context is returned to the parent segment. All values of attribute blocks 0 to 255 are restored to the current values of the parent's attribute blocks.

UIS\$ERASE

Erases the specified rectangle in the virtual display and removes all entities that lie **completely** within the rectangle from the display list.

Format

UIS\$ERASE *vd_id* [*x₁*, *y₁*, *x₂*, *y₂*]

Returns

UIS\$ERASE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display containing the specified rectangle. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

x₁, *y₁*, *x₂*, *y₂*

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinate pairs. The *x₁* and *y₁* arguments are the addresses of *f_floating* point numbers that define the lower-left corner of the rectangle in the virtual display. The *x₂* and *y₂* arguments are the addresses of *f_floating* point numbers that define the upper-right corner of the rectangle in the virtual display. If no rectangle is specified, the entire virtual display is erased.

18-74 **UIS Routine Descriptions**
UIS\$ERASE

Description

UIS\$ERASE removes all graphics entities that lie completely within the rectangle from the display list as if they had never been written. Objects that do not lie completely within the specified rectangle are not erased. Empty segments are not deleted.

Areas within the display window affected by this routine are filled with color specified by entry 0 in the color map of the virtual display.

UIS\$EXECUTE

Executes a binary encoding stream in a specified virtual display.

Format

UIS\$EXECUTE *vd_id* [,*buflen*] [,*bufaddr*]

Returns

UIS\$EXECUTE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

buflen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Length of the binary encoding stream. The **buflen** argument is the address of longword that contains the length of the binary encoding stream.

bufaddr

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **reference**

Binary encoding stream. The **bufaddr** argument is the address of an array of longwords that comprise the binary encoding stream.

18-76 **UIS Routine Descriptions**
UIS\$EXECUTE

Description

If the buffer is omitted, all display windows are erased and refreshed.

Note the effects of the display list flags.

UIS\$EXECUTE_DISPLAY

Creates a virtual display from a display list.

Format

vd_id=UIS\$EXECUTE_DISPLAY *buflen*, *bufaddr*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Longword value returned as the virtual display identifier in the variable *vd_id* or R0 (VAX MACRO).

UIS\$EXECUTE_DISPLAY signals all errors; no condition values are returned.

Arguments

buflen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Buffer length. The ***buflen*** argument is the address of a longword that defines the length of the buffer.

bufaddr

VMS Usage: **vector_byte_signed**
type: **byte integer (signed)**
access: **read only**
mechanism: **by reference**

Buffer address. The ***bufaddr*** argument is the address of an array of integer bytes that contains the binary encoded stream.

The binary encoded stream is executed in the virtual display.

UIS\$EXPAND_ICON

Replaces an icon with its associated viewport.

Format

UIS\$EXPAND_ICON *wd_id* [*,icon_wd_id*] [*,attributes*]

Returns

UIS\$EXPAND_ICON signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

icon_wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Icon window identifier. The ***icon_wd_id*** argument is the address of a longword that uniquely identifies the icon window.

If the ***icon_wd_id*** argument is specified, it must match the value of the ***icon_wd_id*** argument specified in UIS\$SHRINK_TO_ICON.

attributes

VMS Usage: **item_list_pair**
type: **longword integer (signed) or f_floating**
access: **read only**
mechanism: **by reference**

Viewport attributes list. The ***attributes*** argument is the address of data structure such as an array or record. The ***attributes*** can be used to specify exact placement of the display viewport.

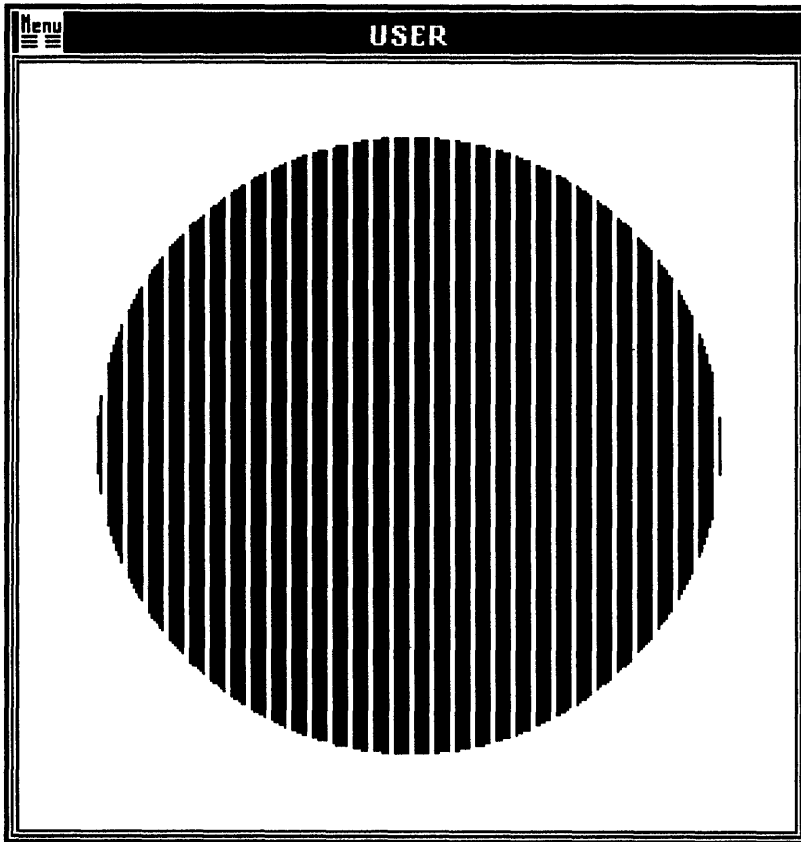
Attribute ID code (WDPL\$_xxx)
Longword value for attribute identified in previous longword
2nd attribute ID code
2nd attribute value
• • •
End of list = 0 (WDPL\$_END_OF_LIST)

ZK-4581-85

See the **attributes** argument in UIS\$CREATE_WINDOW for more information.

18-80 UIS Routine Descriptions
UIS\$EXPAND_ICON

Screen Output



UIS\$EXTRACT_HEADER

Returns the header information needed to create a UIS metafile.

Format

UIS\$EXTRACT_HEADER *vd_id*, [*buflen*, *bufaddr*] [,*retlen*]

Returns

UIS\$EXTRACT_HEADER signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

buflen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Buffer length. The *buflen* argument is the address of a longword that defines the length of the buffer.

bufaddr

VMS Usage: **vector_byte_signed**
type: **byte integer (signed)**
access: **read only**
mechanism: **by reference**

Buffer address. The *bufaddr* argument is the address of an array of bytes that receives the binary encoding stream.

18-82 UIS Routine Descriptions UIS\$EXTRACT_HEADER

retlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Return length. The **retlen** argument is the address of a longword that receives the length of the buffer.

Description

Header information must be at the beginning of all UIS metafiles.

Allocating Space for the Buffer

If you want to know how much space to allocate for the buffer, specify **obj_id** and **retlen** only.

Format of Header Information

The format of header binary instructions is as follows:

Op code 16 bits	Length 16 bits	Arguments
--------------------	-------------------	-----------

ZK-5472-86

If the length field exceeds 32,767 bytes, an extended format is used. The length field should be set to **UIS\$_LENGTH_DIFF** and the extra length field should be set to the total number of bytes in the binary instruction.

Op code 16 bits	Length 16 bits	Extra Length 32 bits	Arguments
--------------------	-------------------	-------------------------	-----------

ZK-5473-86

UIS\$EXTRACT_OBJECT

Returns the binary encoding stream for the desired object (segment or primitive).

Format

UIS\$EXTRACT_OBJECT $\left\{ \begin{array}{l} \text{obj_id} \\ \text{seg_id} \end{array} \right\} [, \text{buflen} , \text{bufaddr}]$
 $[\text{,retlen}]$

Returns

UIS\$EXTRACT_OBJECT signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies the object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

buflen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Length of buffer. The **buflen** argument is the address of a longword that specifies the length of the buffer that receives the binary encoding stream.

18-84 **UIS Routine Descriptions**
UIS\$EXTRACT_OBJECT

bufaddr

VMS Usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference**

Name of an array. The **bufaddr** argument is the address of an array of bytes that receives the binary encoding stream.

retlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Length of the binary encoding stream. The **retlen** argument is the address of a longword that receives the length of the binary encoding stream.

Description

If you want to know how much space to allocate for the buffer, specify **obj_id** and **retlen** only.

If the extracted object lies within a segment, a binary instruction denoting the beginning of the segment precedes all binary instructions associated with the extracted object. A binary instruction denoting the end of the segment follows the binary instructions associated with the extracted object.

18-86 **UIS Routine Descriptions**
UIS\$EXTRACT_PRIVATE

bufaddr

VMS Usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference**

Buffer address. The **bufaddr** argument is the address of an array of bytes that receives the binary encoding stream.

retlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Length of the binary encoding stream. The **retlen** is the address of longword that receives the length of the binary encoding stream.

Description

If more than one private data item is associated with the specified object, all private data items are returned. The following figure describes the format of the data. If you want to know how much space to allocate for the returned encoding, specify the **obj_id** and **retlen** arguments only.

Format of a Private Data Binary Instruction

The format of binary encoding returned is as follows:

Op code 16 bits	Length 16 bits	ATB 16 bits	Arguments
--------------------	-------------------	----------------	-----------

ZK-5475-86

If the length field exceeds 32,767 bytes, an extended format is used. The length field should be set to **UIS\$C_LENGTH_DIFF** and the extra length field should be set to the total number of bytes in the binary instruction.

UIS Routine Descriptions 18-87
UIS\$EXTRACT_PRIVATE

Op code 16 bits	Length 16 bits	Extra Length 32 bits	Arguments
--------------------	-------------------	-------------------------	-----------

ZK-5473-86

Attribute modification instructions precede the binary instruction of the extracted object. The binary instructions of any private data associated with the extracted object follow the binary instruction of the extracted object.

If these arguments are not specified, the coordinates of the entire virtual display are used.

buflen

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Length of a buffer. The **buflen** is the address of a longword that contains the length of the buffer that receives the binary encoding stream.

bufaddr

VMS Usage: **vector_byte_unsigned**
 type: **byte_unsigned**
 access: **read only**
 mechanism: **by reference**

Buffer address. The **bufaddr** argument is the address of an array of bytes that receives the binary encoding stream.

retlen

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **write only**
 mechanism: **by reference**

Length of the binary encoding stream. The **retlen** argument is the address of a longword that receives the length of the binary encoding stream.

Description

If you want to know how much space to allocate for the returned encoding, do not specify the **buflen** and **bufaddr** arguments.

Format of Binary Instructions

The format of binary instructions is as follows:

Op code 16 bits	Length 16 bits	Arguments
--------------------	-------------------	-----------

18-90 **UIS Routine Descriptions**
UIS\$EXTRACT_REGION

If the length field exceeds 32,767 bytes, an extended format is used. The length field should be set to UIS\$C_LENGTH_DIFF and the extra length field should be set to the total number of bytes in the binary instruction.

Op code 16 bits	Length 16 bits	Extra Length 32 bits	Arguments
--------------------	-------------------	-------------------------	-----------

ZK-5473-86

UIS\$EXTRACT_TRAILER

Returns trailer information needed to create a UIS metafile.

Format

UIS\$EXTRACT_TRAILER *vd_id* [*buflen*, *bufaddr*] [*retlen*]

Returns

UIS\$EXTRACT_TRAILER signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

buflen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Buffer length. The **buflen** argument is the address of a longword that defines the length of the buffer.

bufaddr

VMS Usage: **vector_byte_signed**
type: **byte integer (signed)**
access: **read only**
mechanism: **by reference**

Buffer address. The **bufaddr** argument is the address of an array of bytes that receive the binary encoded stream.

18-92 UIS Routine Descriptions

UIS\$EXTRACT_TRAILER

retlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Return length. The **retlen** argument is the address of a longword that defines the returned length of the buffer.

Description

Trailer information must appear at the end of all UIS metafiles.

Allocating Space for the Buffer

If you want to know how much space to allocate for the buffer, specify **obj_id** and **retlen** only.

Format of Trailer Information

The format of trailer binary instructions is as follows:

Op code 16 bits	Length 16 bits	Arguments
--------------------	-------------------	-----------

ZK-5472-86

If the length field exceeds 32,767 bytes, an extended format is used. The length field should be set to **UIS\$C_LENGTH_DIFF** and the extra length field should be set to the total number of bytes in the binary instruction.

Op code 16 bits	Length 16 bits	Extra Length 32 bits	Arguments
--------------------	-------------------	-------------------------	-----------

ZK-5473-86

UIS\$FIND_PRIMITIVE

Locates the next output primitive that intersects the specified rectangle.

Format

obj_id=UIS\$FIND_PRIMITIVE *vd_id*, *x*₁,*y*₁, *x*₂,*y*₂ [*context*]
[*extent*]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the object identifier in the variable *obj_id* or R0 (VAX MACRO). The object identifier uniquely identifies the object and is used as an argument in other routines.

UIS\$FIND_PRIMITIVE signals all errors; no condition values are returned.

Arguments

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

*x*₁,*y*₁,*x*₂,*y*₂
VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the selection rectangle. The *x*₁,*y*₁ and *x*₂,*y*₂ are the addresses of *f_floating* points numbers that define the lower-left and upper-right corners of the rectangle.

18-94 UIS Routine Descriptions

UIS\$FIND_PRIMITIVE

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value. The **context** argument is the address of a longword that stores the state of the search and should not be modified if repetitive searches are desired. If this argument is omitted, only the first match can be found in the display list.

You must initialize the **context** argument to 0 before starting a search operation.

extent

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the extent rectangle array. The **extent** argument is an array of four longwords that receives the world coordinate values of the lower-left and upper-right corner of the extent rectangle.

Description

When you try to locate the specified object closest to the specified location, the size of the rectangle controls the object or primitive matching granularity. Normally, when you search for the primitive nearest a position, the rectangle would surround the position, and have a small width and height (perhaps equivalent to 1 to 10 pixels), depending on the desired granularity.

Once the primitive is located, it returns an object identifier which can be used later to reference the primitive, for example, UIS\$EXTRACT_OBJECT or UIS\$DELETE_OBJECT.

Each time UIS\$FIND_PRIMITIVE is called, it continues the search operation from where it left off, using the context longword to keep track of the current state.

Generally, in order to find all matches, UIS\$FIND_PRIMITIVE is called repeatedly with the same context longword until it returns a value of 0.

UIS\$FIND_SEGMENT

Locates the next segment that contains any objects or primitives that intersect with the specified rectangle.

Format

seg_id=UIS\$FIND_SEGMENT *vd_id*, *x*₁, *y*₁, *x*₂, *y*₂ [,*context*]
[,*extent*]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the segment identifier in the variable *seg_id* or R0 (VAX MACRO). The segment identifier uniquely identifies the segment and is used as an argument in other routines.

UIS\$FIND_SEGMENT signals all errors; no condition values are returned.

Arguments

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

*x*₁,*y*₁,*x*₂,*y*₂
VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the selection rectangle. The *x*₁,*y*₁ and *x*₂,*y*₂ arguments are the addresses of *f_floating* point numbers that define the lower-left and upper-right corners of the rectangle.

18-96 UIS Routine Descriptions

UIS\$FIND_SEGMENT

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Context value. The **context** argument is the address of a longword that stores the state of the search and should not be modified if repetitive searches are desired. If this argument is omitted, only the first match can be found in the display list.

You must initialize the **context** argument to 0 before starting a search operation.

extent

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the extent rectangle array. The **extent** argument is the address of an array of four longwords that receives the world coordinate pairs that define the lower-left and upper-right corners of the extent rectangle containing the segment.

Description

The size of the rectangle controls the matching granularity when trying to locate the primitive closest to a specific position. Normally, when searching for the primitive nearest a position, the rectangle would surround the position, and have a small width and height (perhaps equivalent to 1 to 10 pixels), depending on the desired granularity.

Once the object is located, UIS\$FIND_SEGMENT returns the object identifier for the segment containing that object.

Each time this routine is called, it continues the search operation from where it left off, using the context longword to keep track of the search state.

Generally, in order to find all matches, UIS\$FIND_SEGMENT is called repeatedly with the same context longword until it returns a value of 0.

UIS\$GET_ABS_POINTER_POS

Returns the current pointer position relative to the lower-left corner of the workstation screen.

Format

UIS\$GET_ABS_POINTER_POS *devnam, retx, rety*

Returns

UIS\$GET_ABS_POINTER_POS signals all errors; no condition values are returned.

Arguments

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name string. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify the logical name SYS\$WORKSTATION as the device name string.

retx, rety

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Absolute device coordinate pair. The **retx** and **rety** arguments are the addresses of **f_floating** point longwords that receive the *x* and *y* coordinate positions of the pointer in centimeters relative to the lower-left corner of the display screen.

UIS\$GET_ALIGNED_POSITION

Returns the current position for text output which is the upper-left corner of the character cell.

Format

UIS\$GET_ALIGNED_POSITION *vd_id, atb, retx, rety*

Returns

UIS\$GET_ALIGNED_POSITION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block. The **atb** argument is the address of a longword integer that identifies an attribute block that contains the font to use in calculating the aligned position.

retx, rety

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

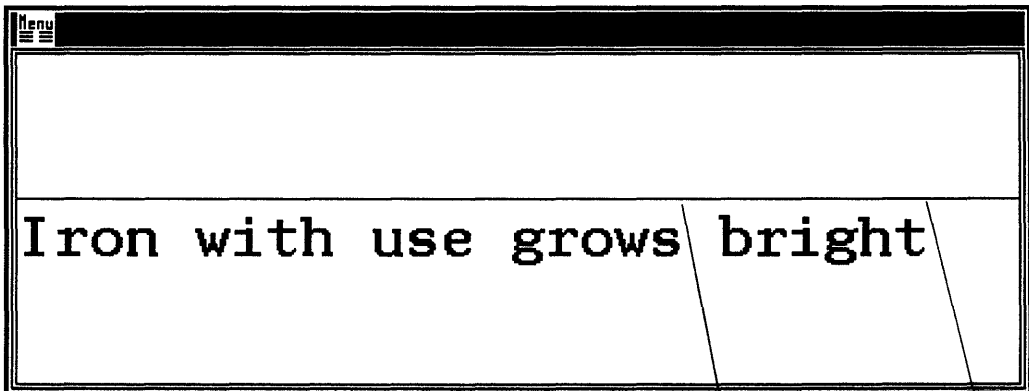
World coordinate pair. The `retx` and `rety` arguments are the addresses of `f_floating` point longwords that receive the current position as `x` and `y` world coordinate positions.

Description

UIS\$GET_ALIGNED_POSITION differs from UIS\$GET_POSITION in that the current position refers to the upper-left corner of the character cell of the next character to be output. This is useful for applications that require the position of the upper-left corner, but do not know enough about the font baseline to determine the proper alignment point. The position is converted into the proper alignment point using the font specified in the given attribute block. See UIS\$SET_ALIGNED_POSITION.

Screen Output

```
⌘ run get_aligned  
x world coordinate = 18.19 y world coordinate = 5.02  
FORTRAN PAUSE  
⌘
```



Text Alignment
Point

Current position
after text
drawing
(18.19, 5.02)

UIS\$GET_ARC_TYPE

Returns the current arc type attribute code. See `UIS$SET_ARC_TYPE` for more information about arc types.

Format

arc_type=**UIS\$GET_ARC_TYPE** *vd_id, atb*

Returns

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the current arc type code in the variable *arc_type*. The arc type code is an integer value representing one of the following UIS constants: `UIS$_ARC_OPEN`, `UIS$_ARC_PIE`, and `UIS$_ARC_CHORD`. See `UIS$SET_ARC_TYPE` for a description of the constants.

`UIS$GET_ARC_TYPE` signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See `UIS$CREATE_DISPLAY` for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block identifier. The *atb* argument is the address of a longword integer that identifies the attribute block from which the arc type is obtained.

Description

Refer to Section 6.6 for more information about UIS symbols and symbol definition files.

UIS\$GET_BACKGROUND_INDEX

Returns the background color index for text and graphics output.

Format

index=**UIS\$GET_BACKGROUND_INDEX** *vd_id*, *atb*

Returns

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the color map index in the variable *index* or R0 (VAX MACRO).

UIS\$GET_BACKGROUND_INDEX signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a longword integer that identifies the attribute block from which the background color index is obtained.

UIS\$GET_BUTTONS

Returns the current state of the pointer buttons.

Format

status=UIS\$GET_BUTTONS *wd_id*, *retstate*

Returns

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Boolean value is returned in the variable *status* or R0 (VAX MACRO). A value of 1 is returned, if the pointer is within the visible portion of the viewport. If the pointer is outside the visible portion of the viewport, a value of 0 is returned.

UIS\$GET_BUTTONS signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

retstate

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

State of the pointer buttons. The *retstate* argument is the address of a longword that receives the current state of the pointer buttons. The state of pointer buttons is returned in a longword whose bits indicate the state of each pointer button, for example, 1 is up and 0 is down. The symbolic definitions for these bits are UIS\$M_POINTER_BUTTON_1, and UIS\$M_

18-104 **UIS Routine Descriptions**
UIS\$GET_BUTTONS

POINTER_BUTTON_2, UIS\$M_POINTER_BUTTON_3, and
UIS\$M_POINTER_BUTTON_4.

Description

The returned status value should always be tested when using this function, because it is always possible that the pointer could be outside the window when the function is called.

UIS\$GET_CHAR_ROTATION

Returns the angle of character rotation in degrees.

Format

angle=UIS\$GET_CHAR_ROTATION *vd_id*, *atb*

Returns

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by value**

Longword value returned as the angle of character rotation in degrees in the variable *angle* or R0 (VAX MACRO). The baseline vector and the actual path of text drawing form the angle of character rotation. The character rotates on its baseline point.

UIS\$GET_CHAR_ROTATION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a number that identifies an attribute block containing the character rotation attribute used to calculate character rotation.

char

VMS Usage: **char_string**
type: **character_string**
access: **write only**
mechanism: **by descriptor**

Single character. The **char** argument is the address of a character string descriptor of a single char. The **char** is specified only for proportionally spaced fonts. It is used as a reference point against which other characters are scaled.

width

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Character width. The **width** argument is the address of an **f_floating** point longword that receives the character width in world coordinates.

height

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Character height. The **height** argument is the address of an **f_floating** point longword that receives the character height in world coordinates.

UIS\$GET_CHAR_SLANT

Returns the angle of character slant in degrees.

Format

angle = **UIS\$GET_CHAR_SLANT** *vd_id*, *atb*

Returns

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by value**

Longword value returned as the angle of character slant in degrees in the variable *angle* or R0 (VAX MACRO). The character cell up vector and the baseline vector form the angle of character slant.

UIS\$GET_CHAR_SLANT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

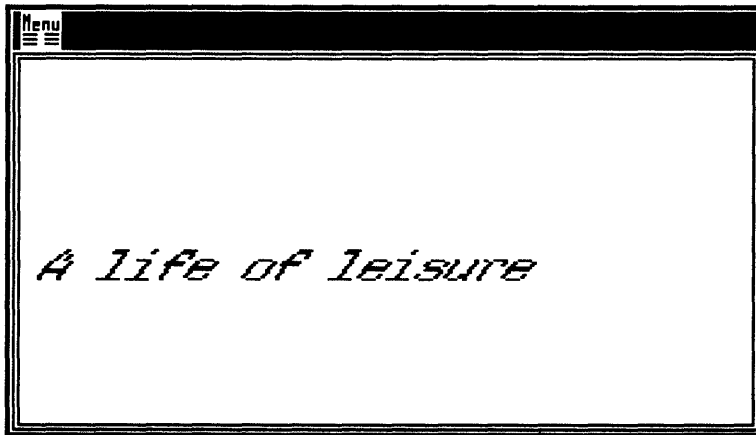
atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a number that identifies an attribute block containing the character slant attribute setting to be returned.

Screen Output

```
⌘ run get_charslant  
The angle of character slant is      35.00 degrees  
FORTRAN PAUSE  
⌘
```



UIS\$GET_CHAR_SPACING

Returns the character spacing factors.

Format

UIS\$GET_CHAR_SPACING *vd_id, atb, dx, dy*

Returns

UIS\$GET_CHAR_SPACING signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies the attribute block from which the character spacing factors are obtained.

dx

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Additional x spacing factor. The **dx** argument is the address of an **f_floating** point longword that receives the x spacing factor. The *x* spacing factor

represents the relative width of the character cell. If 0 is returned, no additional spacing factor was specified.

dy

VMS Usage: **floating_point**

type: **f_floating**

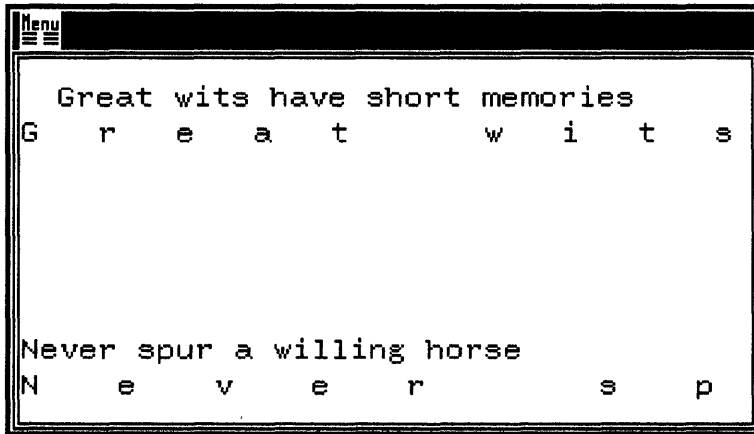
access: **write only**

mechanism: **by reference**

Additional y spacing factor. The **dy** argument is the address of an **f_floating** point longword that receives the y spacing factor. The y spacing factor represents the relative height of the character cell. If 0 is returned, no additional spacing factor was specified.

Screen Output

```
⌘ run get_charspace
x spacing factor = 0.00   y spacing factor = 0.00
x spacing factor = 3.00   y spacing factor = 5.00
x spacing factor = 0.00   y spacing factor = 0.00
x spacing factor = 4.00   y spacing factor = 6.00
FORTRAN PAUSE
⌘
```



UIS\$GET_CLIP

Returns the clipping mode.

Format

status=**UIS\$GET_CLIP** *vd_id, atb* [*x₁, y₁, x₂, y₂*]

Returns

VMS Usage: **boolean**
type: **longword**
access: **write only**
mechanism: **by value**

Boolean value returned as the clipping mode in a status variable or R0 (VAX MACRO). If clipping is enabled, a boolean TRUE is returned. If clipping is disabled, a boolean FALSE is returned.

UIS\$GET_CLIP signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a longword integer that identifies the attribute block from which the clipping rectangle and mode are obtained.

x_1, y_1, x_2, y_2

VMS Usage: **floating_point**

type: **f_floating**

access: **write only**

mechanism: **by reference**

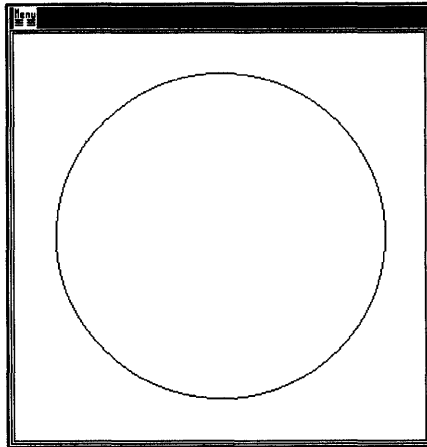
World coordinate pair. The x_1 and y_1 arguments are addresses of **f_floating** point longwords that receive the coordinates of the lower-left corner of the world coordinate clipping rectangle. The x_2 and y_2 arguments are the addresses of **f_floating** point longwords that receive the coordinates of the upper-right corner of the world coordinate clipping rectangle.

18-114 UIS Routine Descriptions

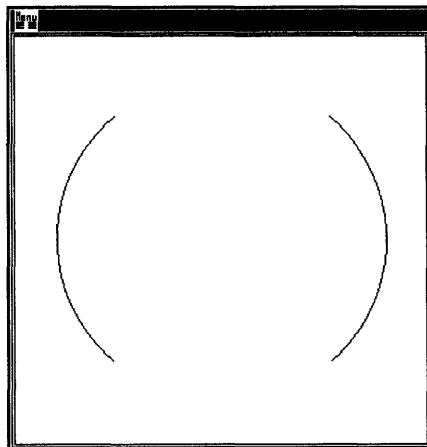
UIS\$GET_CLIP

Screen Output

```
$ run get_clip  
Is clipping enabled? F = FALSE T = TRUE  
F  
FORTRAN PAUSE  
$
```



```
FORTRAN PAUSE  
$ cont  
Is clipping enabled? F = FALSE T = TRUE  
T  
FORTRAN PAUSE  
$
```



UIS\$GET_COLOR

Returns a single red green blue (RGB) color value associated with an entry in a virtual color map.

Format

UIS\$GET_COLOR *vd_id, index, retr, retg, retb [,wd_id]*

Returns

UIS\$GET_COLOR signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual color map index. The **index** argument is the address of a longword that specifies the index of the virtual color map entry to be returned.

retr

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Red value. The **retr** argument is the address of an **f_floating** point longword that receives the red value. The red value is in the range of 0.0 to 1.0, inclusive.

18-116 **UIS Routine Descriptions**
UIS\$GET_COLOR

retg

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Green value. The **retg** argument is the address of an **f_floating** point longword that receives the green value. The green value is in the range of *0.0* to *1.0*, inclusive.

retb

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Blue value. The **retb** argument is the address of an **f_floating** point longword that receives the blue value. The blue value is in the range of *0.0* to *1.0*, inclusive.

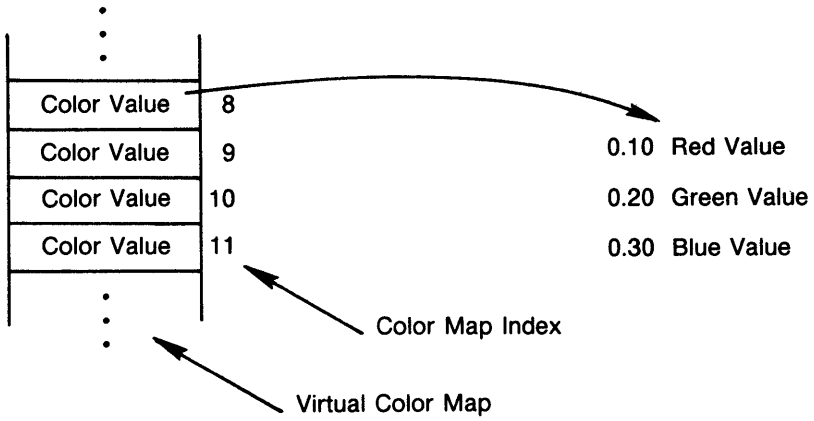
wd_id

VMS Usage: **object_id**
type: **longword**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. If this argument is specified, it must be a valid **wd_id** associated with the virtual display. The colors returned are the realized colors for the specific device for which the window was created. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

If **wd_id** is not specified, the *set* color values, that is, the actual color values in the specified color map entry are returned.

Illustration



UIS\$GET_COLORS

Returns red, green, and blue (RGB) color values associated with one or more entries in the virtual color map.

Format

UIS\$GET_COLORS *vd_id, index, count, retr_vector,*
retg_vector, retb_vector [,wd_id]

Returns

UIS\$GET_COLORS signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Starting color map index. The **index** argument is the address of a longword that specifies the index of the first color map entry to be returned.

If the specified index exceeds the maximum index for the virtual color map, an error is signaled.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

UIS\$GET_COLORS

Number of virtual color map indices. The **count** argument is the address of a longword that defines the total number of color map entries in the virtual color map to be returned including the starting index.

If the total number of indices exceeds the maximum number of indices in the virtual color, an error is signaled.

retr_vector

VMS Usage: **vector_longword_signed**

type: **f_floating**

access: **write only**

mechanism: **by reference**

Red values. The **retr_vector** argument is the address of an array of **f_floating** point longwords that receives the red color values. Each red value is in the range of *0.0* to *1.0*, inclusive.

retg_vector

VMS Usage: **vector_longword_signed**

type: **f_floating**

access: **write only**

mechanism: **by reference**

Green values. The **retg_vector** argument is the address of an array of **f_floating** point longwords that receives the green color values. Each green value is in the range of *0.0* to *1.0*, inclusive.

retb_vector

VMS Usage: **vector_longword_signed**

type: **f_floating**

access: **write only**

mechanism: **by reference**

Blue values. The **retb_vector** argument is the address of an array of **f_floating** point longwords that receives the blue color values. Each blue value is in the range of *0.0* to *1.0*, inclusive.

wd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

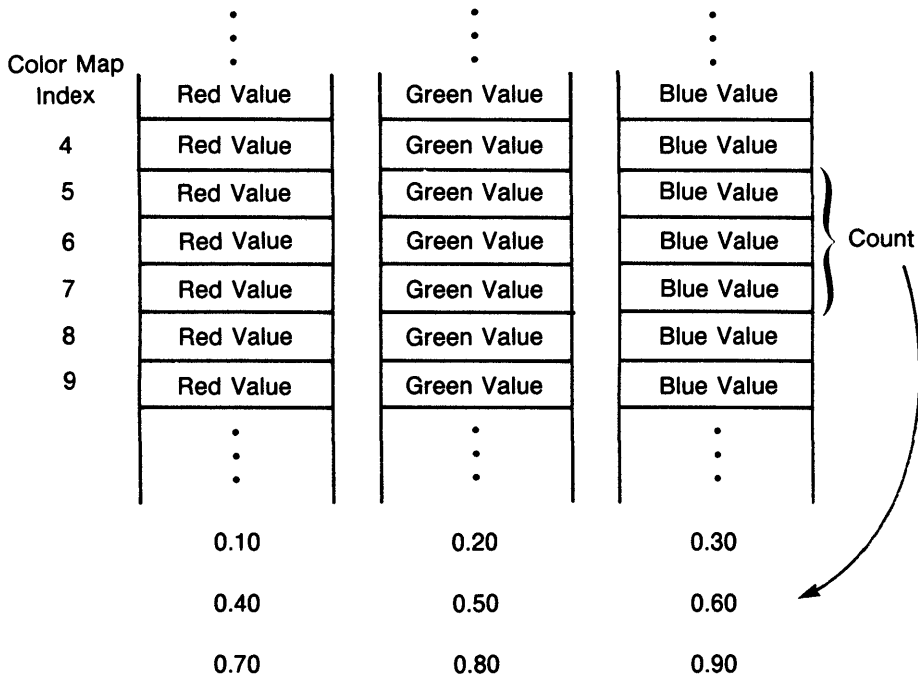
Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. If specified, the **wd_id** argument must be a valid display window identifier associated with the virtual display. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

18-120 **UIS Routine Descriptions**
UIS\$GET_COLORS

The color values returned are the *realized* color values for the specific device for which the display window was created.

If the `wd_id` argument is not specified, the red, green, and blue color values returned are the *set* color values originally established by `UIS$SET_COLOR` or `UIS$SET_COLORS`.

Illustration



UIS\$GET_CURRENT_OBJECT

Returns the identifier of the last object drawn in the virtual display and added to the display list.

Format

current_id=UIS\$GET_CURRENT_OBJECT *vd_id*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the identifier of the current object in the variable *current_id* or R0 (VAX MACRO).

UIS\$GET_CURRENT_OBJECT signals all errors; no condition values are returned.

Argument

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

Description

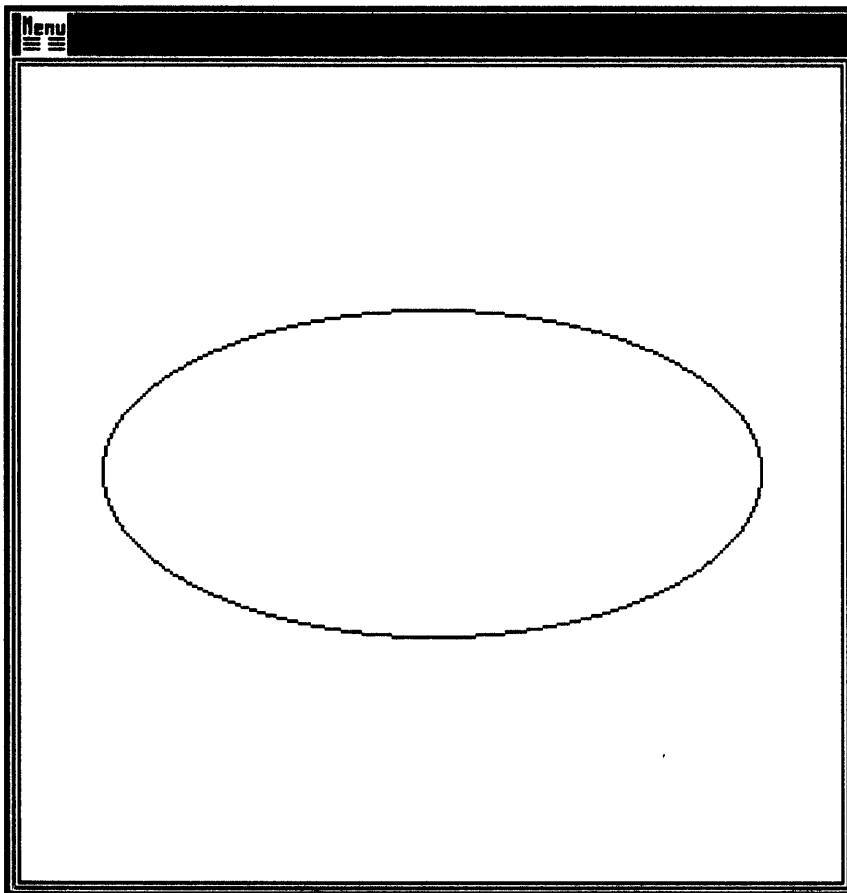
If there are no objects in the display list, the root segment identifier is returned. If UIS\$GET_CURRENT_OBJECT is called after a call to UIS\$SET_INSERTION_POSITION, the returned identifier is based on the current insertion position in the segment.

18-122 UIS Routine Descriptions
UIS\$GET_CURRENT_OBJECT

Screen Output

```
Ⓢ run get_currobj  
Identifier of current object = 114752  
FORTRAN PAUSE
```

```
Ⓢ █
```



UIS\$GET_DISPLAY_SIZE

Obtains the dimensions of the workstation display screen.

Format

UIS\$GET_DISPLAY_SIZE *devnam, retwidth, reheight
[,retresolx, retresoly] [,retpwidth,
retpheight]*

Returns

UIS\$GET_DISPLAY_SIZE signals all errors; no condition values are returned.

Arguments

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name string. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify SYS\$WORKSTATION as the device name character string.

retwidth, reheight

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

VAXstation display screen size. The **retwidth** and **reheight** arguments are the addresses of **f_floating** point longwords that receive the physical display screen width and height in centimeters.

retresolx, retresoly

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

18-124 **UIS Routine Descriptions**
UIS\$GET_DISPLAY_SIZE

VAXstation display screen resolution. The **retresolx** and **retresoly** arguments are the addresses of *f*-floating point longwords that receive the *x* and *y* resolution in pixels per centimeters.

retpwidth, retpheight

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **write only**

mechanism: **by reference**

VAXstation screen size in pixels. The **retpwidth** and **retpheight** arguments are the addresses of integer longwords that receive the width and height of the screen in pixels.

Description

The height and width dimensions can be used when deciding the size of a virtual display or viewport. The resolution values can be used when it is important for the application to determine the exact physical size (or world coordinate dimensions) that map to a single pixel.

Screen Output

```
⌘ run get_display
Display screen characteristics
width = 33.58 cm height = 28.34 cm
x resolution = 30.49 pixels/cm
y resolution = 30.49 pixels/cm
width = 1024 pixels height = 864 pixels
FORTRAN PAUSE
⌘ █
```

UIS\$GET_FILL_PATTERN

Returns the index of the fill pattern.

Format

status=**UIS\$GET_FILL_PATTERN** *vd_id, atb* [,*index*]

Returns

VMS Usage: **boolean**
type: **longword**
access: **write only**
mechanism: **by value**

Boolean value returned as the filling mode in a status variable or R0 (VAX MACRO). The boolean TRUE is returned if filling is enabled, otherwise the boolean value is FALSE.

UIS\$GET_FILL_PATTERN signals all errors; no condition values are returned.

Arguments

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

atb
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The ***atb*** argument is the address of a longword integer that identifies the attribute block from which the fill pattern index is obtained.

index

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **write only**

mechanism: **by reference**

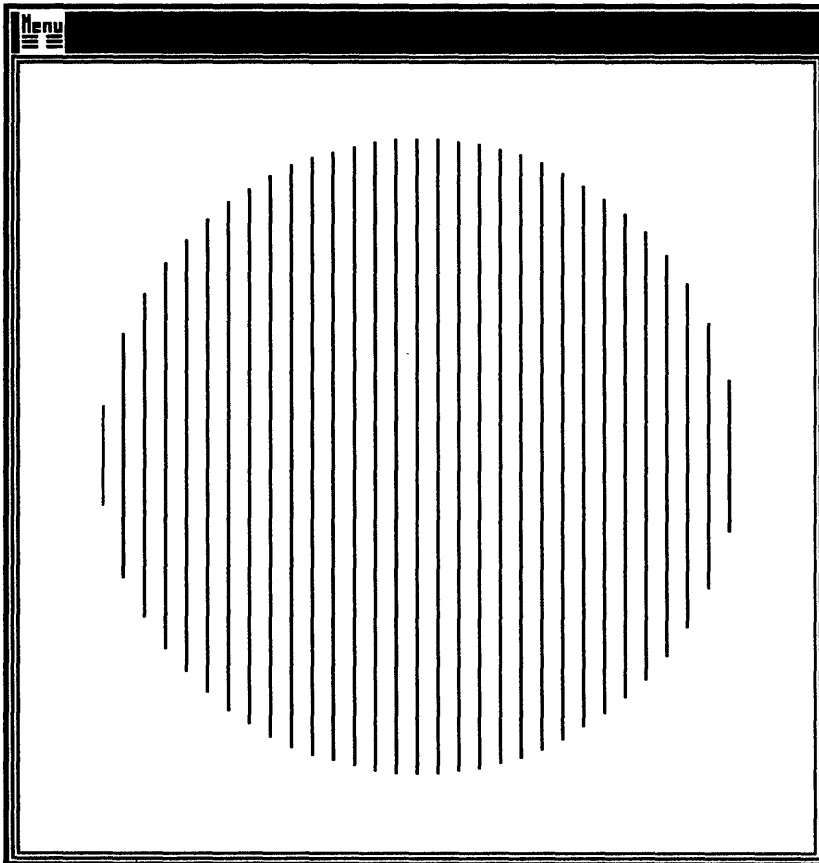
Index of the fill pattern. The **index** argument is the address of a longword that receives the value of the fill pattern symbol index. This is the index of a glyph in a fill pattern font.

18-128 UIS Routine Descriptions
UIS\$GET_FILL_PATTERN

Screen Output

```

$ run get_fill
Are fill patterns enabled? F = FALSE T = TRUE
T
What is the index of the current fill pattern?
  7
FORTRAN PAUSE
$ █
```



UIS\$GET_FONT

Returns the name of font file.

Format

UIS\$GET_FONT *vd_id, atb, bufferdesc* [*,length*]

Returns

UIS\$GET_FONT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies the attribute block from which the font file name is obtained.

bufferdesc

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Font file name string. The **bufferdesc** argument is the address of a character string descriptor of a location that receives the font file name character string.

18-130 UIS Routine Descriptions
UIS\$GET_FONT

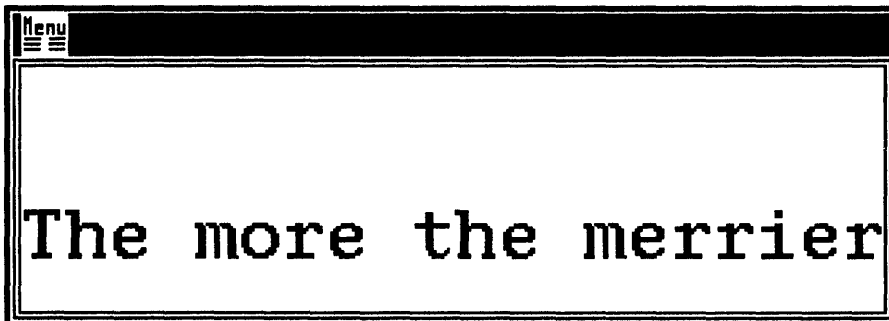
length

VMS Usage: **word_signed**
type: **word (signed)**
access: **write only**
mechanism: **by reference**

Length of the font file character string. The **length** argument is the address of a word that receives the length of font file name character string.

Screen Output

```
⌘ run get_fontname  
font name is DTABER0R07SK00GG0001UZZZZ02A000  
length of font name is      31 characters  
FORTRAN PAUSE  
⌘ █
```



UIS\$GET_FONT_ATTRIBUTES

Returns information about the ascender, descender, height, width, and font parameters.

Format

UIS\$GET_FONT_ATTRIBUTES *font_id, ascender,
descender, height
[,maximum_width] [item_list]*

Returns

UIS\$GET_FONT_ATTRIBUTES signals all errors; no condition values are returned.

Arguments

font_id

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Font file name. The **font_id** argument is the address of a string descriptor of the font file name only. UIS searches the directory SYS\$FONT for the correct file type.

ascender

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Character ascender. The **ascender** argument is the address of a longword that receives the distance between the font baseline and the top of the character cell in pixels.

descender

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

18-132 **UIS Routine Descriptions**
UIS\$GET_FONT_ATTRIBUTES

Character descender. The **descender** argument is the address of a longword that receives the distance between the font baseline and the bottom of the character cell in pixels.

height

VMS Usage: **longword_unsigned**
type: **longowrd (unsigned)**
access: **write only**
mechanism: **by reference**

Height of the character cell. The **height** argument is the address of a longword that receives the height of the character cell in pixels.

maximum_width

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

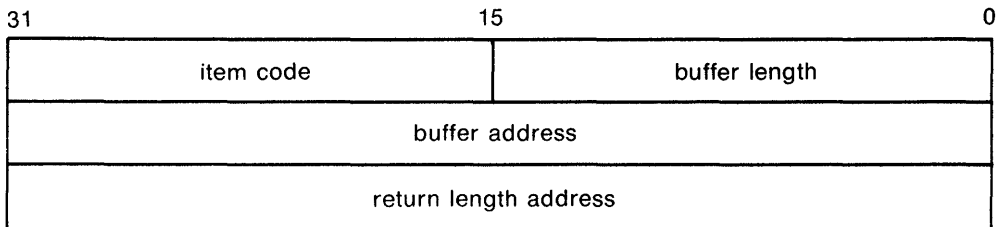
Maximum width of a character cell. The **average_width** argument is the address of a longword that receives the maximum width of a character cell in the font in pixels.

item_list

VMS Usage: **item_list_3**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Item list specifying additional font information to be returned. The **item_list** argument is the address of a list of item descriptors, each of which describes an item of information. A longword value of 0 terminates the list of item descriptors.

The structure of the item list is described in the following figure.



The following table lists valid item codes.

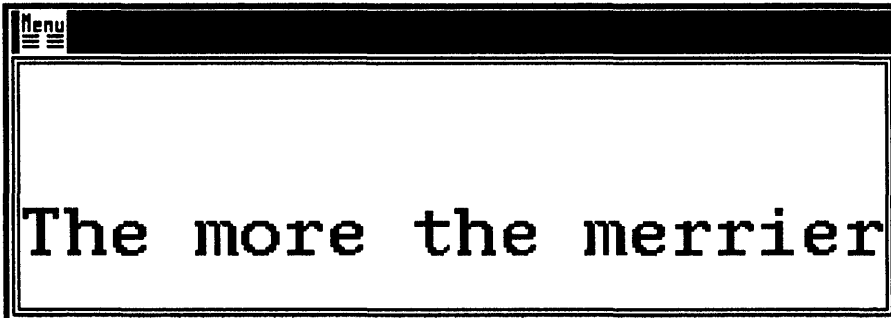
Item Code	Information Returned
Character Information	
UIS\$_FNT_FIRST_CHAR	First character in the font
UIS\$_FNT_LAST_CHAR	Last defined character in the font
UIS\$_FNT_GUTPERPIX_X	x resolution of the font in gutenbergs per pixel
UIS\$_FNT_GUTPERPIX_Y	y resolution of the font in gutenbergs per pixel
UIS\$_FNT_AVERAGE_GUT ¹	Average width of a character in the font
UIS\$_FNT_WIDTH	Width in pixels of all glyphs in the font, if the font is monospaced. A zero is returned, if the font is proportionally spaced.
Font Flags²	
UIS\$_FNT_FIXED	True, if the font is monospaced False, if the font is proportionally spaced.
UIS\$_FNT_CELLEQRAST	True, if the cell width of all glyphs in the font equals the width the glyph's raster.
UIS\$_FNT_VA_FONT	True, if this is a VA font.
Font Name	
UIS\$_FNT_FONT_ID	Font identifier string
¹ The font designer assigns this number. Although, the graphics subsystem copies the number, no interpretation is applied to it. UIS does not use the number. ² The value 1 is returned, if TRUE, and 0, if FALSE.	

18-134 UIS Routine Descriptions
UIS\$GET_FONT_ATTRIBUTES

Screen Output

```

$ run get_fontattr
font name is DTABEROR07SK00GG0001UZZZZO2A000
length of font name is      31 characters
FORTRAN PAUSE
$ cont
length of ascender          26 pixels
length of descender         4 pixels
height of character cell    30 pixels
FORTRAN PAUSE
$ █
```



UIS\$GET_FONT_SIZE

Obtains the size of a character or string of characters in the specified font in physical dimensions.

Format

UIS\$GET_FONT_SIZE *fontid, text_string, retwidth, retheight*

Returns

UIS\$GET_FONT_SIZE signals all errors; no condition values are returned.

Arguments

fontid

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Font identifier. The **fontid** argument is the address of a character string descriptor of a font file name. Specify only the font file name. UIS searches the directory SYS\$FONT for the correct file type.

text_string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Text string. The **text_string** argument is the address of a descriptor of a character or character string.

retwidth, retheight

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

String width and height. The **retwidth** and **retheight** arguments are the addresses of **f_floating** point longwords that receive the width and height of the character or character string in centimeters.

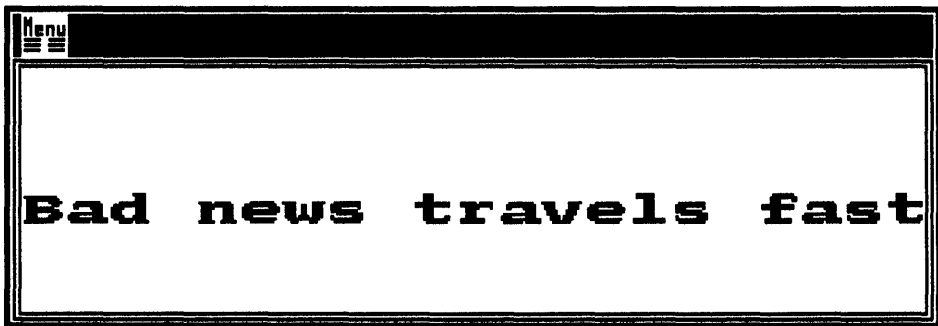
18-136 **UIS Routine Descriptions**
UIS\$GET_FONT_SIZE

Description

UIS\$GET_FONT_SIZE can be used to determine the proper size of a display viewport based on the size of the characters in a given font.

Screen Output

```
UIS$ run get_fontsize
string length = 11.01970 cm
character height = 0.4919507 cm
FORTRAN PAUSE
UIS$ █
```



UIS\$GET_HW_COLOR_INFO

Returns information about the hardware color map.

Format

UIS\$GET_HW_COLOR_INFO *devnam* [,*type*] [,*indices*]
[,*colors*] [,*maps*] [,*rbits*] [,*gbits*]
[,*bbits*] [,*ibits*] [,*res_indices*]
[,*regen*]

Returns

UIS\$GET_HW_COLOR_INFO signals all errors; no condition values are returned.

Arguments

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify SYS\$WORKSTATION in the **devnam** argument.

type

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Device type. The **type** argument is the address of a longword that receives the device type. The following table lists device type values.

18-138 **UIS Routine Descriptions**
UIS\$GET_HW_COLOR_INFO

Device Type	Value	Possible Colors
Monochrome	UIS\$_DEV_MONO	Black and white
Intensity	UIS\$_DEV_INTENSITY	Up to 2 ²⁴ gray tones
Color	UIS\$_DEV_COLOR	Up to 2 ²⁴ chromatic colors

indices

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of entries or simultaneous colors. The **indices** argument is the address of longword that receives the number of entries or simultaneous colors in the hardware color map.

colors

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of possible colors. The **colors** argument is the address of a longword that receives the number of possible colors represented in the color map. For example monochrome equals 2.

maps

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of hardware color maps. The **maps** argument is the address of a longword that receives the number of hardware color maps.

rbits

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of binary bits of precision for red. The **rbits** argument is the address of a longword that receives the number of binary bits of precision for the color red.

gbits

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of binary bits of precision for green. The **gbits** argument is the address of a longword that receives the number of binary bits of precision for the color green.

bbits

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of binary bits of precision for blue. The **bbits** argument is the address of a longword that receives the number of binary bits of precision for the color blue.

ibits

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of binary bits of precision for intensity. The **ibits** argument is the address of a longword that receives the number of binary bits of precision for intensity.

res_indices

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number entries in the hardware color map reserved for special use. The **res_indices** argument is the address of a longword that receives the number entries in the hardware color map reserved for special use.

regen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Color regeneration characteristics. The **regen** argument is the address of a longword that receives the color regeneration characteristics. The **regen**

18-140 **UIS Routine Descriptions**
UIS\$GET_HW_COLOR_INFO

argument indicates whether the color and intensity changes affect previously drawn display objects that specified the same color index in the hardware look up table. The following symbols are valid values: UIS\$C_DEV_RETRO or UIS\$C_DEV_NONRETRO.

The following table summarizes regeneration characteristics of direct and mapped color systems.

System	Regeneration Characteristics
Direct color	Usually sequential
Mapped color	Usually retroactive

UIS\$GET_INTENSITIES

Returns intensity values associated with one or more entries in the virtual color map.

Format

UIS\$GET_INTENSITIES *vd_id, index, count, reti_vector*
[,wd_id]

Returns

UIS\$GET_INTENSITIES signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Starting color map index. The **index** argument is the address of a longword that specifies the index of the first color map entry to be returned. If the specified index exceeds the maximum index of the virtual color map, an error is signaled.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

18-142 UIS Routine Descriptions

UIS\$GET_INTENSITIES

Number of indices. The **count** argument is the address of a longword that specifies the total number of color map entries to be returned including the starting index. If the specified count exceeds the maximum number of virtual color map entries, an error is signaled.

reti_vector

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Intensity values. The **reti_vector** argument is the address of an array of **f_floating** point longwords that receives the intensity values. Each intensity value is in the range of *0.0* to *1.0*, inclusively.

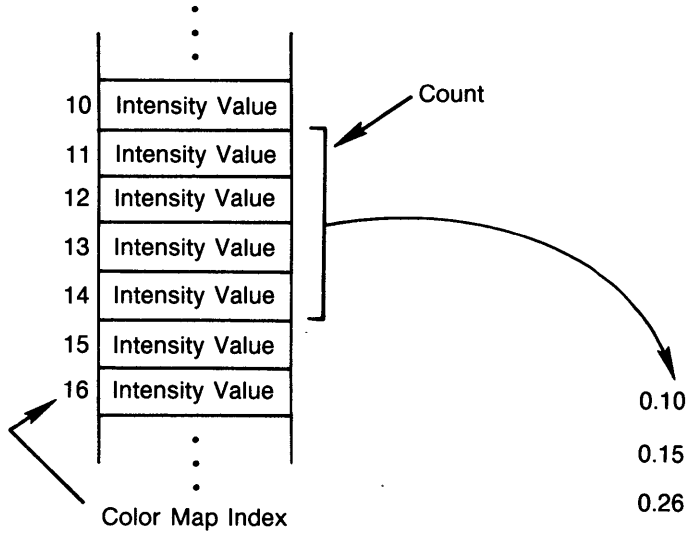
wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. If the **wd_id** argument is specified, it must be a valid display window identifier associated with the virtual display. The returned values are the *realized* intensities for the specific device for which the display window was created. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

If the **wd_id** argument is not specified, the intensity values returned are *set* color values originally established by a call to **UIS\$SET_INTENSITY** or **UIS\$SET_INTENSITIES**.

Illustration



UIS\$GET_INTENSITY

Returns the intensity value associated with a single entry in the color map.

Format

UIS\$GET_INTENSITY *vd_id, index, reti, [,wd_id]*

Returns

UIS\$GET_INTENSITY signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Color map index. The **index** argument is the address of a longword integer that identifies the index of an entry in the color map associated with the virtual display. If the specified index exceeds the maximum number of indices in the virtual color map, an error is signaled.

reti

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Intensity value. The **reti** argument is the address of an **f_floating** point longword that receives the intensity value. The intensity value is in the range of 0.0 to 1.0, inclusive.

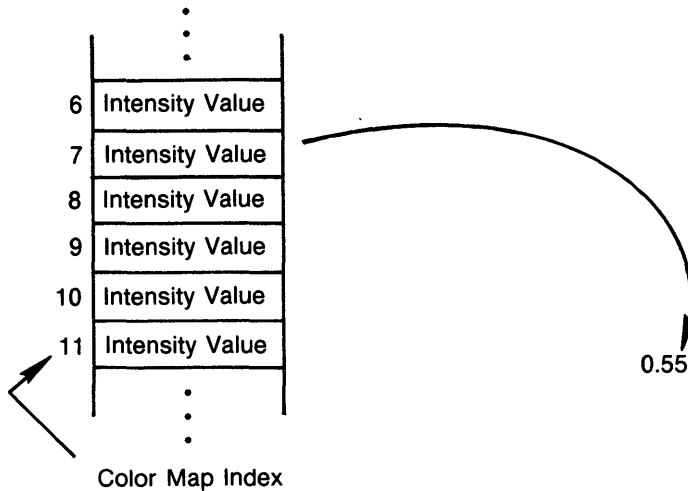
wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument. If this argument is specified, it must be a valid **wd_id** associated with the virtual display, and the returned values are the *realized* intensities for the specific device for which the window was created.

If the **wd_id** argument is not specified, the returned intensity values are *set* intensity originally established by a call to UIS\$SET_INTENSITY or UIS\$SET_INTENSITIES.

Illustration



Disabled keyboard characteristics. The **disable_items** argument is the address of a longword mask that receives the bit mask of the disabled keyboard characteristics.

click_volume

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Key click volume level. The **click_volume** argument is the address of a longword that receives the key click volume level. The key click volume is in the range of 1 to 8, inclusively, where 1 is quiet and 8 is loud.

Description

The enable and disable item lists are longword masks containing bits designating the characteristics to be enabled or disabled. The valid bits in the keyboard characteristics enable and disable masks are:

Symbol	Description ¹
UIS\$M_KB_AUTORPT	Enable/disable keyboard autorepeat
UIS\$M_KB_KEYCLICK	Enable/disable keyboard keyclick
UIS\$M_KB_UDF6	Enable/disable up button transitions for F6 to F10 keys
UIS\$M_KB_UDF11	Enable/disable up button transitions for F11 to F14 keys
UIS\$M_KB_UDF17	Enable/disable up button transitions for F17 to F20 keys
UIS\$M_KB_HELPDO	Enable/disable up button transitions for HELP and DO keys
UIS\$M_KB_UDE1	Enable/disable up button transitions for E1 to E6 keys
UIS\$M_KB_ARROW	Enable/disable up button transitions for arrow keys
UIS\$M_KB_KEYPAD	Enable/disable up button transitions for numeric keypad keys

¹By default down button transitions are enabled.

UIS\$GET_LINE_STYLE

Returns the line style patterns.

Format

style=**UIS\$GET_LINE_STYLE** *vd_id*, *atb*

Returns

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the line style bit vector in the variable *style* or R0 (VAX MACRO).

UIS\$GET_LINE_STYLE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

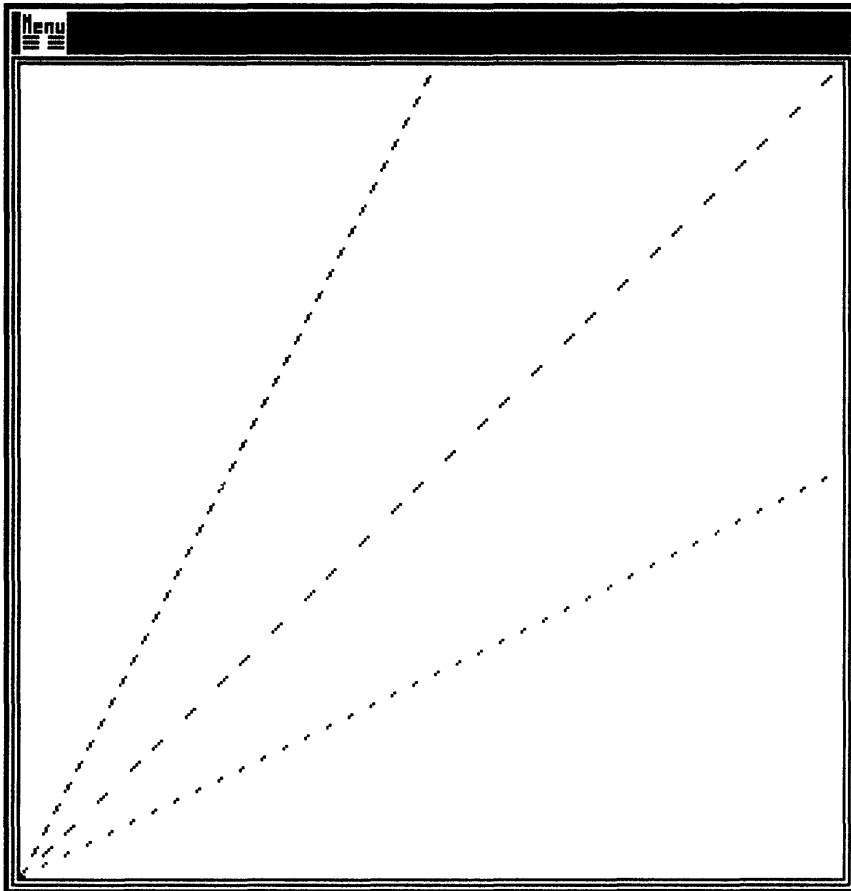
Attribute block number. The *atb* argument is the address of a longword integer that identifies an attribute block from which the line style pattern or bit vector is obtained.

Screen Output

```

$ run get_linestyle
line no.1 style = F0F0F0F0
line no.2 style = F00F00F0
line no.3 style = C0C0C0C0
FORTRAN PAUSE
$ █

```



UIS\$GET_LINE_WIDTH

Returns the line width.

Format

width = **UIS\$GET_LINE_WIDTH** *vd_id*, *atb* [*mode*]

Returns

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by value**

F_floating point value returned as the line width in the variable *width* or R0 (VAX MACRO).

UIS\$GET_LINE_WIDTH signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The ***atb*** argument is the address of a longword integer that identifies the attribute block from which the line width is obtained.

UIS\$GET_LINE_WIDTH***mode***

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

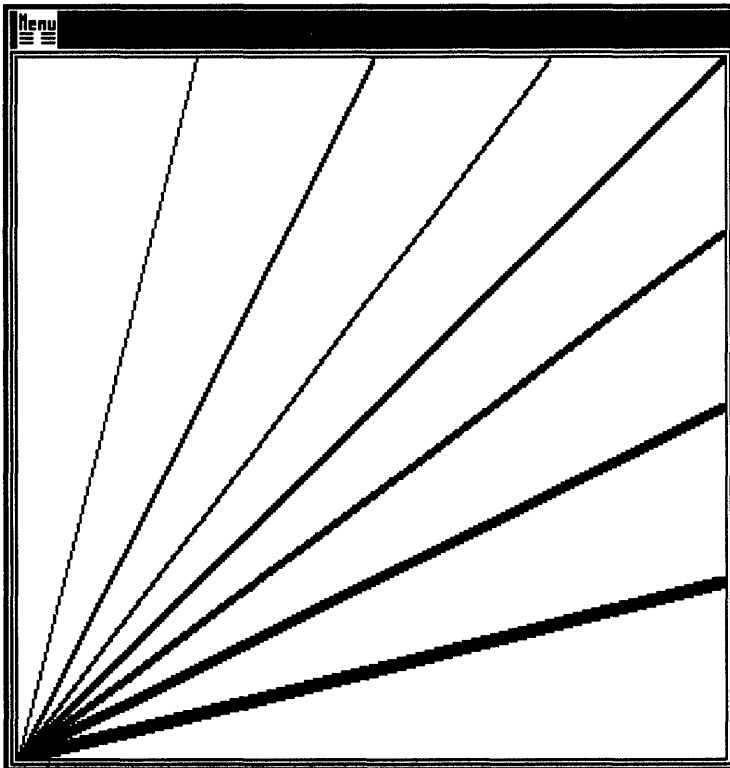
Line width mode. The optional **mode** argument is the address of a longword that receives the line width specification mode (WDPL\$_WIDTH_WORLD or WDPL\$_WIDTH_PIXELS). If WDPL\$_WIDTH_WORLD is returned, the line width is interpreted as world coordinates. If WDPL\$_WIDTH_PIXELS is returned, the line width is interpreted as pixels.

18-152 UIS Routine Descriptions
 UIS\$GET_LINE_WIDTH

Screen Output

```

$ run get_linewidth
line width = 1.00 pixels
line width = 2.00 pixels
line width = 2.00 pixels
line width = 3.00 pixels
line width = 4.00 pixels
line width = 5.00 pixels
line width = 6.00 pixels
FORTRAN PAUSE
$ █
```



UIS\$GET_NEXT_OBJECT

Returns the identifier of the next object in the display list.

Format

next_id=UIS\$GET_NEXT_OBJECT { *obj_id*
 seg_id } [*flags*]

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Longword value returned as the next object identifier in the variable *next_id* or R0 (VAX MACRO). The next object identifier uniquely identifies the next specified object in the display list and is used as an argument in other routines.

UIS\$GET_NEXT_OBJECT signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The *obj_id* argument is the address of a longword that uniquely identifies the object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The *seg_id* argument is the address of a longword that uniquely identifies the segment.

18-154 **UIS Routine Descriptions**
UIS\$GET_NEXT_OBJECT

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Flags. The **flags** argument is the address of a longword that controls how the display list is searched. If the **flags** argument is set using **UIS\$M_DL_SAME_SEGMENT**, the next object in the segment containing the object specified is returned.

If the **flags** argument is omitted, the next object in the display list, regardless of the segment in which it is contained, is returned.

Description

If a zero is returned, the next object was not found.

18-156 **UIS Routine Descriptions**
UIS\$GET_OBJECT_ATTRIBUTES

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies the object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

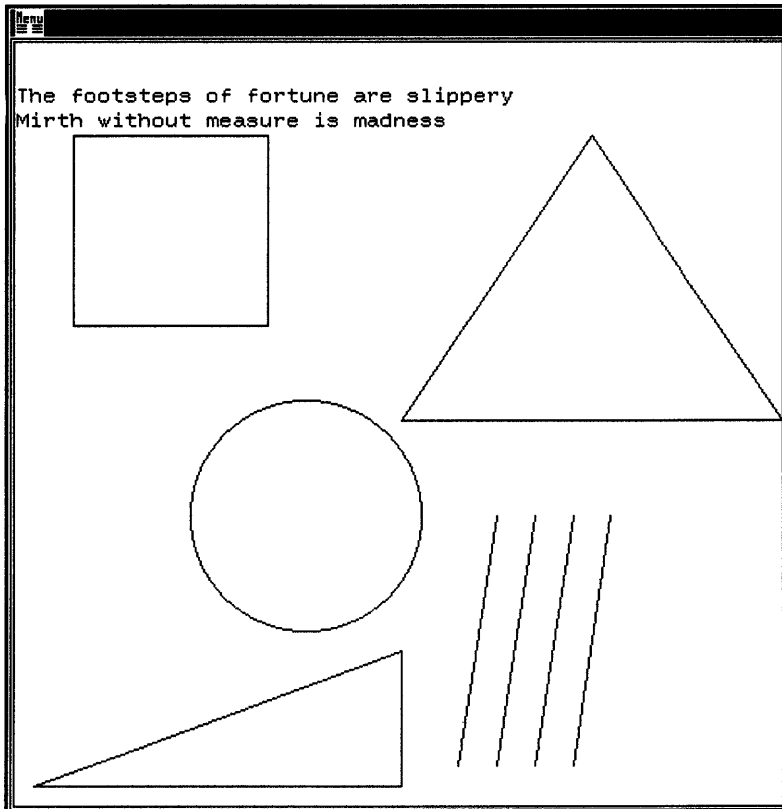
extent

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **write only**
mechanism: **by reference**

World coordinates of the extent rectangle. The **extent** argument is the address of an array of four longwords that receives the values of the world coordinates of the lower-left corner and the upper-right corner of the extent rectangle containing the object.

Screen Output

```
⌘ RUN WALK  
DISPLAY LIST ELEMENTS  
-----  
IDENTIFIER      OBJECT TYPE  
113992          UIS$C_OBJECT_SEGMENT  
115328          UIS$C_OBJECT_ELLIPSE  
115575          UIS$C_OBJECT_PLOT  
115822          UIS$C_OBJECT_PLOT  
116069          UIS$C_OBJECT_PLOT  
116316          UIS$C_OBJECT_TEXT  
116810          UIS$C_OBJECT_TEXT  
117057          UIS$C_OBJECT_LINE  
FORTRAN PAUSE  
⌘
```



Description

If the specified object is the outermost segment or root segment, its own object identifier is returned.

UIS\$GET_POINTER_POSITION

Returns the current pointer position in world coordinates.

Format

status = **UIS\$GET_POINTER_POSITION** *vd_id*, *wd_id*, *retx*,
rety

Returns

VMS Usage: **boolean**
type: **longword**
access: **write only**
mechanism: **by value**

Boolean value returned as the current position of the pointer in a status variable. **UIS\$GET_POINTER_POSITION** returns the boolean TRUE value 1 if the pointer is within the visible portion of the viewport, 0 is returned if the pointer is outside the visible portion of the viewport. In the latter case, the x and y values are returned as 0,0.

UIS\$GET_POINTER_POSITION signals all errors; no condition values are returned.

Arguments

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See **UIS\$CREATE_DISPLAY** for more information about the *vd_id* argument.

wd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

retx, rety

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

World coordinate pair. The **retx** and **rety** arguments are the addresses of **f_floating** point longwords that receives the pointer x and y world coordinates.

Description

Note that the returned status value should always be tested when using this routine, since it is always possible that the pointer could be outside the window when the service is called and the x,y values would be meaningless.

UIS\$GET_POSITION

Returns the current baseline position for text output.

Format

UIS\$GET_POSITION *vd_id, retx, rety*

Returns

UIS\$GET_POSITION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

retx, rety

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

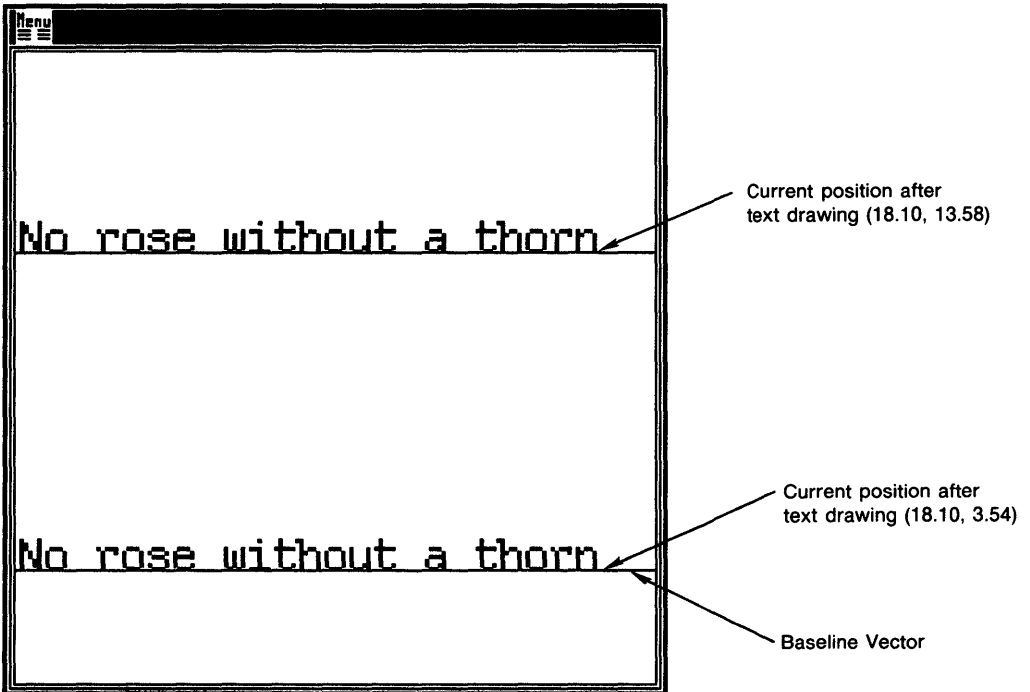
World coordinate pair. The **retx** and **rety** arguments are addresses of **f_floating** point longwords that receive the x and y world coordinate positions.

Description

UIS\$TEXT and UIS\$NEW_TEXT_LINE recognize the concept of current position. The position refers to the alignment point on the baseline of the next character to be output.

Screen Output

```
$ run get_pos  
What is the current text position in world coordinates?  
x coordinate = 18.10  
y coordinate = 13.58  
What is the current text position in world coordinates?  
x coordinate = 18.10  
y coordinate = 3.54  
FORTRAN PAUSE  
$
```



flags

VMS Usage: **mask_longword**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Flags. The **flags** argument is the address of a longword that controls how the display list is searched. If the **flags** argument is specified using UIS\$M_DL_SAME_SEGMENT, the previous object in the segment containing the object specified is returned.

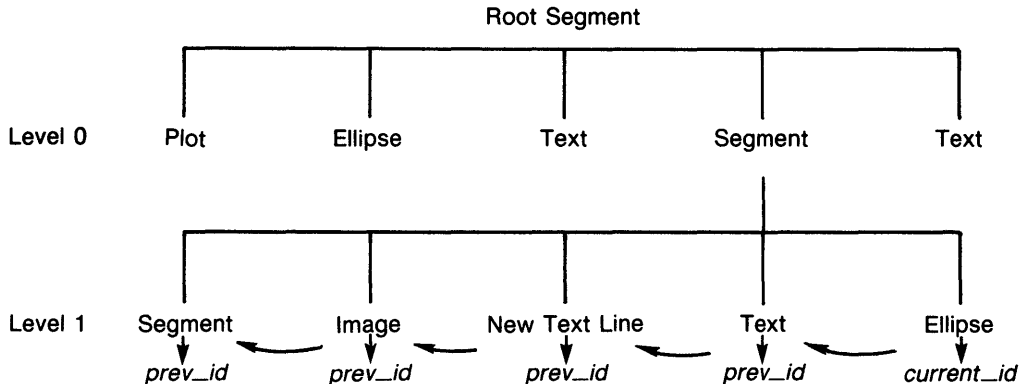
If the **flags** argument is omitted, the previous object in the display list, regardless of the segment in which it is contained, is returned.

Description

If no previous object is found, a zero is returned.

Illustration

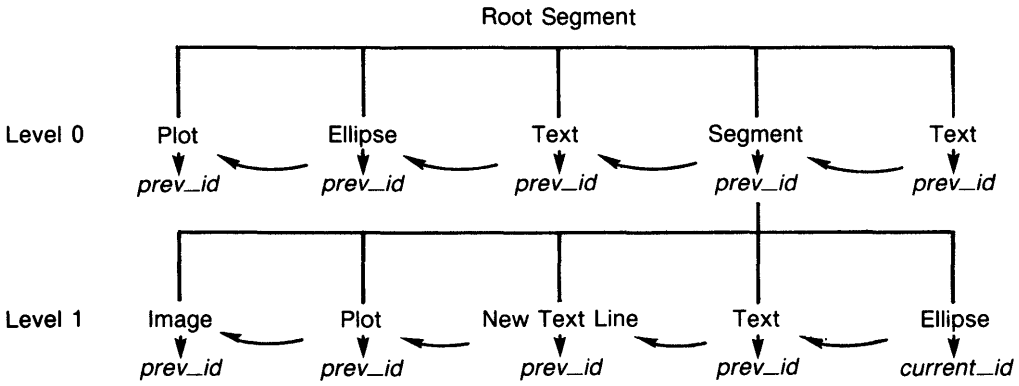
The following figure illustrates how UIS\$GET_PREVIOUS_OBJECT of each previous object within the same segment.



ZK-5363-86

The following figure illustrates how UIS\$GET_PREVIOUS_OBJECT returns the object identifier of all objects in the display list.

UIS Routine Descriptions
UIS\$GET_PREVIOUS_OBJECT



UIS\$GET_ROOT_SEGMENT

Returns the root segment of the specified virtual display.

Format

root_id=UIS\$GET_ROOT_SEGMENT *vd_id*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the root segment identifier in the variable *root_id* or R0 (VAX MACRO). The root segment identifier uniquely identifies the root segment.

UIS\$GET_ROOT_SEGMENT signals all errors; no condition values are returned.

Argument

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

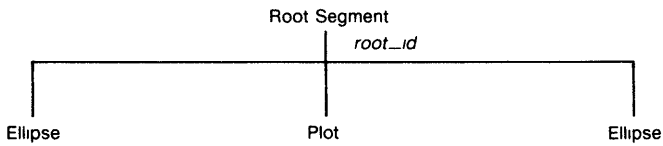
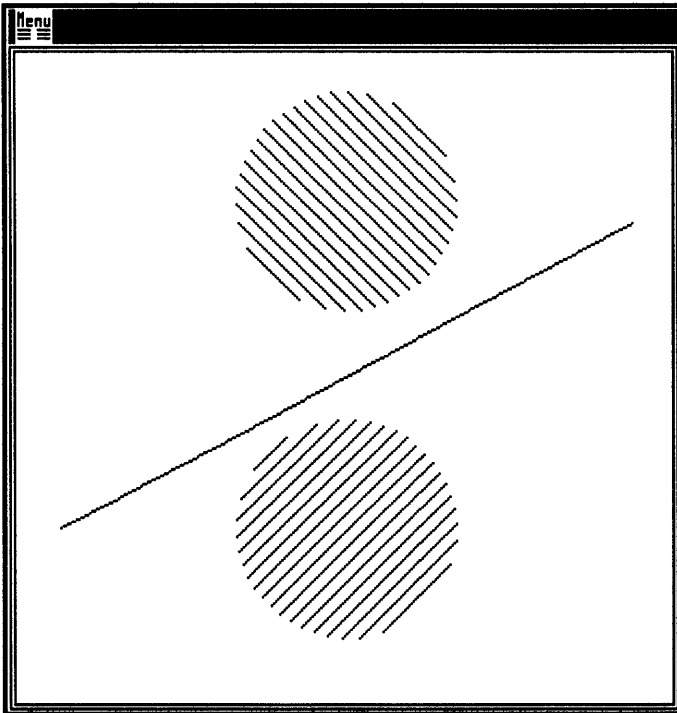
Description

UIS\$GET_ROOT_SEGMENT can be used with UIS\$EXTRACT_OBJECT to extract an entire display list.

18-168 **UIS Routine Descriptions**
UIS\$GET_ROOT_SEGMENT

Screen Output

```
UIS$ run get_rootseg  
The root segment identifier for virtual display is 112968  
FORTRAN PAUSE  
UIS$ █
```



UIS\$GET_TB_INFO

Returns the characteristics of the tablet device.

Format

status=UIS\$GET_TB_INFO *devnam, retwidth, retheight, retresolx, retresoly* [,*retpwidth, retpheight*]

Returns

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Longword value returned in a status variable. If the value 1 is returned, the pointing device is a tablet. If the value 0 is returned, the pointing device is a mouse and the returned information will be zeros. A tablet is required for digitizing.

UIS\$GET_TB_INFO signals all errors; no condition values are returned.

Arguments

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name. The **devname** argument is the address of a character string descriptor of the workstation device name. Specify the logical name SYS\$WORKSTATION.

retwidth

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Tablet width. The **retwidth** argument is the address of an **f_floating** point longword that receives the width of the tablet in centimeters.

18-170 UIS Routine Descriptions

UIS\$GET_TB_INFO

retheight

VMS Usage: **floating_point**

type: **f_floating**

access: **write only**

mechanism: **by reference**

Tablet height. The **retheight** argument is the address of an **f_floating** point longword that receives the height of the tablet in centimeters.

retresolx

VMS Usage: **floating_point**

type: **f_floating**

access: **write only**

mechanism: **by reference**

Tablet *x* resolution. The **retresolx** argument is the address of an **f_floating** longword that receives the *x* resolution of the tablet in centimeters per pixel.

retresoly

VMS Usage: **floating_point**

type: **f_floating**

access: **write only**

mechanism: **by reference**

Tablet *y* resolution. The **retresoly** argument is the address of an **f_floating** point longword that receives the *y* resolution of the tablet in centimeters per pixel.

retpwidth

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **write only**

mechanism: **by reference**

Tablet width. The **retpwidth** argument is the address of a longword that receives the width of the tablet in pixels.

retpheight

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **write only**

mechanism: **by reference**

Tablet height. The **retpheight** argument is the address of a longword that receives the height of the tablet in pixels.

Description

A call to UIS\$GET_TB_INFO is recommended prior to establishing digitizing. UIS\$GET_TB_INFO returns a value indicating whether the device is a mouse or tablet. A tablet is required for digitizing.

Note that you may invalidate the results of this call, if you unplug the tablet and replace it with a mouse while running an application.

UIS\$GET_TB_POSITION

Polls for the position of the pointing device on the tablet.

Format

UIS\$GET_TB_POSITION *tb_id ,retx ,rety*

Returns

UIS\$GET_TB_POSITION signals all errors; no condition values are returned.

Arguments

tb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Tablet identifier. The **tb_id** argument is the address of a longword that uniquely identifies the tablet. See UIS\$CREATE_TB for more information about the **tb_id** argument.

retx, rety

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Digitizer position. The **retx, rety** arguments are the addresses of **f_floating** numbers that define the current digitizer position.

Description

The digitizer's position will not be available if the pointing device is a mouse.

If the pointer is not on the tablet, UIS\$GET_TB_POSITION returns the last pointer that was reported.

UIS\$GET_TEXT_FORMATTING

Returns a mask describing the enabled text formatting modes.

Format

formatting = **UIS\$GET_TEXT_FORMATTING** *vd_id*, *atb*

Returns

VMS Usage: **mask_longword**
 type: **longword (unsigned)**
 access: **write only**
 mechanism: **by value**

Longword mask returned as the current formatting mode in the variable *formatting* or R0 (VAX MACRO). The following table lists the formatting modes.

Formatting Mode	Function
UIS\$_TEXT_FORMAT_LEFT	Left justified, ragged right (default)
UIS\$_TEXT_FORMAT_RIGHT	Right justified, left ragged
UIS\$_TEXT_FORMAT_CENTER	Centered line between left and right margin
UIS\$_TEXT_FORMAT_JUSTIFY	Justified lines, space filled to right margin
UIS\$_TEXT_FORMAT_NOJUSTIFY	No text justification

UIS\$GET_TEXT_FORMATTING signals all errors; no condition values are returned.

Arguments

vd_id
 VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

18-174 **UIS Routine Descriptions**
UIS\$GET_TEXT_FORMATTING

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block containing the text formatting attribute setting to be returned.

UIS\$GET_TEXT_MARGINS

Returns the text margins for a line of text.

Format

UIS\$GET_TEXT_MARGINS *vd_id ,atb ,x ,y [,margin_length]*

Returns

UIS\$GET_TEXT_MARGINS signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block containing the modified text margins attribute.

x,y

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Starting margin position. The **x,y** arguments are the addresses of **f_floating** longwords that receive the starting margin relative to the direction of text drawing.

18-176 **UIS Routine Descriptions**
UIS\$GET_TEXT_MARGINS


margin_length

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Ending margin position. The **margin_length** is the address of an **f_floating** longword that receives the distance to the end margin. The margin is measured along the actual path of text drawing in the direction of the major text path.

Screen Output

```
⌘ run get_margins
margin settings
left margin x coordinate 5.00
left margin y coordinate 15.00
distance from left margin to right margin                    20.00
FORTRAN PAUSE
⌘
```

	<p>Hoist your sail when the wind is fair Hoist your sail when the wind is fair Hoist your sail when the wind is fair Hoist your sail when the wind is fair Hoist your sail when the wind is fair</p>	
---	---	--

UIS\$GET_TEXT_PATH

Returns text path types. See UIS\$SET_TEXT_PATH for information about valid text path types.

Format

UIS\$GET_TEXT_PATH *vd_id, atb [,major] [,minor]*

Returns

UIS\$GET_TEXT_PATH signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a number that identifies an attribute block containing the text path attribute setting to be returned.

major

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Major text path type. The **major** argument is the address of a code that identifies a major text path type. The major text path of text drawing is the direction of text drawing along a line.

18-178 **UIS Routine Descriptions**
UIS\$GET_TEXT_PATH

minor

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Minor text path type. The **minor** argument is the address of a code that identifies a minor text path type. The minor path of text drawing is the direction used for new text line creation.

Description

The following table contains symbols for valid character drawing directions.

Path	Direction
UIS\$_TEXT_PATH_RIGHT	Left to right (default major text path)
UIS\$_TEXT_PATH_LEFT	Right to left
UIS\$_TEXT_PATH_UP	Bottom to top
UIS\$_TEXT_PATH_DOWN	Top to bottom (default minor text path)

UIS\$GET_TEXT_SLOPE

Returns the angle of the actual path of text drawing relative to the major path in degrees.

Format

angle=UIS\$GET_TEXT_SLOPE *vd_id*, *atb*

Returns

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by value**

Longword value returned as the angle of the actual path of text drawing relative to the major path in degrees in the variable *angle* or R0 (VAX MACRO). Degrees are measured counterclockwise.

UIS\$GET_TEXT_SLOPE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

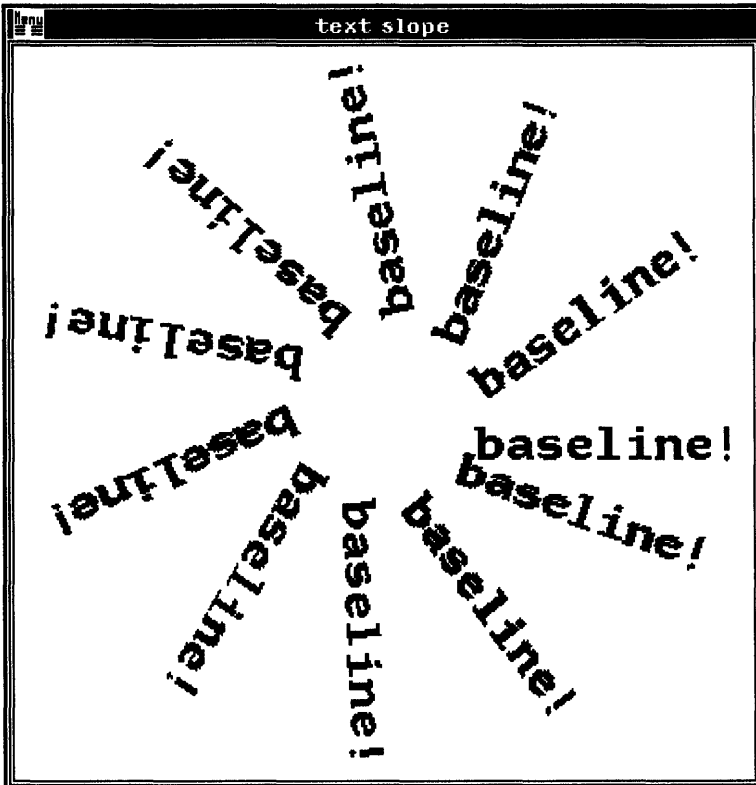
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a longword that identifies an attribute block from which the text slope attribute setting is to be returned.

18-180 UIS Routine Descriptions
UIS\$GET_TEXT_SLOPE

Screen Output

```
$ run get_slope
The angle of the text baseline is 0.00 degrees
The angle of the text baseline is 34.00 degrees
The angle of the text baseline is 68.00 degrees
The angle of the text baseline is 102.00 degrees
The angle of the text baseline is 136.00 degrees
The angle of the text baseline is 170.00 degrees
The angle of the text baseline is 204.00 degrees
The angle of the text baseline is 238.00 degrees
The angle of the text baseline is 272.00 degrees
The angle of the text baseline is 306.00 degrees
The angle of the text baseline is 340.00 degrees
FORTRAN PAUSE
$
```



UIS\$GET_VCM_ID

Returns the virtual color map identifier used by the specified virtual display.

Format

vcm_id=UIS\$GET_VCM_ID *vd_id*

Returns

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the virtual color map identifier in the variable *vcm_id* or R0 (VAX MACRO). The virtual color map identifier uniquely identifies a virtual color map for a specified virtual display. See UIS\$CREATE_COLOR_MAP for more information about the **vcm_id** argument.

UIS\$GET_VCM_ID signals all errors; no condition values are returned.

Argument

vd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

UIS\$GET_VIEWPORT_ICON

Returns boolean value indicating whether or not the icon is visible.

Format

boolean=UIS\$GET_VIEWPORT_ICON *wd_id* [*icon_wd_id*]

Returns

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Boolean value returned in a status variable or R0 (VAX MACRO) indicating whether an icon has replaced a viewport, a *1* denotes a TRUE condition; a *0* denotes a FALSE condition.

UIS\$GET_VIEWPORT_ICON signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

icon_wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Icon identifier. The *icon_wd_id* argument is the address of a longword that uniquely identifies the icon.

Screen Output

```
$ run window_options  
Is the icon is visible? F = FALSE T = TRUE  
T
```



UIS\$GET_VIEWPORT_POSITION

Returns the position of the lower-left corner of the display viewport relative to the lower-left corner of the screen.

Format

UIS\$GET_VIEWPORT_POSITION *wd_id, retx, rety*

Returns

UIS\$GET_VIEWPORT_POSITION signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE for more information about the **wd_id** argument.

retx, rety

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Absolute device coordinate pair. The **retx** and **rety** arguments are the addresses of **f_floating** point longwords that receive the *x* and *y* coordinates of the display viewport origin in centimeters.

These coordinates refer to the inside of the viewport and do not include the border.

Description

UIS\$GET_VIEWPORT_POSITION is useful in the exact placement of windows.

Screen Output

See UIS\$GET_VIEWPORT_SIZE.

UIS\$GET_VIEWPORT_SIZE

Returns the size of the display viewport associated with the specified display window.

Format

UIS\$GET_VIEWPORT_SIZE *wd_id, retwidth, retheight*

Returns

UIS\$GET_VIEWPORT_SIZE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

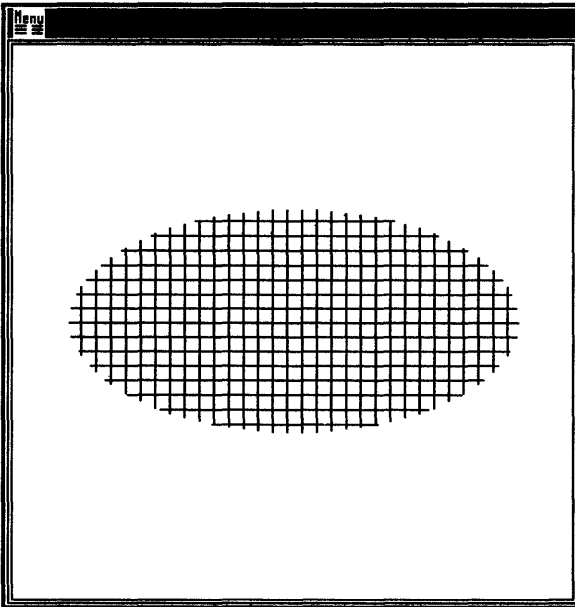
retwidth, retheight

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Display viewport width and height. The **retwidth** and **retheight** arguments are the addresses of **f_floating** point longwords that receive the display viewport width and height in centimeters.

Screen Output

```
$ run get_viewpos_size  
The viewport position on the display screen in absolute coordinates  
x coordinate = 12.86 cm y coordinate = 1.97cm  
The physical dimensions of the display viewport  
width of viewport 9.97 cm height of viewport 9.97 cm  
FORTRAN PAUSE  
$
```



UIS\$GET_VISIBILITY

Returns a boolean value that indicates whether or not the specified rectangle in the display window is visible.

Format

status = **UIS\$GET_VISIBILITY** *vd_id*, *wd_id* [*x*₁, *y*₁ [*x*₂, *y*₂]]

Returns

VMS Usage: **boolean**
type: **longword**
access: **write only**
mechanism: **by value**

Boolean value returned in a status variable or R0 (VAX MACRO). The returned value, the visibility status, is a boolean TRUE only if the entire area is visible, and a boolean FALSE if even a portion of the area is occluded or clipped.

UIS\$GET_VISIBILITY signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

x_1, y_1, x_2, y_2

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

World coordinates of a rectangle in the display window. The x_1 and y_1 arguments are addresses of **f_floating** point numbers that define the lower-left corner of a rectangle in the display window. The x_2 and y_2 arguments are addresses of **f_floating** point numbers that define the upper-right corner of a rectangle in the display window.

If the coordinates of the rectangle are not specified, the dimensions of the entire display window are used by default.

If only one point is specified, only that point is checked.

UIS\$GET_WINDOW_ATTRIBUTES

Returns the value of the mask `WDPL$C_ATTRIBUTES` used in the creation of the specified window. See `UIS$CREATE_WINDOW` for more information about window and viewport attributes.

Format

attributes = **UIS\$GET_WINDOW_ATTRIBUTES** *wd_id*

Returns

VMS Usage: **mask_longword**
type: **longword**
access: **write only**
mechanism: **by value**

Longword mask representing one or more attributes of the specified display window and returned in the variable *attributes* or `R0` (VAX MACRO). See `UIS$CREATE_WINDOW` for more information.

`UIS$GET_WINDOW_ATTRIBUTES` signals all errors; no condition values are returned.

Argument

wd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See `UIS$CREATE_WINDOW` for more information about the *wd_id* argument.

UIS\$GET_WINDOW_SIZE

Returns the dimensions of the display window.

Format

UIS\$GET_WINDOW_SIZE *vd_id, wd_id, x₁, y₁, x₂, y₂*

Returns

UIS\$GET_WINDOW_SIZE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about *vd_id*.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about *wd_id*.

x₁, y₁, x₂, y₂

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

World coordinate pairs. The *x₁, y₁* and the *x₂, y₂* arguments are the addresses of *f_floating* longwords that receive the locations of the lower-left and upper-right corners of the display window in world coordinates.

UIS\$GET_WRITING_INDEX

Returns the writing color index for text and graphics output.

Format

index=**UIS\$GET_WRITING_INDEX** *vd_id*, *atb*

Returns

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the color map index in the variable *index* or R0 (VAX MACRO).

UIS\$GET_WRITING_INDEX signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

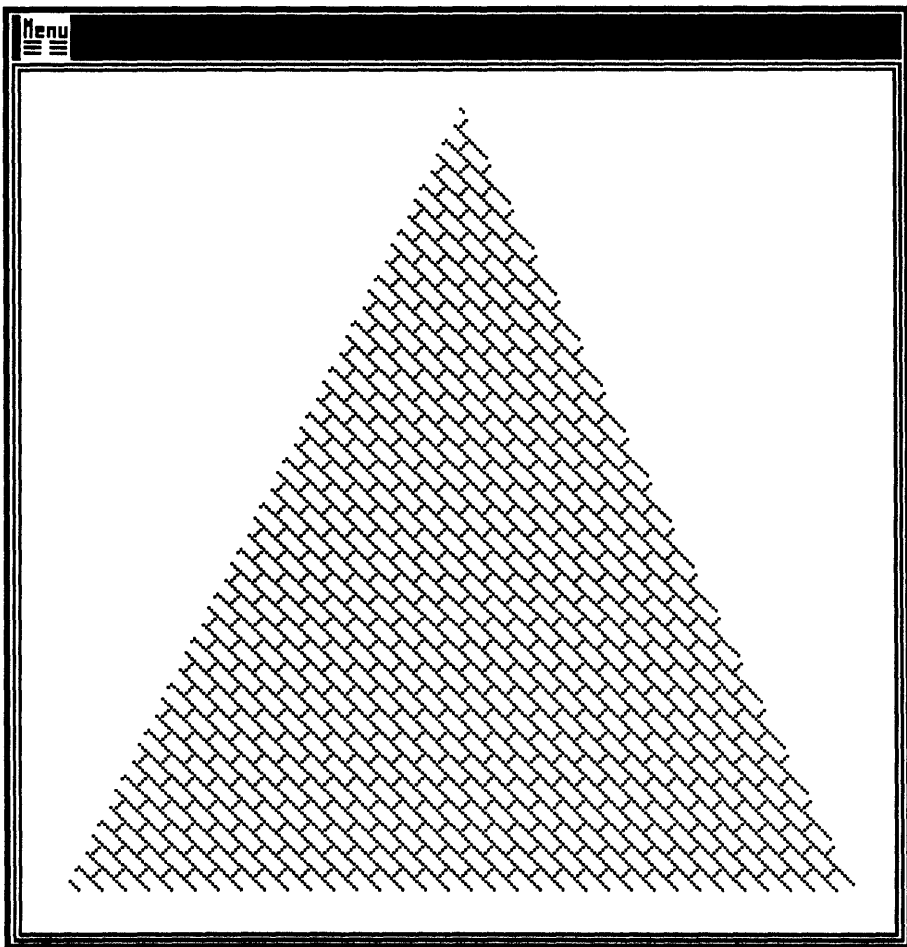
Attribute block number. The *atb* argument is the address of a longword integer that identifies an attribute block from which the writing color index is obtained.

Screen Output

```

$ run get_writindex
The current writing index is 1
FORTRAN PAUSE
$

```



UIS\$GET_WRITING_MODE

Returns the writing mode.

Format

mode=**UIS\$GET_WRITING_MODE** *vd_id*, *atb*

Returns

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as a UIS writing modes in the variable *mode* or R0 (VAX MACRO). See Table 9-2 for more information about writing modes.

UIS\$GET_WRITING_MODE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a longword integer that identifies an attribute block from which the writing mode is obtained.

UIS\$GET_WS_COLOR

Returns the R (red), G (green), and B (blue) values associated with the workstation standard color.

Format

UIS\$GET_WS_COLOR *vd_id, color_id, retr, retg, retb*
[,wd_id]

Returns

UIS\$GET_WS_COLOR signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

color_id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Workstation standard color. The **color_id** argument is the address of a longword integer that identifies a symbolic code for the workstation standard color. If the **color_id** argument is invalid, an error is signaled.

The following table lists possible workstation standard color symbols and their current values.

UIS\$GET_WS_COLOR

Standard Color	Symbol
Background	UIS\$C_WS_BCOLOR
Foreground	UIS\$C_WS_FCOLOR
Black	UIS\$C_WS_BLACK
White	UIS\$C_WS_WHITE
Red	UIS\$C_WS_RED
Green	UIS\$C_WS_GREEN
Blue	UIS\$C_WS_BLUE
Cyan	UIS\$C_WS_CYAN
Yellow	UIS\$C_WS_YELLOW
Magenta	UIS\$C_WS_MAGENTA
Grey (25%)	UIS\$C_WS_GREY25
Grey (50%)	UIS\$C_WS_GREY50
Grey (75%)	UIS\$C_WS_GREY75

retrVMS Usage: **floating_point**type: **f_floating**access: **write only**mechanism: **by reference**

Red value. The **retr** argument is the address of an **f_floating** point longword that receives the red value. The red value is in the range of *0.0* to *1.0*, inclusive.

retgVMS Usage: **floating_point**type: **f_floating**access: **write only**mechanism: **by reference**

Green value. The **retg** argument is the address of an **f_floating** point longword that receives the green value. The green value is in the range of *0.0* to *1.0*, inclusive.

retbVMS Usage: **floating_point**type: **f_floating**access: **write only**mechanism: **by reference**

UIS\$GET_WS_COLOR

Blue value. The **retb** argument is the address of an **f_floating** point longword that receives the blue value. The blue value is in the range of *0.0* to *1.0*, inclusive.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument. If this argument is specified, then it must be a valid **wd_id** associated with the virtual display, and the returned values are the realized colors for the specific device for which the window was created.

UIS\$GET_WS_INTENSITY

Returns the intensity values associated with a workstation standard color.

Format

UIS\$GET_WS_INTENSITY *vd_id, color_id, reti [,wd_id]*

Returns

UIS\$GET_WS_INTENSITY signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

color_id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Workstation standard color identifier. The **color_id** argument is the address of a longword that identifies a symbolic code for the workstation standard color. If the **color_id** argument is invalid, an error is signaled.

The following table lists possible workstation standard color symbols.

Standard Color	Symbol
Background	UIS\$C_WS_BCOLOR
Foreground	UIS\$C_WS_FCOLOR
Black	UIS\$C_WS_BLACK
White	UIS\$C_WS_WHITE
Red	UIS\$C_WS_RED
Green	UIS\$C_WS_GREEN
Blue	UIS\$C_WS_BLUE
Cyan	UIS\$C_WS_CYAN
Yellow	UIS\$C_WS_YELLOW
Magenta	UIS\$C_WS_MAGENTA
Grey (25%)	UIS\$C_WS_GREY25
Grey (50%)	UIS\$C_WS_GREY50
Grey (75%)	UIS\$C_WS_GREY75

reti

VMS Usage: **floating_point**
 type: **f_floating**
 access: **write only**
 mechanism: **by reference**

Intensity value. The **reti** argument is the address of an **f_floating** longword that receives the intensity value. The intensity value is in the range of 0.0 to 1.0, inclusive.

wd_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

If this argument is specified, then it must be a valid **wd_id** associated with the virtual display, and the returned values are the realized intensities for the specific device for which the window was created.

UIS\$HLS_TO_RGB

Converts color representation values of hue, lightness, and saturation (HLS) to red, green, and blue (RGB) values.

Format

UIS\$HLS_TO_RGB *H, L, S, retr, retg, retb*

Returns

UIS\$HLS_TO_RGB signals all errors; no condition values are returned.

Arguments

H

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Hue. The **H** argument is the address of an **f_floating** number that defines the hue of a color.

L

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Lightness. The **L** argument is the address of an **f_floating** number that defines the lightness of a color.

S

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Saturation. The **S** argument is the address of an **f_floating** number that defines color saturation.

retr

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Red value. The **retr** argument is the address of an **f_floating** point longword that receives the red value.

retg

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Green value. The **retg** argument is the address of an **f_floating** point longword that receives the green value.

retb

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Blue value. The **retb** argument is the address of an **f_floating** point longword that receives the blue value.

UIS\$HSV_TO_RGB

Converts color representation values of hue, saturation, and value (HSV) to red, green, and blue (RGB) values.

Format

UIS\$HSV_TO_RGB *H, S, V, retr, retg, retb*

Returns

UIS\$HSV_TO_RGB signals all errors; no condition values are returned.

Arguments

H

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Hue. The **H** argument is the address of an **f_floating** number that defines the hue of a color.

S

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Saturation. The **S** argument is the address of an **f_floating** number that defines the saturation of a color.

V

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Value. The **V** argument is the address of an **f_floating** number that defines the value of a color.

retr

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Red value. The **retr** argument is the address of an **f_floating** longword that receives the red color value.

retg

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Green value. The **retg** argument is the address of an **f_floating** longword that receives the green color value.

retb

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Blue value. The **retb** argument is the address of an **f_floating** longword that receives the blue color value.

UIS\$IMAGE

Draws a raster image in a specified rectangle in the display viewport.

Format

UIS\$IMAGE *vd_id, atb, x₁, y₁, x₂, y₂, rasterwidth,
rasterheight, bitsperpixel, rasteraddr*

Returns

UIS\$IMAGE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See **UIS\$CREATE_DISPLAY** for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies the image.

x₁, y₁, x₂, y₂

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the rectangle in the virtual display. The **x₁** and **y₁** arguments are the addresses of **f_floating** point numbers that define the lower-left corner of the rectangle in the virtual display. The **x₂** and **y₂**

arguments are the addresses of *f*-floating point numbers that define the upper-right corner of the rectangle in the virtual display.

rasterwidth

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Width of the raster image. The **rasterwidth** argument is the address of a longword that defines the width of the raster image in pixels.

rasterheight

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Height of the raster image. The **rasterheight** is the address of a longword that defines the height of the raster image in pixels.

bitsperpixel

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Number of bits per pixel in the raster image. The **bitsperpixel** argument is the address of a longword that defines the number of bits per pixel in the raster image. The **bitsperpixel** argument is currently required to be 1 or 8.

If the value 8 is specified for **bitsperpixel** on a single plane system, the results are unpredictable.

rasteraddr

VMS Usage: **vector_longword_unsigned**
 type: **longword_unsigned**
 access: **read only**
 mechanism: **by reference**

Bitmap image. The **rasteraddr** argument is the address of an array that defines a bitmap image. You must first create a bitmap by defining a data structure such as a record or array. When you assign values to the field or array element in the data structure, you are setting the bits of the image to be drawn by UIS\$IMAGE. See the Description section for information about setting bits.

Description

The bitmap image is drawn to the display viewport as a raster image. The raster image dimensions are described by the width, height, and bits per pixel parameters. The width and height give the number of pixels in each dimension, and bits per pixel represents the number of bits that makes up each pixel. The raster is read from memory as "height" bit vectors each of which is "width" pixels long and each pixel is "bits/pixel" bits long.

If the destination rectangle is larger than the raster size by at least an integer multiple, the raster is automatically scaled on a per pixel basis to the space available. Thus, a 1 x 1 raster can be written into an arbitrarily large destination rectangle, and the entire region is filled with the pattern.

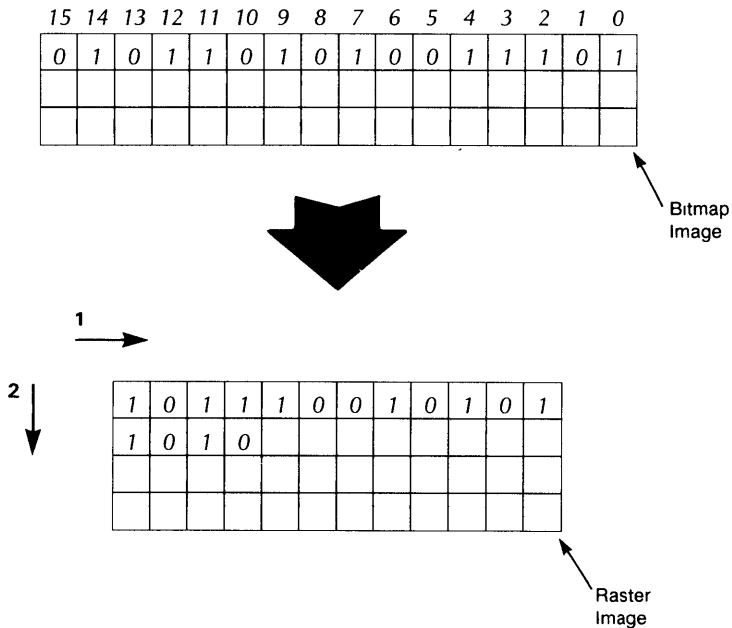
If the destination rectangle is not an exact multiple of the raster size, then the remaining space on the right and top will not be written.

The procedure for mapping values in the bitmap to the raster image is as follows:

1. Each bit in the raster is set from left-most bit to the right-most bit
2. Each row is filled from the top row to the bottom row.

NOTE: The raster image is not byte- or word-aligned.

The following figure illustrates the setting of bits in the bitmap.



ZK-4627 85

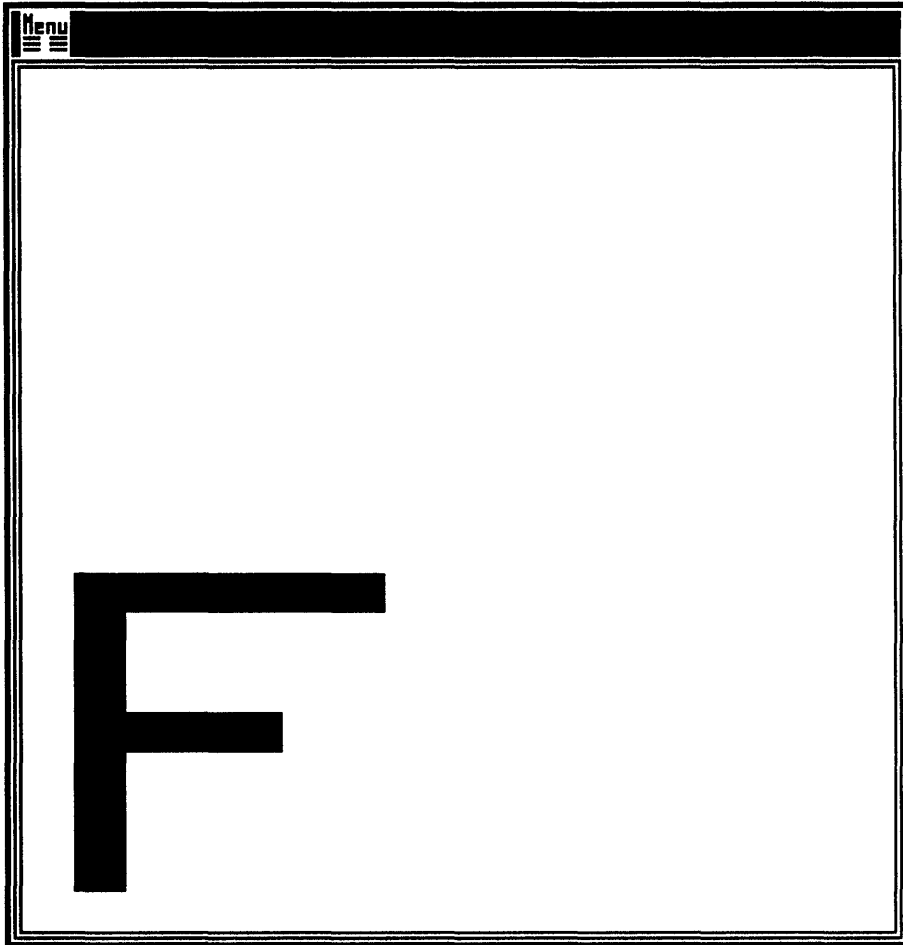
Example

```

INTEGER*2 BITMAP(20)
DATA BITMAP/2*0,2*16380,5*12,2*1020,7*12,2*0/
VD_ID=UIS$CREATE_DISPLAY(0.0,0.0,40.0,40.0,10.0,10.0)
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION')
CALL UIS$IMAGE(VD_ID,0,0.0,0.0,20.0,20.0,16,20,1,BITMAP)
  
```

18-208 UIS Routine Descriptions
UIS\$IMAGE

Screen Output



UIS\$INSERT_OBJECT

Inserts the specified object into the display list at the position specified by the insertion pointer. See UIS\$SET_INSERTION_POSITION for information about setting the pointer in the display list.

Format

UIS\$INSERT_OBJECT { *obj_id* }
 { *seg_id* }

Returns

UIS\$INSERT_OBJECT signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies an object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies a segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

UIS\$LINE

Draws an unfilled point, line, or series of unconnected lines depending on the number of positions specified.

Format

UIS\$LINE *vd_id, atb, x₁, y₁ [,x₂,y₂ [,...x_n,y_n]]*

Returns

UIS\$LINE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies line style and line width or both.

x, y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinate pair. The **x** and **y** arguments are the addresses of **f_floating** point numbers that define a point in the virtual display. If the arguments are repeated to specify a second position, a line is created. Up to 126 world

coordinate pairs may be specified as arguments. See the "DESCRIPTION" section below for more information about this argument.

Description

If one position is specified, then a point is drawn. If two positions are specified, a single vector is drawn. If more than two positions are specified, unconnected lines are drawn. Up to 252 arguments can be specified, a maximum of a 126 unconnected lines are drawn using this routine. If a larger number of points must be specified in a single call, UIS\$LINE_ARRAY should be used.

The points or lines are drawn with the line pattern and width for the attribute block. UIS\$LINE ignores the fill pattern attribute.

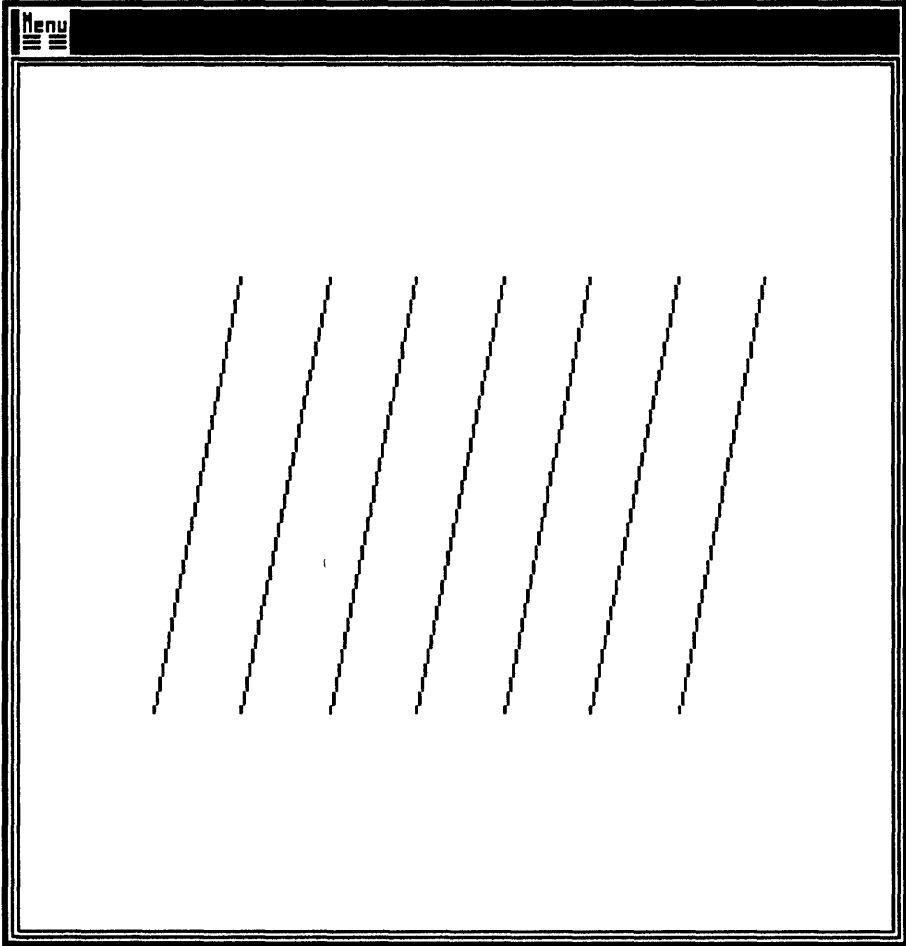
Example

```
.  
. .  
call uis$line(vd_id,0,3.0,5.0,5.0,15.0,5.0,5.0,7.0,15.0,7.0,5.0,  
2      9.0,15.0,  
2      9.0,5.0,11.0,15.0,11.0,5.0,13.0,15.0,  
2      13.0,5.0,15.0,15.0,15.0,5.0,17.0,15.0)
```

A single call to UIS\$LINE draws five unconnected lines.

18-212 UIS Routine Descriptions
UIS\$LINE

Screen Output



UIS\$LINE_ARRAY

Draws an unfilled point, line, or series of unconnected lines depending on the number of positions specified. This routine performs the same functions as UIS\$LINE except that x and y coordinates are stored in arrays.

Format

UIS\$LINE_ARRAY *vd_id, atb, count, x_vector, y_vector*

Returns

UIS\$LINE_ARRAY signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies line style or line width or both.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

18-214 **UIS Routine Descriptions**
UIS\$LINE_ARRAY

Number of points. The **count** argument is the address of longword integer that denotes the number of world coordinate pairs defined in the arguments **x_vector** and **y_vector**.

x_vector, y_vector

VMS Usage: **vector_longword_signed**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Array of *x* and *y* world coordinates. The **x_vector** argument is the address of an array of **f_floating** numbers whose elements are the *x* world coordinate values of points defined in the virtual display. The **y_vector** argument is the address of an array of **f_floating** numbers whose elements are the *y* world coordinate values of points defined in the virtual display.

Description

A maximum of 32,767 points can be plotted in a single call. **UIS\$LINE_ARRAY** is the same as **UIS\$LINE** except that the *x* and *y* coordinates are specified using two arrays, each of length *count* points.

UIS\$MEASURE_TEXT

Measures a text string as if it were output in a virtual display.

Format

UIS\$MEASURE_TEXT *vd_id, atb, text_string, retwidth, retheight, [,ctllist, ctllen] [,posarray]*

Returns

UIS\$MEASURE_TEXT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies text output.

text_string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Text string. The **text_string** argument is the address of a character string descriptor of a text string.

18-216 **UIS Routine Descriptions**
UIS\$MEASURE_TEXT

retwidth, retheight

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

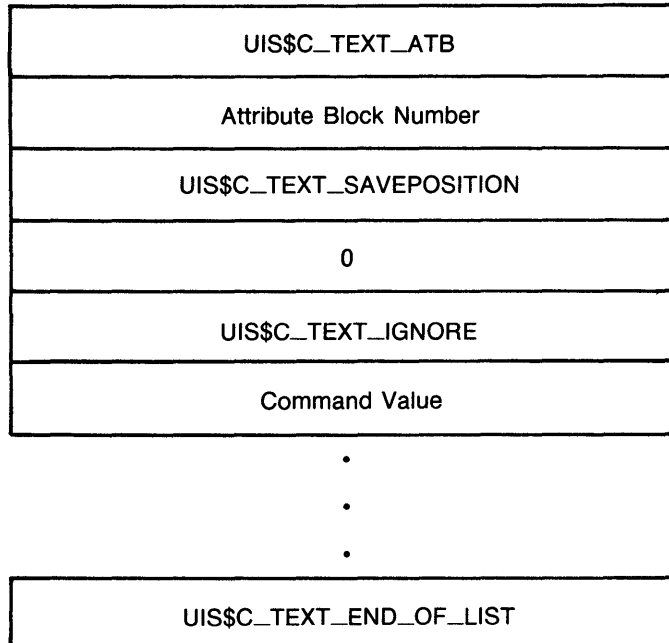
World coordinate width. The **retwidth** and **retheight** arguments are the addresses of *f_floating* point longwords that receive the world coordinate width and height of the text.

ctl

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Text formatting control list. The **ctl** argument is the address of an array of longwords that describe the font, text rendition, format, and positioning of fragments of the text string. See **UIS\$TEXT** for a description of the control list and its commands.

The control list consists of a sequence of data elements, each two longwords in length. The first longword of each element is a tag. The second longword is either a value particular to the type of element specified or zero. Following is a diagram showing the structure of a text control list.



ZK-5426-86

The following table describes valid formatting commands.

Formatting Command	Function
Commands Without Values¹	
UIS\$C_TEXT_NOP	Nil operation
UIS\$C_TEXT_RESTORE_POSITION	Restores the current writing position
UIS\$C_TEXT_SAVE_POSITION	Saves the current writing position
Commands Requiring Values	
UIS\$C_TEXT_ATB	Specifies an attribute block number
UIS\$C_TEXT_HPOS_ABSOLUTE	Specifies a new current x position
UIS\$C_TEXT_HPOS_RELATIVE	Modifies the current x position by a delta
UIS\$C_TEXT_IGNORE	Skips <i>n</i> characters

¹Second longword must be zero.

18-218 **UIS Routine Descriptions**
UIS\$MEASURE_TEXT

Formatting Command	Function
Commands Requiring Values	
UIS\$_TEXT_NEW_LINE	Skips <i>n</i> new lines and positions at the left margin
UIS\$_TEXT_TAB_ABSOLUTE	Writes white space to the new absolute position
UIS\$_TEXT_TAB_RELATIVE	Writes white space to the new relative position
UIS\$_TEXT_VPOS_ABSOLUTE	Writes a new current y position
UIS\$_TEXT_VPOS_RELATIVE	Modifies the current y position by a delta
UIS\$_TEXT_WRITE	Writes <i>n</i> characters
Commands Not Requiring a Second Longword	
UIS\$_TEXT_END_OF_LIST	Terminates the control list

When UIS encounters illegal commands and values within the control list, it skips the invalid item and signals an error.

ctllen

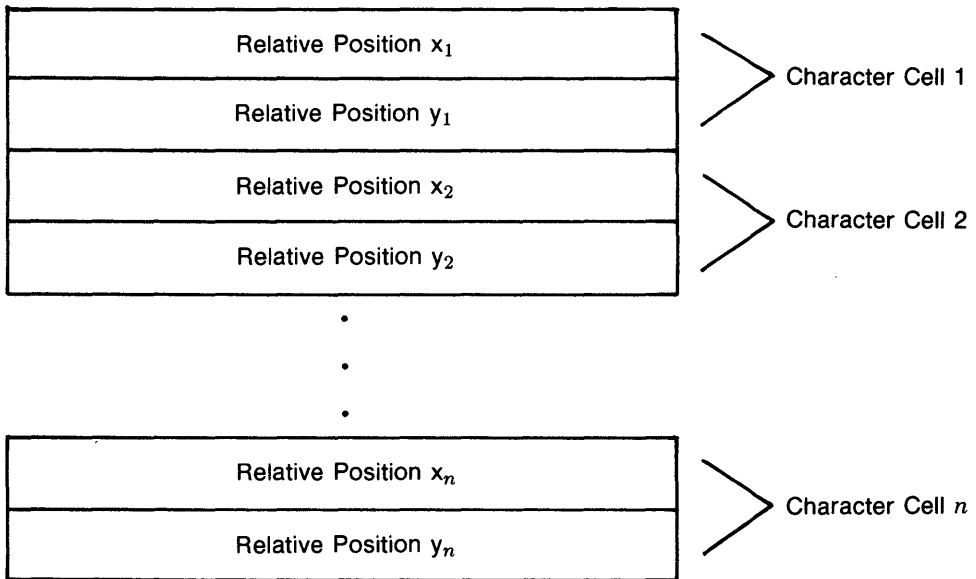
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Length of the text formatting control list. The **ctllen** argument is the address of a longword that specifies the length of the text formatting control list in longwords.

posarray

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Character position array. The **posarray** argument is the address of an array of longwords that receives the character positions in world coordinates, that is, relative offsets at which each character would have been displayed. Following is a diagram showing the format of the character position array.



ZK-5425-86

The width and height of the text string is calculated according to the formatting described in the **atb** and **ctlist** arguments.

Description

UIS\$MEASURE_TEXT is used in justification and text positioning applications. The routine returns the height and width of the text string in world coordinates.

18-220 UIS Routine Descriptions
UIS\$MEASURE_TEXT

Screen Output

```

$ run measure
string width in world coordinates = 16.95
string height in world coordinates = 4.92
The contents of the character position array are
x coordinate = 0.00 y coordinate 0.00
x coordinate = 0.81 y coordinate 0.00
x coordinate = 1.61 y coordinate 0.00
x coordinate = 2.42 y coordinate 0.00
x coordinate = 3.23 y coordinate 0.00
x coordinate = 4.04 y coordinate 0.00

```

Positions of the first six characters including the space relative to the x axis

FORTRAN PAUSE

\$

Menu

far away and long ago

UIS\$MOVE_AREA

Shifts a portion of a virtual display to another position in the display window.

Format

UIS\$MOVE_AREA *vd_id, x1, y1, x2, y2, new_x, new_y*

Returns

UIS\$MOVE_AREA signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

x1, y1, x2, y2

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the source rectangle. The *x1* and *y1* arguments are the addresses of *f_floating* point numbers that define the lower-left corner of the source rectangle. The *x2* and *y2* are the addresses of *f_floating* point numbers that define the upper-right corner of the source rectangle.

new_x, new_y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinate pair. The *new_x* and *new_y* arguments are the addresses of *f_floating* point numbers that define the lower-left corner of

18-222 **UIS Routine Descriptions**
UIS\$MOVE_AREA

the destination rectangle. The proportions of the coordinate space of the destination rectangle are the same as those of the source rectangle.

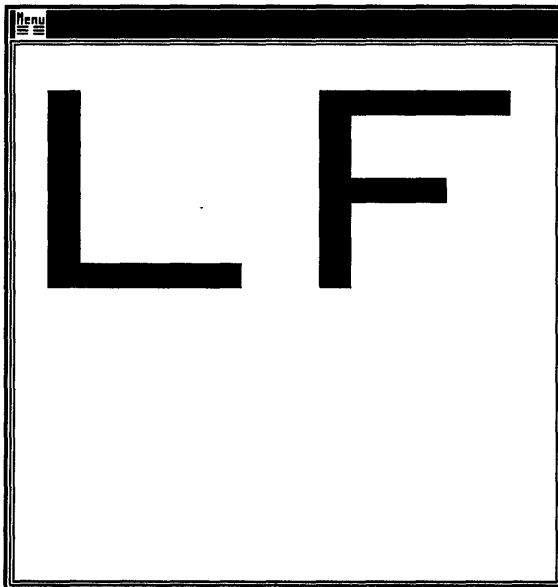
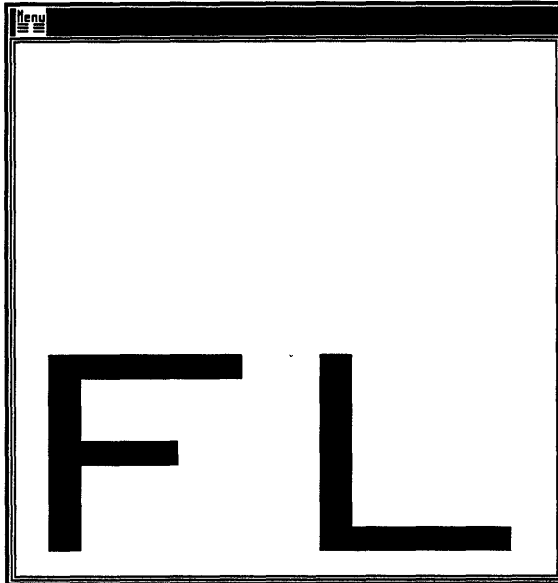
Description

Note that display objects that are only partially contained within the specified source rectangle, though partially moved within existing display windows will be completely moved within the display list.

The nonoccluding portion of the source rectangle (if any) is erased after the operation.

NOTE: To avoid distortion within the destination rectangle, the aspect ratios of the source rectangle and the display viewport must be equal.

Screen Output



UIS\$MOVE_VIEWPORT

Moves the display viewport on the workstation screen.

Format

UIS\$MOVE_VIEWPORT *wd_id, attributes*

Returns

UIS\$MOVE_VIEWPORT signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

attributes

VMS Usage: **item_list_pair**
type: **longword**
access: **read only**
mechanism: **by reference**

Display viewport attribute list. The **attributes** argument is the address of data structure that contains longword pairs, or *doublets*. The first longword stores an attribute ID code and the second longword holds the attribute value (which can be real or integer).

The following figure describes the structure of the window attributes list.

UIS Routine Descriptions 18-225
UIS\$MOVE_VIEWPORT

Attribute ID code (WDPL\$C__xxx)
Longword value for attribute identified in previous longword
2nd attribute ID code
2nd attribute value
• • •
End of list = 0 (WDPL\$C__END__OF__LIST)

ZK-4581-85

Only positional attributes are significant.

UIS\$MOVE_WINDOW

Redefines the world coordinates of the specified display window.

Format

UIS\$MOVE_WINDOW *vd_id, wd_id, x₁, y₁, x₂, y₂*

Returns

UIS\$MOVE_WINDOW signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

x₁, y₁, x₂, y₂

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the new display window. The **x₁** and **y₁** arguments are the addresses of **f_floating** point numbers that define that lower-left corner of the display window. The **x₂** and **y₂** arguments are the addresses

of `f_`floating point numbers that define the upper-right corner of the new display window.

Description

UIS\$MOVE_WINDOW redefines the world coordinates of the specified display window. As a result, what is displayed in the associated display viewport may change. You can pan around a virtual display or scroll through a virtual display. If the display window rectangle changes dimensions or aspect ratio, then scaling is performed to map the new window size to the existing display viewport size.

UIS\$NEW_TEXT_LINE

Moves the current text position along the actual path of text drawing to the starting margin, and then in the direction of the minor text path. Depending on the minor text path, the width or height of the character cell is used for spacing between characters and lines.

Format

UIS\$NEW_TEXT_LINE *vd_id, atb*

Returns

UIS\$NEW_TEXT_LINE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies text output.

Description

Font, text path, character spacing, and text slope attributes influence the behavior.

UIS\$PLOT

Draws a filled or unfilled point, line, or polygon depending on the number of positions specified.

Format

UIS\$PLOT *vd_id, atb, x₁, y₁ [,x₂,y₂ [,...x_n,y_n]]*

Returns

UIS\$PLOT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a longword integer that identifies an attribute block that modifies line style and line width or both.

x, y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinate pair. The *x* and *y* arguments are the addresses of *f_floating* point numbers that define a point in the virtual display. If the arguments are repeated to specify a second position, a line is created. Up to 126 world

18-230 UIS Routine Descriptions

UIS\$PLOT

coordinate pairs may be specified as arguments. See the Description section below for more information about this argument.

Description

If one position is specified, then a point is drawn. If two positions are specified, a single vector is drawn. If more than two positions are specified, a connected polygon is drawn. Up to 252 arguments can be specified, giving a maximum of a 126-point polygon using this routine. If a larger number of points must be specified in a single call, UIS\$PLOT_ARRAY should be used.

The points or lines are drawn with the line pattern and width for the attribute block, and if the fill pattern attribute is enabled for the attribute block, the enclosed area is filled with the current fill pattern.

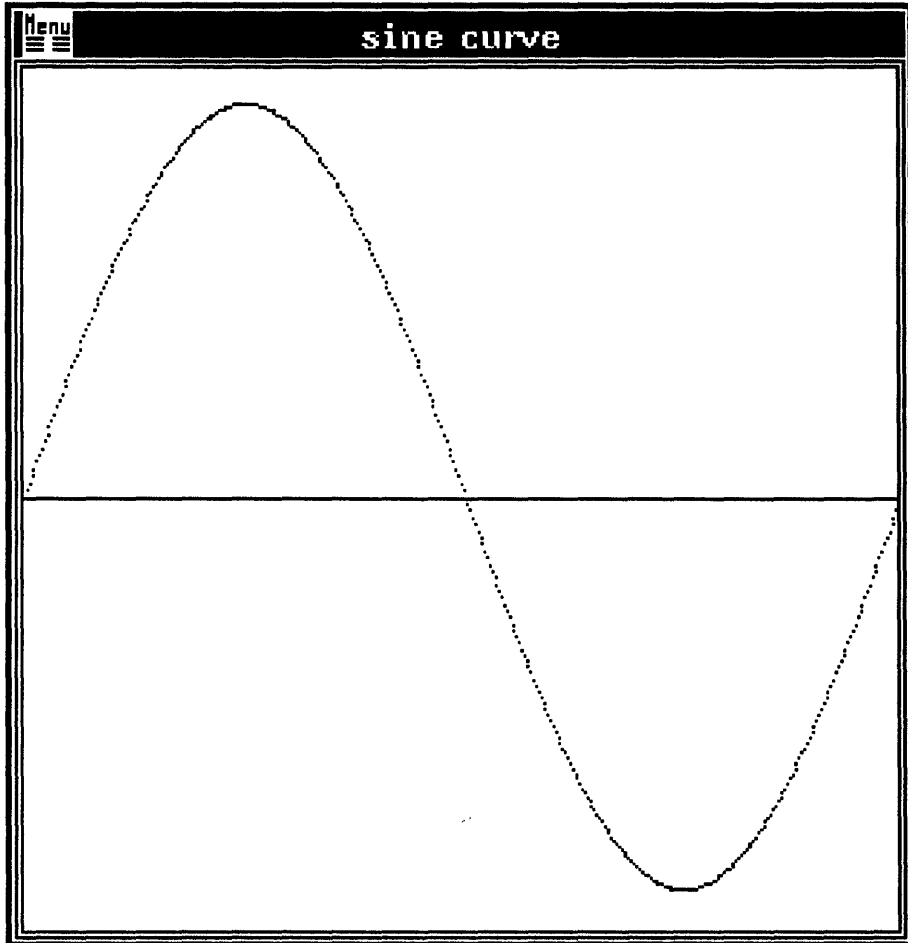
NOTE: VAX PASCAL application programs should use UIS\$PLOT_ARRAY to create lines and polygons.

Example

```
.  
. .  
. .  
REAL*4 I  
VD_ID=UIS$CREATE_DISPLAY(0.0,-1.1,360.0,1.1,10.0,10.0)  
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','SINE CURVE')  
CALL UIS$PLOT(VD_ID,0,0.0,0.0,360.0,0.0)  
  
DO I=1,360  
CALL UIS$PLOT(VD_ID,0,I,SIND(I))  
ENDDO  
. .  
. .
```

The preceding example draws a sine curve.

Screen Output



UIS\$PLOT_ARRAY

Draws an unfilled or filled point, line or polygon depending on the number of positions specified. This routine performs the same functions as `UIS$PLOT`.

Format

UIS\$PLOT_ARRAY *vd_id, atb, count, x_vector, y_vector*

Returns

`UIS$PLOT_ARRAY` signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The `vd_id` argument is the address of a longword that uniquely identifies a virtual display. See `UIS$CREATE_DISPLAY` for more information about the `vd_id` argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The `atb` argument is the address of a longword integer that identifies an attribute block that modifies line style or line width or both.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

UIS\$PLOT_ARRAY

Number of points. The **count** argument is the address of longword integer that denotes the number of world coordinate pairs defined in the arguments **x_vector** and **y_vector**.

x_vector, y_vector

VMS Usage: **vector_longword_signed**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Array of *x* and *y* world coordinates. The **x_vector** argument is the address of an array of **f_floating** numbers whose elements are the *x* world coordinate values of points defined in the virtual display. The **y_vector** argument is the address of an array of **f_floating** numbers whose elements are the *y* world coordinate values of points defined in the virtual display.

Description

A maximum of 65,535 points can be plotted in a single call. **UIS\$PLOT_ARRAY** is the same as **UIS\$PLOT** except that the *x* and *y* coordinates are specified using two arrays, each of length *count* points.

UIS\$POP_VIEWPORT

Pops the viewport associated with the display window to the forefront of the screen, over any other viewports that currently occlude it.

Format

UIS\$POP_VIEWPORT *wd_id*

Returns

UIS\$POP_VIEWPORT signals all errors; no condition values are returned.

Argument

wd_id

VMS Usage: **identifier**

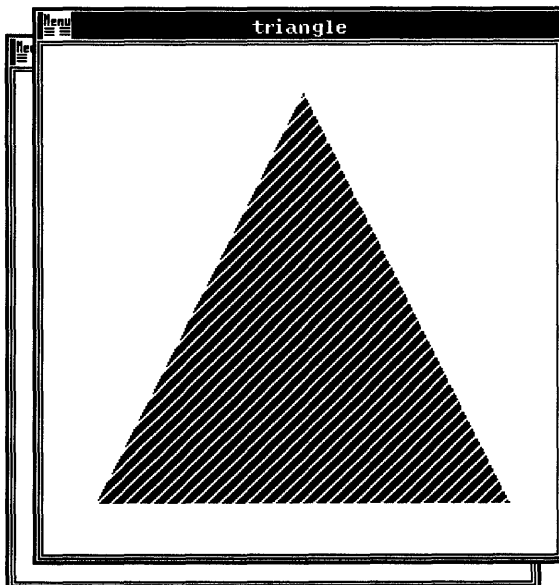
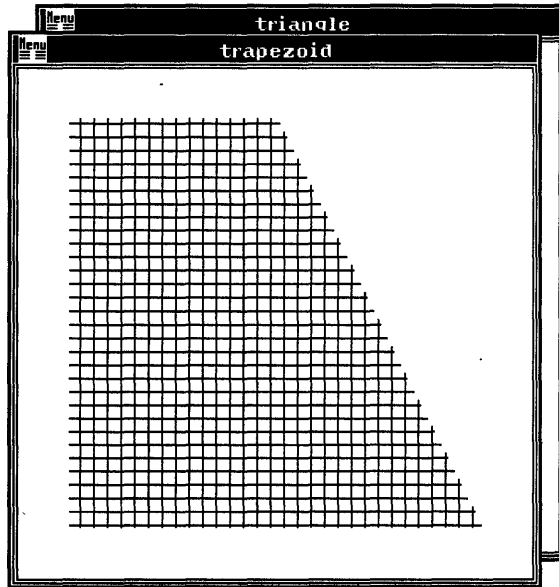
type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

Screen Output



UIS\$PRESENT

Verifies that UIS software is installed on the system.

Format

status=**UIS\$PRESENT** [*major_version*][*minor_version*]

Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned in the variable *status* or R0 (VAX MACRO). A value of 1 TRUE indicates that UIS is installed on the system. Otherwise, the error status SHR\$_PROD_NOTINS is returned if UIS\$PRESENT is executed on a VAX/VMS system running the stub UIS shareable image. The stub shareable image is currently installed on non-VAXstation systems.

Arguments

major_version

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Major version number. The **major_version** argument is the address of a word that receives the major version number. For UIS Version 3.0, the major version number 3 is returned.

minor_version

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Minor version number. The **minor_version** argument is the address of a word that receives the minor version number. For UIS Version 3.0, the minor version number 0 is returned.

UIS\$PRIVATE

Associates application-specific data with the most recently output graphic information (graphics or text) or with the specified graphic object.

Format

UIS\$PRIVATE { *obj_id* } , *facnum*, *buffer*
 { *vd_id* }

Returns

UIS\$PRIVATE signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies an object.

vd_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See **UIS\$CREATE_DISPLAY** for more information about the **vd_id** argument.

facnum

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Facility number. The **facnum** argument is the address of a longword that identifies the creator of the private data.

18-238 **UIS Routine Descriptions**
UIS\$PRIVATE

Values defined with the high bit set are reserved to DIGITAL.

buffer

VMS Usage: **vector_byte_unsigned**

type: **byte (unsigned)**

access: **read only**

mechanism: **by descriptor**

Location of the private data. The **buffer** argument is a descriptor of an array of bytes. The byte array contains the private data.

Description

If you select a graphic item and store it in a file, the application-specific data will be copied with it. If nothing has been output since the beginning of a segment, the data will be associated with the segment.

Many private data items can be associated with the same graphic object.

UIS\$PUSH_VIEWPORT

Pushes the viewport associated with the display window to the background of the screen, behind any other viewports it occludes.

Format

UIS\$PUSH_VIEWPORT *wd_id*

Returns

UIS\$PUSH_VIEWPORT signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

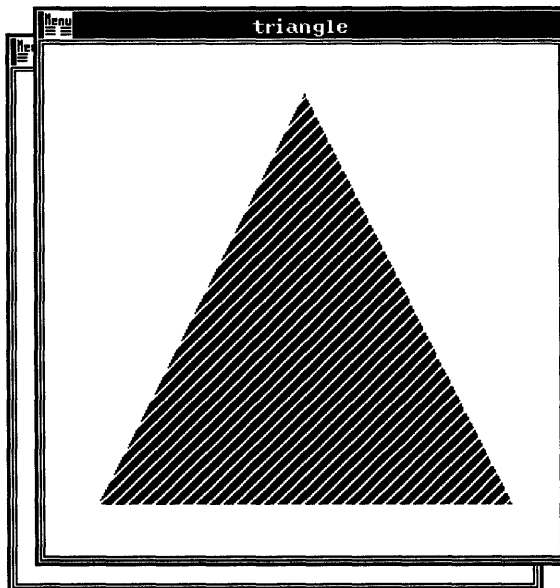
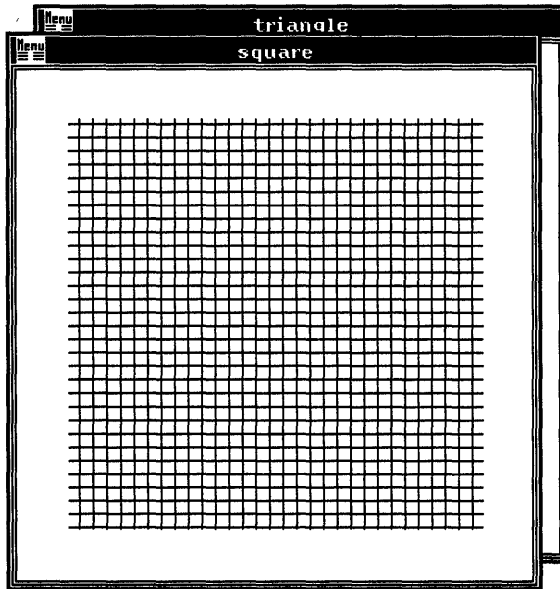
access: **read only**

mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

18-240 UIS Routine Descriptions
UIS\$PUSH_VIEWPORT

Screen Output



UIS\$READ_CHAR

Allows an application to read a single character from the keyboard.

Format

keybuf=UIS\$READ_CHAR *kb_id* [,*flags*]

Returns

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **write only**
 mechanism: **by reference**

Longword integer returned key information in the variable *keybuf* or R0 (VAX MACRO). The *keybuf* variable is the address of a longword buffer that receives the key information. The low two bytes are the key code. The key codes are based on the codes found in the module \$SMGDEF in SYS\$LIBRARY:STARLET.MLB. Bit <31> is set to 1 to indicate that the key is down. For additional information about *keybuf*, see the DESCRIPTION section.

UIS\$READ_CHAR signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Virtual keyboard identifier. The *kb_id* argument is the address of a longword that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the *kb_id* argument.

flags

VMS Usage: **mask_longword**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Flags. The *flags* argument is the address of a longword mask that controls whether UIS\$READ_CHAR executes immediately or until a character is

18-242 UIS Routine Descriptions

UIS\$READ_CHAR

received. If bit <0> is clear, UIS\$READ_CHAR waits until a character is typed. If bit <0> is set and no character is currently waiting, UIS\$READ_CHAR returns a value of 0.

Specify UIS\$M_NOWAIT to set bit <0> in the longword mask.

Description

The following table defines the bits in the high- and lower-order word.

Field	Symbol
1-16	UIS\$W_KEY_CODE
28	UIS\$V_KEY_SHIFT ¹
29	UIS\$V_KEY_CTRL ¹
30	UIS\$V_KEY_LOCK ¹
31	UIS\$V_KEY_DOWN ¹

¹This symbol is returned as SET if the corresponding key on the keyboard was down when the input event occurred.

UIS\$RESIZE_WINDOW

Deletes the old display window and creates a new window. The routine reexecutes the display list of the virtual display, if it exists.

Format

UIS\$RESIZE_WINDOW *vd_id, wd_id [,new_abs_x, new_abs_y] [,new_width, new_height] [,new_wc_x1, new_wc_y1, new_wc_x2, new_wc_y2,]*

Returns

UIS\$RESIZE_WINDOW signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

18-244 UIS Routine Descriptions

UIS\$RESIZE_WINDOW

new_abs_x, new_abs_y

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Absolute device coordinate pair. The **new_abs_x** and **new_abs_y** arguments are the addresses of **f_floating** point numbers that define the location of the newly resized display viewport in centimeters.

new_width, new_height

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

Width and height of the newly resized display viewport. The **width** and **height** arguments are the addresses of **f_floating** point numbers that define the width and height of the newly resized display viewport in centimeters.

new_wc_x1, new_wc_y1, new_wc_x2, new_wc_y2

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

World coordinates of the newly resized display window. The **x₁** and **y₁** arguments are the addresses of **f_floating** point numbers that define the location of the lower-left corner of the resized display window in world coordinates. The **x₂** and **y₂** arguments are the addresses of **f_floating** point numbers that define the location of the upper-right corner of the resized display window in world coordinates.

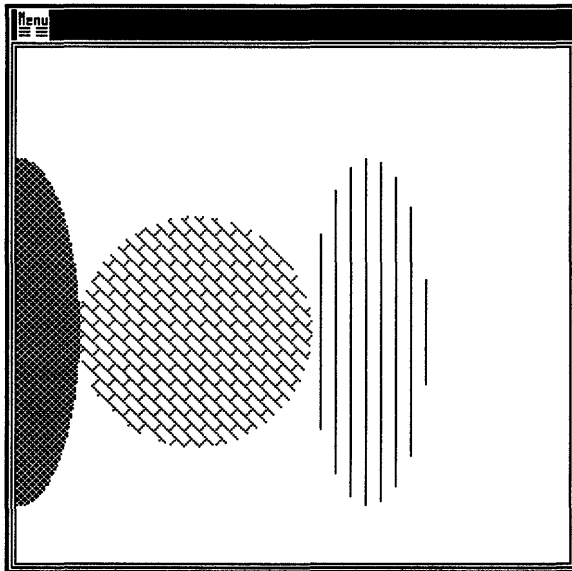
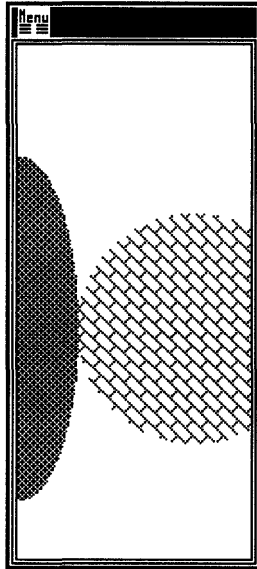
Description

The viewport resize operation of the user interface uses **UIS\$RESIZE_WINDOW** by default.

If **UIS\$RESIZE_WINDOW** is called outside an AST routine, the value of all unspecified parameters defaults to those specified in **UIS\$CREATE_WINDOW**.

If **UIS\$RESIZE_WINDOW** is called within an AST routine, the value of all unspecified parameters defaults to the current values associated with the absolute position, dimensions, and world coordinate range of the stretchy box.

Screen Output



UIS\$RESTORE_CMS_COLORS

Resets the appropriate entries in the hardware color map to the current RGB values in the color map segment.

Format

UIS\$RESTORE_CMS_COLORS *cms_id*

Returns

UIS\$RESTORE_CMS_COLORS signals all errors; no condition values are returned.

Argument

cms_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Color map segment identifier. The **cms_id** argument is the address of a longword that uniquely identifies the color map segment. See UIS\$CREATE_COLOR_MAP_SEG for more information about the **cms_id** argument.

Description

An application running in an unfavorable environment (where other applications are sharing hardware color map entries) can use UIS\$RESTORE_CMS_COLORS to reestablish all its entries when it is the active application. Normally, this call is not required since the UIS window management software transparently handles the multiplexing of the hardware color map. If possible, the update is synchronized to the display's vertical retrace.

UIS\$RGB_TO_HLS

Converts red, green, and blue (RGB) color representation values to hue, lightness, and saturation (HLS) color values.

Format

UIS\$RGB_TO_HLS *R, G, B, reth, retl, rets*

Returns

UIS\$RGB_TO_HLS signals all errors; no condition values are returned.

Arguments

R

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Red value. The **R** argument is the address of a longword that defines the red color value.

G

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Green value. The **G** argument is the address of a longword that defines the green color value.

B

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Blue value. The **B** argument is the address of a longword that defines the blue color value.

18-248 **UIS Routine Descriptions**
UIS\$RGB_TO_HLS

reth

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Hue. The **reth** argument is the address of an **f_floating** point longword that receives the hue color value.

retl

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Lightness. The **retl** argument is the address of an **f_floating** point longword that receives the lightness value.

rets

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Saturation. The **rets** argument is the address of an **f_floating** point longword that receives the color saturation value.

UIS\$RGB_TO_HSV

Converts color representation values of red, green, and blue (RGB) to hue, saturation, and value (HSV).

Format

UIS\$RGB_TO_HSV *R, G, B, reth, rets, retv*

Returns

UIS\$RGB_TO_HSV signals all errors; no condition values are returned.

Arguments

R

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Red value. The **R** argument is the address of an **f_floating** number that defines the red color value.

G

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Green value. The **G** argument is the address of an **f_floating** number that defines the green color value.

B

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Blue value. The **B** argument is the address of an **f_floating** number that defines the blue color value.

18-250 **UIS Routine Descriptions**
UIS\$RGB_TO_HSV

reth

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Hue. The **reth** argument is the address of an **f_floating** longword that receives the hue value.

rets

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Saturation. The **rets** argument is the address of an **f_floating** longword that receives the saturation value.

retv

VMS Usage: **floating_point**
type: **f_floating**
access: **write only**
mechanism: **by reference**

Value. The **retv** argument is the address of an **f_floating** longword that receives the value of the color.

UIS\$SET_ADDOPT_AST

Specifies a user-requested AST routine to be executed whenever the "Additional Options" menu item is selected in the Window Options Menu.

Format

UIS\$SET_ADDOPT_AST *wd_id*, [*astadr* [,*astprm*]]

Returns

UIS\$SET_ADDOPT_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The **astadr** argument is the address of a procedure entry mask of a user-supplied subroutine that is called at AST level whenever the "Additional Options" item in the Window Options Menu is selected.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure such as an array or record to be used by the AST routine.

18-252 **UIS Routine Descriptions**
UIS\$SET_ADDOPT_AST

Calls to `UIS$SET_ADDOPT_AST` in VAX FORTRAN application programs should be coded as follows: `%REF(%LOC(astprm))`.

Description

Additional options are disabled by default.

UIS\$SET_ALIGNED_POSITION

Sets the current position for text output at the upper-left corner of the character cell of the next character. See UIS\$GET_ALIGNED_POSITION for information about returning text alignment data.

Format

UIS\$SET_ALIGNED_POSITION *vd_id, atb, x, y*

Returns

UIS\$SET_ALIGNED_POSITION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** is the address of a longword that identifies an attribute block.

x, y

VMS Usage: **floating_number**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinate pair. The **x** and **y** arguments are the addresses of **f_floating** point numbers that define the current position for text output.

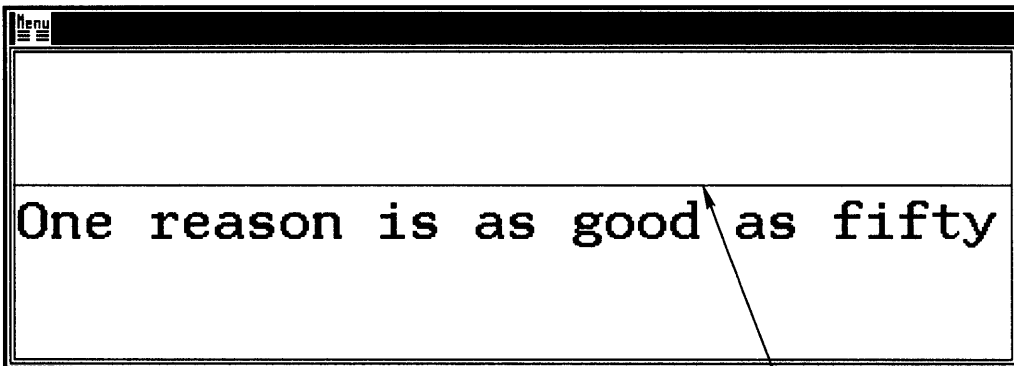
18-254 **UIS Routine Descriptions**
UIS\$SET_ALIGNED_POSITION

Description

UIS\$SET_ALIGNED_POSITION is useful in applications that know the position of the upper left corner, but also do not know enough about the font baseline to determine the proper alignment point. The position is converted into the proper alignment point using the font specified in the given attribute block.

UIS maintains the current text position as a baseline position.

Screen Output



Text alignment
along top of the
character cell

UIS\$SET_ARC_TYPE

Sets the current arc type used in the UIS\$ELLIPSE and UIS\$CIRCLE routines.

Format

UIS\$SET_ARC_TYPE *vd_id, iatb, oatb, arc_type*

Returns

UIS\$SET_ARC_TYPE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a longword integer that identifies an attribute block.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The *oatb* argument is the address of a longword that identifies a newly modified attribute block that controls the appearance of an arc.

18-256 **UIS Routine Descriptions**
UIS\$SET_ARC_TYPE

arc_type

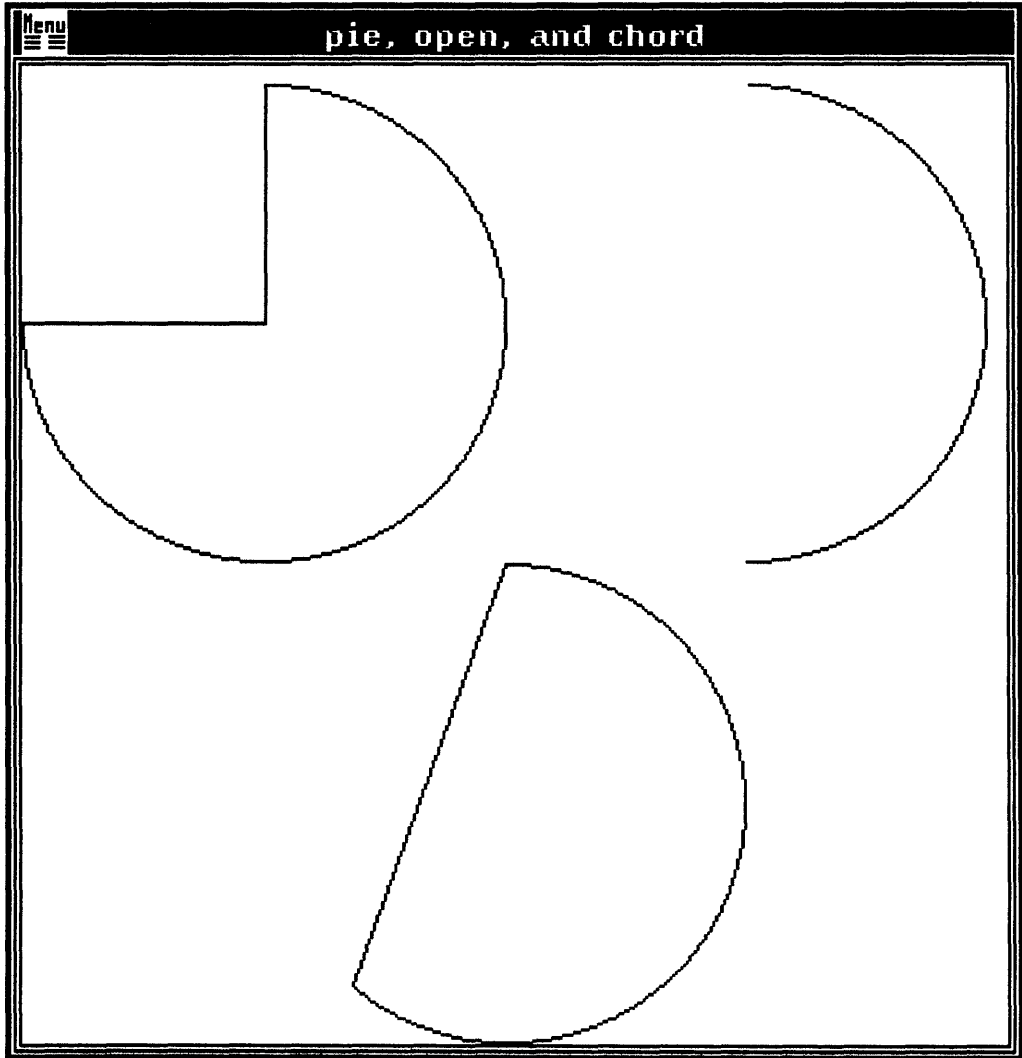
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Arc type code. The **arc_type** argument is the address of a longword value that redefines the attribute setting of the input attribute block. Specify one of the following constants **UIS\$C_ARC_PIE**, **UIS\$C_CHORD**, or **UIS\$C_ARC_OPEN**.

The following table lists symbols for arc types and their functions.

Symbol	Function
UIS\$C_ARC_CHORD	Draws a line connecting the end points of the arc
UIS\$C_ARC_OPEN	Does not draw any lines (default)
UIS\$C_ARC_PIE	Draws radii to the end points of the arc

Screen Output



UIS\$SET_BACKGROUND_INDEX

Sets the background color index for text and graphics output.

Format

UIS\$SET_BACKGROUND_INDEX *vd_id, iatb, oatb, index*

Returns

UIS\$SET_BACKGROUND_INDEX signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword integer that specifies the attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword integer that identifies the newly modified attribute block.

index

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Color map index. The **index** argument is the address of a longword that specifies the color map index. If the index exceeds the maximum index for the associated color map, an error is signaled.

UIS\$SET_BUTTON_AST

Allows an application to find out when a button on the pointing device is depressed or released in a given rectangle within a display viewport.

Format

UIS\$SET_BUTTON_AST *vd_id, wd_id* [*,astadr* [*,astprm*]
,keybuf] [*, x1, y1, x2, y2*]

Returns

UIS\$SET_BUTTON_AST signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The ***astadr*** argument is the address of an entry mask to a procedure that is called at AST level whenever a pointer button is depressed

or released. To cancel the AST-enabling request of `UIS$SET_BUTTON_AST`, specify `0` in the `astadr` argument.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The `astprm` argument is the address of a single argument or data structure, such as a record or an array, to be passed to the AST routine. Calls to `UIS$SET_BUTTON_AST` in FORTRAN application programs should be coded as follows: `%REF(%LOC(astprm))`.

keybuf

VMS Usage: **address**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Key buffer. The `keybuf` argument is the address of a longword buffer that receives button information whenever a pointer button is depressed or released. The low two bytes are the key code. The buttons are located on the left, center, and right of the device and are defined as `UIS$C_POINTER_BUTTON_1`, `UIS$C_POINTER_BUTTON_2`, `UIS$C_POINTER_BUTTON_3`, and `UIS$C_POINTER_BUTTON_4` respectively.

The bit `<31>` is set to `1` if the button has been pressed, and `0` if the button has been released. The buffer is not overwritten with subsequent button transitions until the AST routine completes.

The following table defines the bits in the high- and lower-order word.

Field	Symbol
1-16	<code>UIS\$W_KEY_CODE</code>
28	<code>UIS\$V_KEY_SHIFT</code> ¹
29	<code>UIS\$V_KEY_CTRL</code> ¹
30	<code>UIS\$V_KEY_LOCK</code> ¹
31	<code>UIS\$V_KEY_DOWN</code> ¹

¹This symbol is returned as SET if the corresponding key on the keyboard was down when the input event occurred.

18-262 UIS Routine Descriptions UIS\$SET_BUTTON_AST

x_1, y_1, x_2, y_2

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

World coordinates of a rectangle in the display window. The x_1 and y_1 arguments are the addresses of **f_floating** point numbers that define the lower-left corner of a rectangle in the display window. The x_2 and y_2 arguments are the addresses of **f_floating** point numbers that define the upper-right corner of a rectangle in the display window. If no rectangle is specified, the entire display window is assumed.

Description

This function can be called any number of times for different rectangles within the same display window or many display windows.

To disable UIS\$SET_BUTTON_AST, omit the **astadr**, **astprm**, and **keybuf** arguments.

Pointer Region Priorities

UIS pointer regions are placed on the VAXstation screen in the order in which they are created. Therefore, if you create two overlapping viewports, and then use UIS\$SET_POINTER_PATTERN, UIS\$SET_BUTTON_AST, or UIS\$SET_POINTER_AST to define different pointer patterns for each viewport, the *correctness* of the result will depend on the order in which you both created the viewports and defined the cursor regions. For example, if you create the viewports and define the cursor patterns in the following manner, the viewport 1 cursor pattern will have a higher priority than viewport 2 cursor pattern in the overlapping region.

1. Create viewport 1
2. Create overlapping viewport 2
3. Define viewport 2 cursor pattern
4. Define viewport 1 cursor pattern

UIS\$SET_BUTTON_AST

The preceding example causes the unexpected result that the viewport 1 cursor pattern will take priority over the viewport 2 cursor pattern in the overlapping region. This problem can be corrected by creating the viewports and defining the cursor patterns in the same order. To correct the problem, create the viewports and define cursor patterns in the following order:

1. Create viewport 1
2. Define viewport 1 cursor pattern
3. Create overlapping viewport 2
4. Define viewport 2 cursor pattern

The solution is for either UIS or your application to always pop the viewport before defining the cursor region for it.

UIS\$SET_CHAR_ROTATION

Sets the angle of character rotation, measured counterclockwise relative to the actual path of text drawing.

Format

UIS\$SET_CHAR_ROTATION *vd_id ,iatb ,oatb ,angle*

Returns

UIS\$SET_CHAR_ROTATION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a number that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a modified attribute block.

angle

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Angle of character rotation. The **angle** argument is the address of an **f_floating** point number that defines the angle of character rotation in degrees counterclockwise about the baseline point relative to the actual path of text drawing.

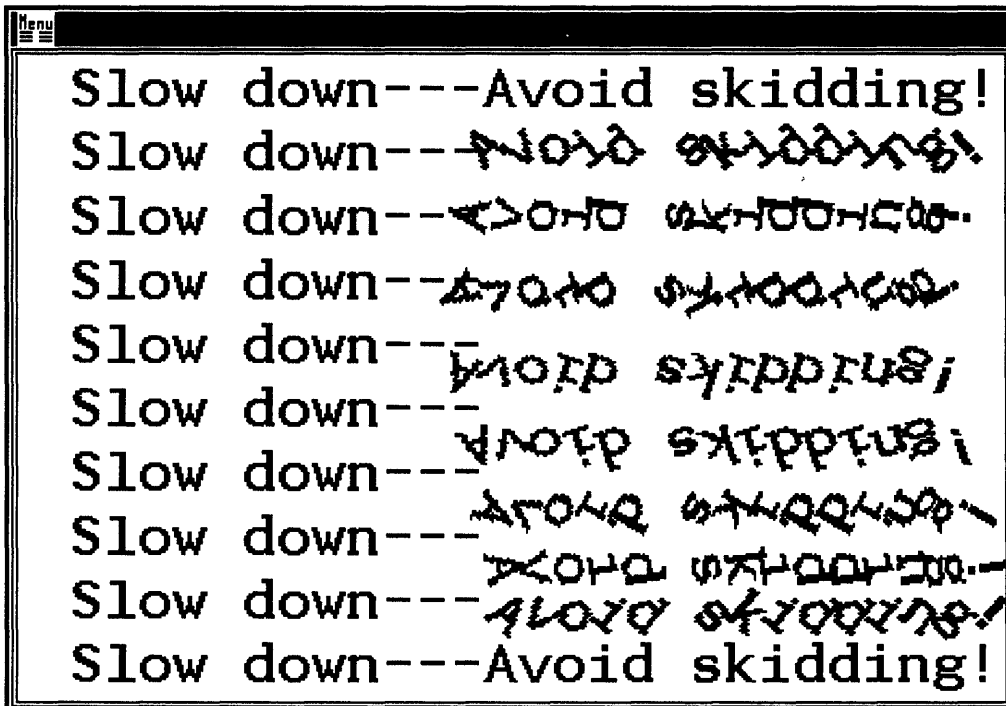
Description

For example, an angle of 0 degrees (the default) means that the character's baseline vector and the actual path of text drawing form an angle of 0 degrees.

Example

```
.  
. .  
CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_5')  
CALLUIS$SET_TEXT_MARGINS(VD_ID,1,1,1.0,20.0,18.0)  
CALL UIS$SET_ALIGNED_POSITION(VD_ID,1,1.0,20.0)  
  
DO I=0,360,40  
CALL UIS$TEXT(VD_ID,1,'Slow down---')  
CALL UIS$SET_CHAR_ROTATION(VD_ID,1,2,FLOAT(I))  
CALL UIS$TEXT(VD_ID,2,'Avoid skidding!')  
CALL UIS$NEW_TEXT_LINE(VD_ID,2)  
ENDDO  
  
. . .
```

Screen Output



UIS\$SET_CHAR_SIZE

Sets the world coordinate size of a specified character set.

Format

UIS\$SET_CHAR_SIZE *vd_id, iatb, oatb* [,char] [,width]
[,height]

Returns

UIS\$SET_CHAR_SIZE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The ***iatb*** argument is the address of a longword that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The ***oatb*** argument is the address of a longword that identifies a modified attribute block.

18-268 **UIS Routine Descriptions**
UIS\$SET_CHAR_SIZE

char

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Single character. The **char** argument is the address of a descriptor of a single character.

If **char** is not specified, the widest character in the font is chosen.

width

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Character width. The **width** argument is the address of an **f_floating** point longword that defines the character width in world coordinates.

See DESCRIPTION section for information about omitting the **width** argument.

height

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Character height. The **height** argument is the address of an **f_floating** point longword that defines the character height in world coordinates.

See DESCRIPTION section for information about omitting the **height** argument.

Description

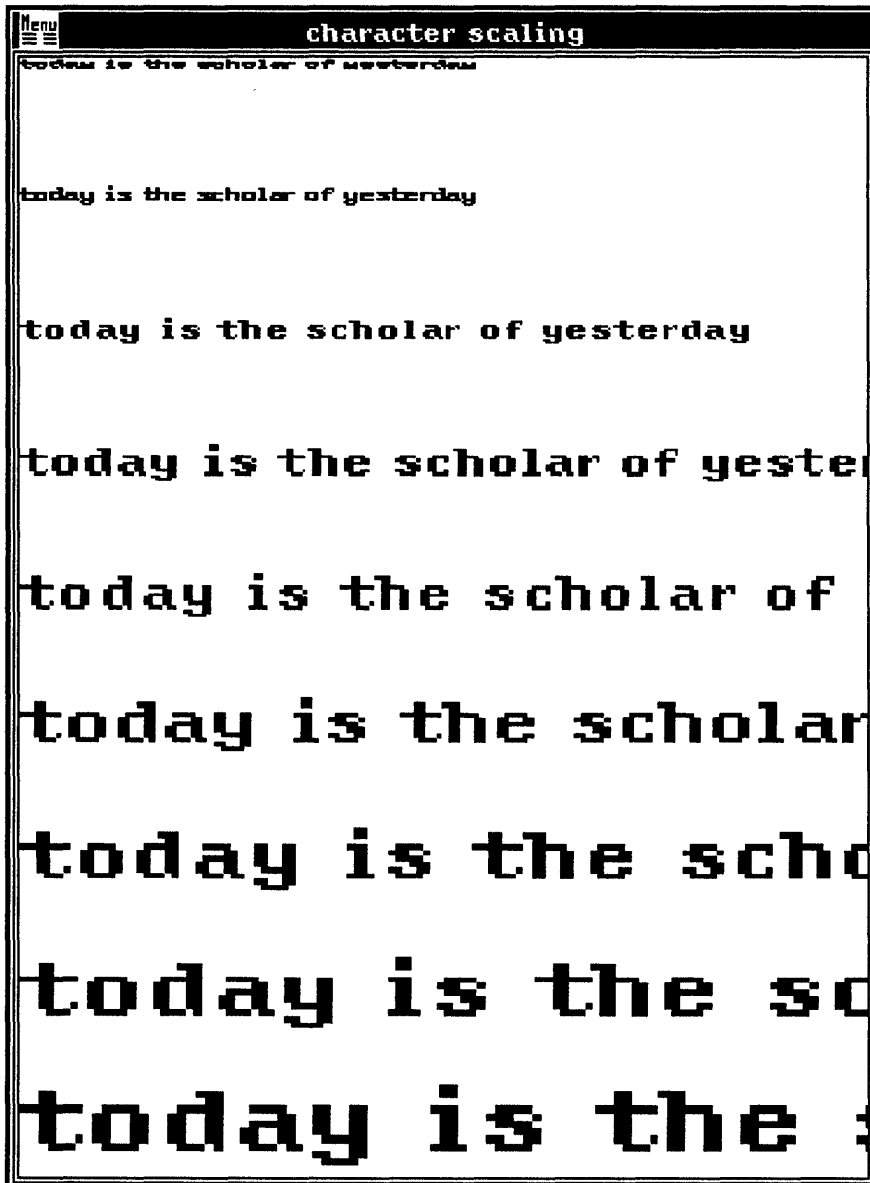
To disable character scaling, omit all of the following arguments: **char**, **width**, and **height**.

To scale characters to their nominal size as specified in the font, do not specify **width** and **height**. Scaling is only visible when you use a window that does not have the same aspect ratio as the virtual display. The particular character you specify in the argument **char** makes no difference in this case.

UIS\$SET_CHAR_SIZE

If you specify either **width** or **height** only, characters are scaled to the size you specify and in the direction you specify. In the unspecified direction, characters are scaled so as to maintain the same ratio of width and height as the unscaled characters.

Screen Output



UIS\$SET_CHAR_SLANT

Sets the character slant angle.

Format

UIS\$SET_CHAR_SLANT *vd_id, iatb, oatb, angle*

Returns

UIS\$SET_CHAR_SLANT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a number that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The *oatb* argument is the address of a number that identifies a modified attribute block.

18-272 **UIS Routine Descriptions**
UIS\$SET_CHAR_SLANT

angle

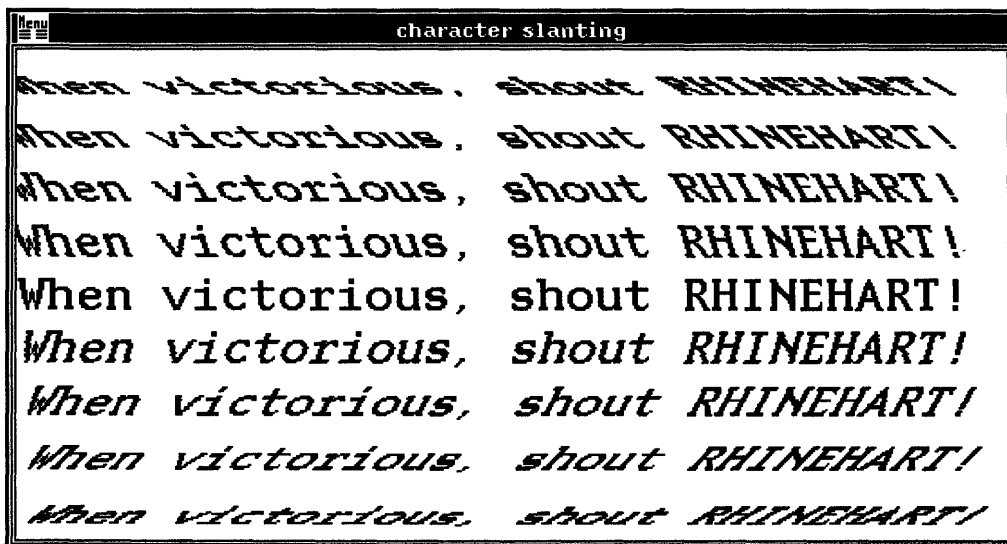
VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Angle of character slant. The **angle** argument is the address of an **f_floating** point number that defines the angle of character slant in degrees.

The character slant angle refers to an angle formed by the character's up vector and baseline vector.

For example, 0 degrees (the default) indicates that the character up vector is perpendicular to the baseline vector, and the character is not slanted. A counterclockwise movement from 0 degrees produces a negative angle of character slant. A clockwise movement from 0 degrees produces a positive angle of character slant.

Screen Output



UIS\$SET_CHAR_SPACING

Sets the attribute that controls the amount of additional spacing between text characters (*x* factor) and between text lines (*y* factor) when the UIS\$NEW_LINE_TEXT routine is used.

Format

UIS\$SET_CHAR_SPACING *vd_id, iatb, oatb, dx, dy*

Returns

UIS\$SET_CHAR_SPACING signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword value that identifies an attribute block to be modified. Either the attribute block 0 or a previously modified attribute block may be specified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

18-274 UIS Routine Descriptions

UIS\$SET_CHAR_SPACING

Output attribute block number. The **oatb** argument is the address of a longword value that identifies the newly modified attribute block that controls the spacing between characters.

dx

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Additional *x* factor spacing. The **dx** argument is the address of an **f_floating** point longword value that defines the *x* spacing factor. If this argument is 0.0, no additional spacing is performed. Negative values are allowed, characters may overlap.

dy

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Additional *y* factor spacing. The **dy** is the argument of an **f_floating** point longword value that defines the *y* spacing factor. If this argument is 0.0, no additional spacing is performed. Negative values are allowed, characters may overlap.

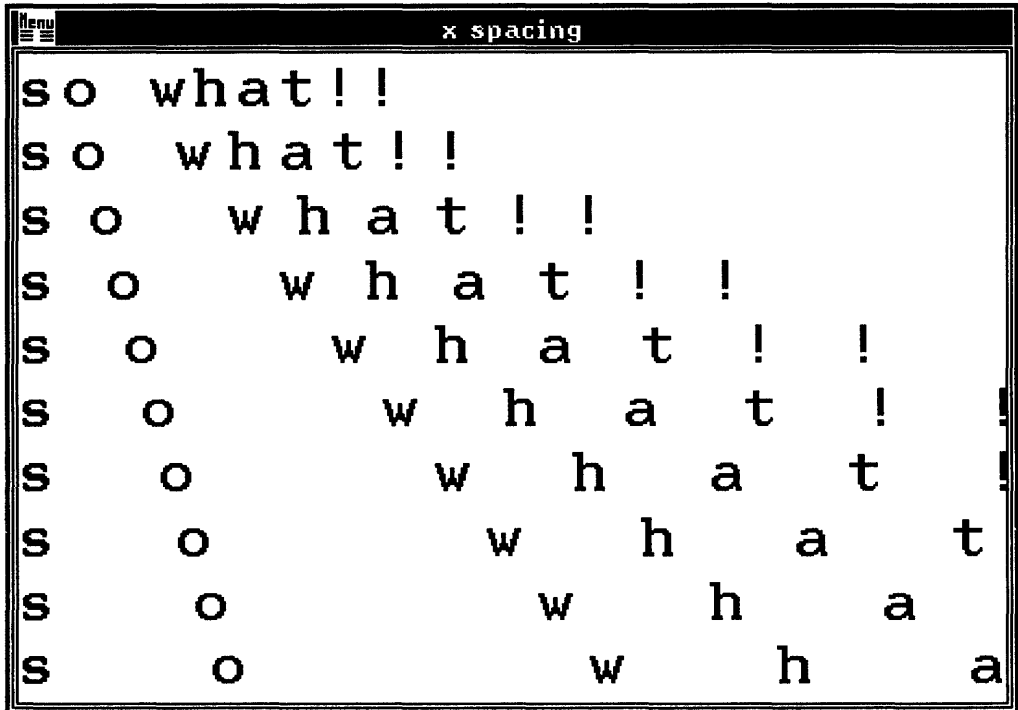
Description

The values of the *x* and *y* factors are multiplied by the width or height of the character, and the resulting value is used as the additional spacing distance.

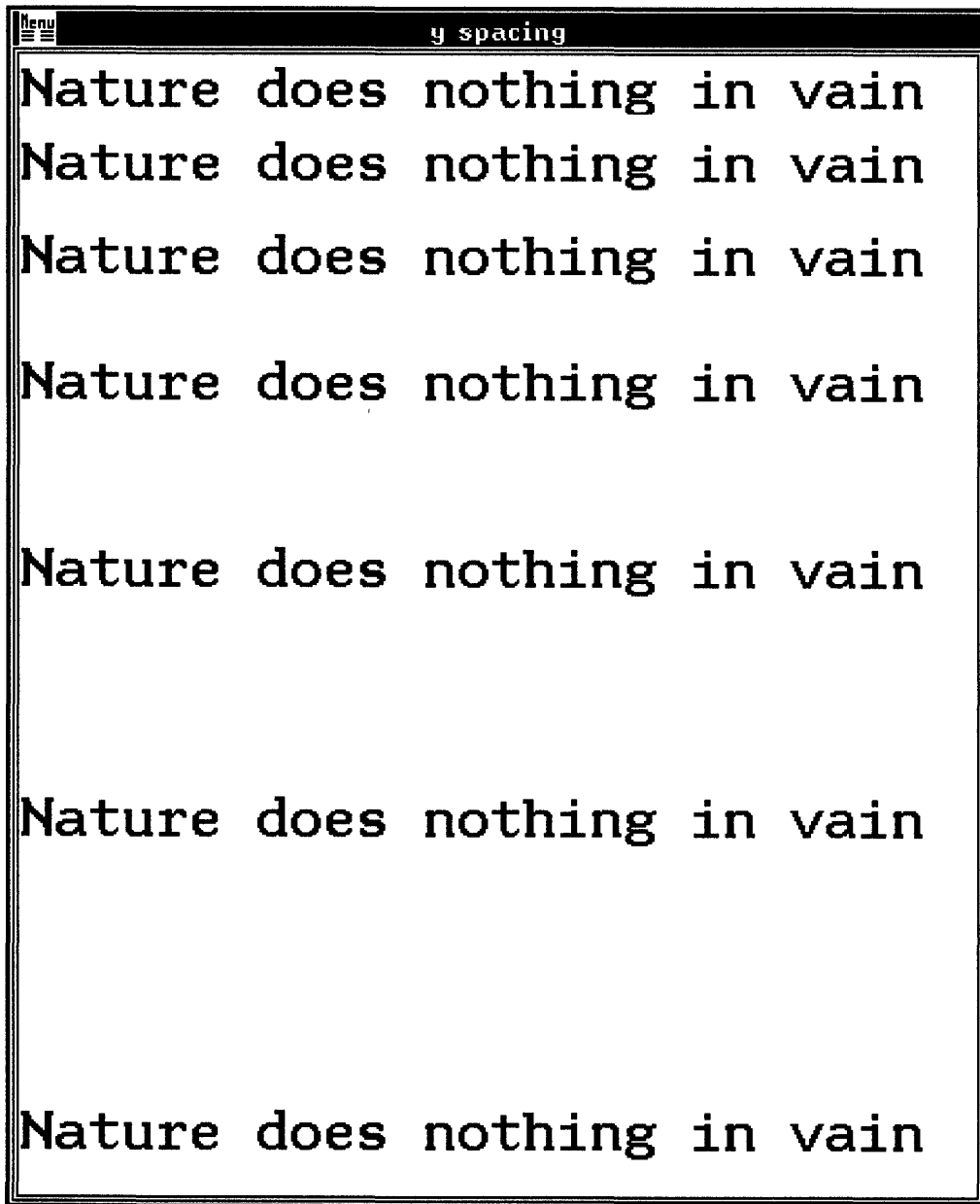
Proportionally spaced characters maintain their appropriate spacing.

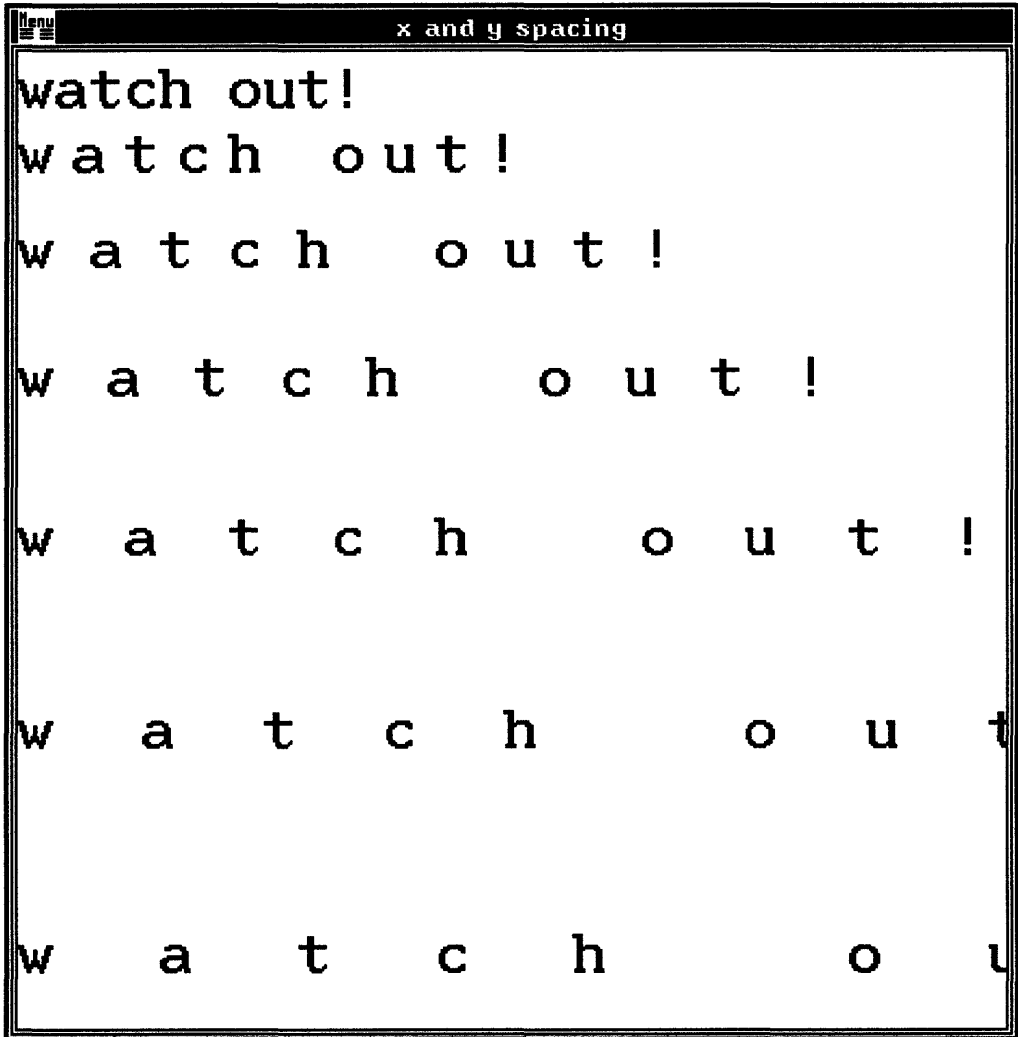
The default is no extra spacing.

Screen Output



```
Menu x spacing
so what!!
so what!!
s o w h a t ! !
s o w h a t ! !
s o w h a t ! !
s o w h a t ! !
s o w h a t ! !
s o w h a t
s o w h a t
s o w h a
s o w h a
```





UIS\$SET_CLIP

Sets a clipping rectangle in the virtual display and enables clipping for this attribute block.

Format

UIS\$SET_CLIP *vd_id, iatb, oatb* [*,x₁, y₁, x₂, y₂*]

Returns

UIS\$SET_CLIP signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword value that identifies an attribute block to be modified. Either the attribute block 0 or a previously modified attribute block can be specified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword value that identifies a newly modified attribute block.

UIS\$SET_CLIP **$x_1, y_1, x_2, y_2$** VMS Usage: **floating_point**type: **f_floating**access: **read only**mechanism: **by reference**

World coordinates of the clipping rectangle. The x_1 and y_1 arguments are the addresses of f_floating point numbers that define the lower left corner of the clipping rectangle in world coordinates. The x_2 and y_2 arguments are the addresses of f_floating point numbers that define the upper right corner of the clipping rectangle in world coordinates. Only graphic objects and portions of graphic objects drawn **within** the clipping rectangle are seen.

If the world coordinates of the clipping rectangle corners are not specified, then clipping is disabled for this attribute block.

Example

```

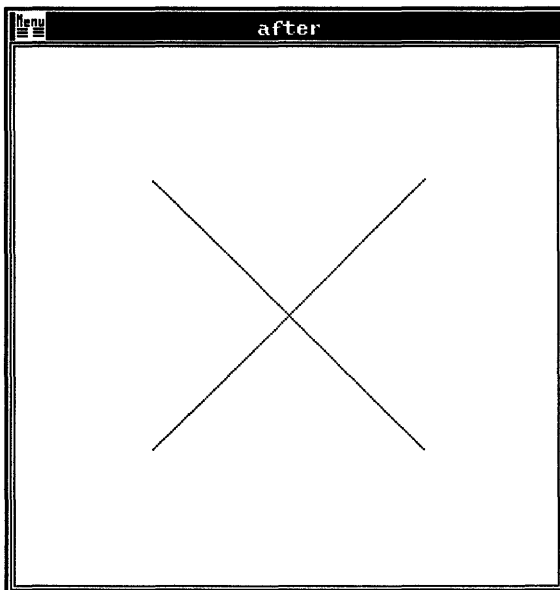
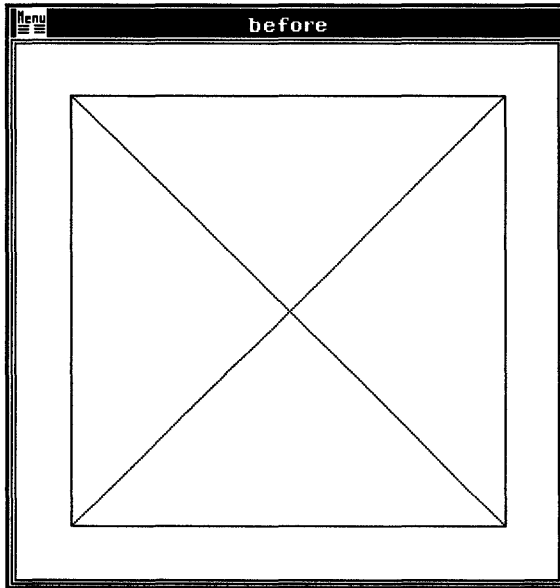
.
.
.
WD_ID1=UIS$CREATE_WINDOW(VD_ID, 'SYS$WORKSTATION', 'AFTER')
CALL UIS$ERASE(VD_ID)
CALL UIS$SET_CLIP(VD_ID, 0, 1, 5.0, 5.0, 15.0, 15.0)

CALL UIS$PLOT(VD_ID, 1, 2.0, 2.0, 18.0, 2.0, 18.0, 18.0, 2.0, 18.0,
2      2.0, 2.0)
CALL UIS$PLOT(VD_ID, 1, 2.0, 2.0, 18.0, 18.0,)
CALL UIS$PLOT(VD_ID, 1, 2.0, 18.0, 18.0, 2.0)
.
.
.

```

18-280 UIS Routine Descriptions
UIS\$SET_CLIP

Screen Output



UIS\$SET_CLOSE_AST

Specifies a user-requested AST routine to be executed when the "Delete" menu item is selected in the Window Options Menu.

Format

UIS\$SET_CLOSE_AST *wd_id* [,*astadr* [,*astprm*]]

Returns

UIS\$SET_CLOSE_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The *astadr* argument is the address of a procedure entry mask of a user-supplied subroutine that is called at AST level whenever the delete item in the Window Options Menu is selected. See the Description section for more information about disabling close AST routines.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The *astprm* is the address of a single argument or data structure, such as an array or record, to be used by the AST routine. Calls to

18-282 UIS Routine Descriptions

UIS\$SET_CLOSE_AST

UIS\$SET_CLOSE_AST in FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

Description

Typically, UIS\$SET_CLOSE_AST is called to override the default window closing behavior. If a CLOSE AST routine are not specified, UIS calls UIS\$CLOSE_WINDOW by default. If this behavior is not sufficient, the application program may call UIS\$SET_CLOSE_AST with its own close routine.

If the application has previously enabled close ASTs, but no longer needs to do special tasks when closing a window, it may specify UIS\$CLOSE_WINDOW as the **astadr** parameter to reenble the default UIS action.

Closing a window may be completely disabled in any of the following ways:

- Specify 0 in the **astadr** argument
- Specify only the **wd_id** argument.
- Omit the **astadr** and **astprm** arguments.

When window closing is disabled, the "Delete" menu item in the Window Options Menu changes from boldface to lightface.

To reenble the default window closing behavior, specify UIS\$C_DEFAULT_CLOSE as the **astadr** argument in a subsequent call to UIS\$SET_CLOSE_AST.

UIS\$SET_COLOR

Sets a single entry in the virtual color map associated with the virtual display. The color map entry is an RGB value for a specific color.

Format

UIS\$SET_COLOR *vd_id, index, R, G, B*

Returns

UIS\$SET_COLOR signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Color map index. The **index** argument is the address of a longword value that identifies an entry in the color map. If the index exceeds the maximum index for the associated color map, an error is signaled.

R

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Red value. The **R** argument is the address of an **f_floating** point number that defines the red value. The red value is in the range of *0.0* to *1.0*, inclusive.

18-284 **UIS Routine Descriptions**
UIS\$SET_COLOR

G

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Green value. The **G** argument is the address of an **f_floating** point number that defines the green value. The green value is in the range of *0.0* to *1.0*, inclusive.

B

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Blue value. The **B** argument is the address of an **f_floating** point number that defines the blue value. The blue value is in the range of *0.0* to *1.0*, inclusive.

Description

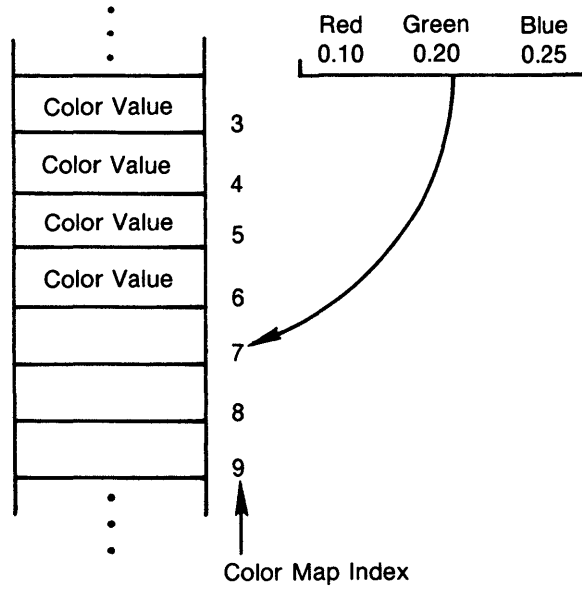
To maximize compatibility between monochrome and color display devices, **UIS\$SET_COLOR** performs an internal transformation of the red, green, and blue values when the actual workstation display is monochromatic.

A single intensity value in the range of *0.0* to *1.0* is derived using the following formula.

$$I = (0.30 * R) + (0.59 * G) + (0.11 * B)$$

On monochrome systems, this derived intensity value is then compared to *0.5*. If the value is greater than or equal to *0.5*, then white pixels are written. Otherwise, black pixels are written.

Illustration



UIS\$SET_COLORS

Sets more than one color entry in the virtual color map.

Format

UIS\$SET_COLORS *vd_id, index, count, r_vector, g_vector,
 b_vector*

Returns

UIS\$SET_COLORS signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Starting color map index. The **index** argument is the address of a longword that defines the starting index in the virtual color map.

If the index exceeds the maximum index for the virtual color map, an error is signaled.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of indices. The **count** argument is the address of a longword that contains the number of indices including the starting index of the color map. If the count exceeds the maximum number of virtual color map entries, an error is signaled.

r_vector

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Red values. The **r_vector** argument is the address of an array of **f_floating** point numbers that define the red values.

g_vector

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Green values. The **g_vector** argument is the address of an array of **f_floating** point numbers that define the green values.

b_vector

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **read only**
mechanism: **by reference**

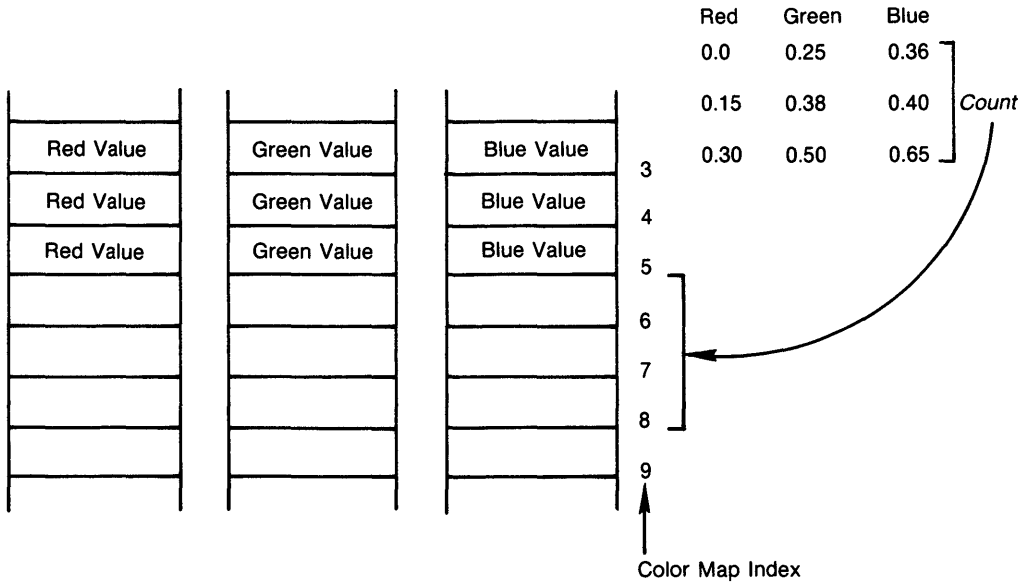
Blue values. The **b_vector** argument is the address of an array of **f_floating** point numbers that define the blue values.

Description

On color and intensity systems, color map updates of greater than approximately 80 entries cause visible screen disturbance, which appears as a black bar across the top inch of the display screen. This anomaly is caused by a hardware restriction that precludes large lookup table updates within the vertical blanking interval of the raster scan.

18-288 UIS Routine Descriptions
UIS\$SET_COLORS

Illustration



UIS\$SET_EXPAND_ICON_AST

Specifies a user-requested AST routine to be executed whenever an icon is to be replaced with its associated display viewport.

Format

UIS\$SET_EXPAND_ICON_AST *wd_id* [,*astadr* [,*astprm*]]

Returns

UIS\$SET_EXPAND_ICON_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The *astadr* argument is the address of an entry mask of a user-written procedure called at AST level whenever the "Expand Icon" menu item in the Window Options Menu is selected.

To cancel the AST-enabling request of UIS\$SET_EXPAND_ICON_AST, specify 0 in the *astadr* argument.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

18-290 UIS Routine Descriptions

UIS\$SET_EXPAND_ICON_AST

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine. Calls to `UIS$SET_EXPAND_ICON_AST` in VAX FORTRAN application programs should be coded as follows: `%REF(%LOC(astprm))`.

Description

The user interface for replacing an icon with a display viewport can be disabled by calling `UIS$SET_EXPAND_ICON_AST` with the **wd_id** argument only.

To reenble the default behavior of `UIS$SET_EXPAND_ICON_AST`, specify the constant `UIS$C_DEFAULT_EXPAND_ICON` in the **astadr** argument.

UIS\$SET_FILL_PATTERN

Sets the current fill pattern used in area fill operations.

Format

UIS\$SET_FILL_PATTERN *vd_id, iatb, oatb [,index]*

Returns

UIS\$SET_FILL_PATTERN signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword integer value that identifies an attribute block to be modified. Either the attribute block 0 or a previously modified attribute block may be specified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

18-292 UIS Routine Descriptions

UIS\$SET_FILL_PATTERN

Output attribute block number. The **oatb** argument is the address of a longword integer value that identifies the newly modified attribute block that controls the fill pattern.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Index of the fill pattern in the current font. The **index** argument is the address of a longword value that identifies a character glyph in the current font. The value specified in the **index** argument modifies the current fill pattern index specified in the input attribute block.

If the **index** argument is not specified, fill patterns are disabled.

Description

The fill pattern is expressed as a character glyph in the font currently associated with the same attribute block. There are usually several font files reserved to store fill patterns (rasters). At present, fill patterns of width greater than 32 bits are not supported.

UIS provides a font file containing a variety of fill patterns. This font file is referenced by UIS\$FILL_PATTERNS. Entries in the UIS\$FILL_PATTERNS font are symbolically referenced by the symbols PATT\$C_xxx.

To get a listing of all fill pattern symbols available to application programs, see Section 6.6 for a list of symbol definition files.

Refer to Appendix D for illustrations showing each UIS fill pattern.

Example

```
.  
. .  
. .  
CALL UIS$SET_FONT(VD_ID,0,1,'UIS$FILL_PATTERNS') ①  
CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_VERT1_7) ②  
CALL UIS$SET_FONT(VD_ID,1,2,'UIS$FILL_PATTERNS')  
CALL UIS$SET_FILL_PATTERN(VD_ID,2,2,PATT$C_HORIZ1_7)  
  
CALL UIS$CIRCLE(VD_ID,1,10.0,10.0,8.0)
```


UIS\$SET_FILL_PATTERN

```
CALL UIS$ERASE(VD_ID)
CALL UIS$PLOT(VD_ID,2,2.0,2.0,18.0,2.0,18.0,18.0,2.0,18.0,
2      2.0,2.0)
```

```
CALL UIS$CIRCLE(VD_ID,1,10.0,10.0,8.0)
CALL UIS$PLOT(VD_ID,2,2.0,2.0,18.0,2.0,18.0,18.0,2.0,18.0,
2      2.0,2.0)
```

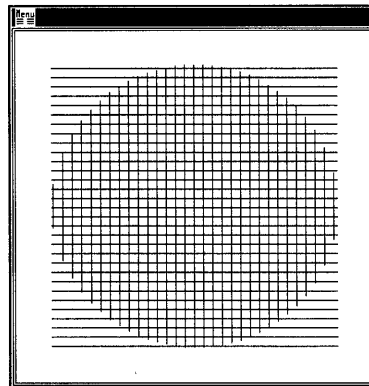
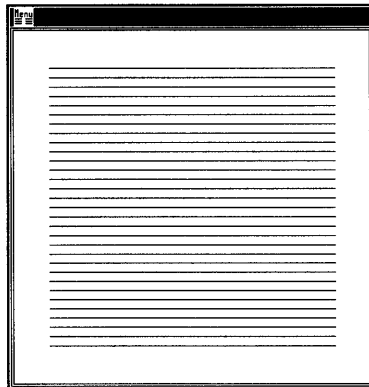
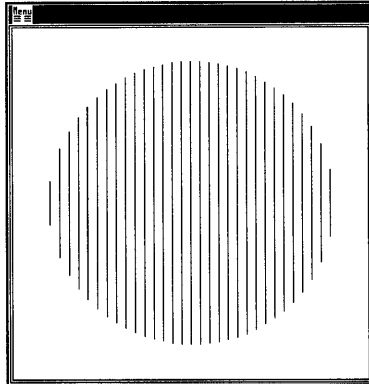
.
.
.

The preceding example fills the circle with a vertical fill pattern and a square with a horizontal fill pattern. Please note that enabling fill patterns for a single graphic object is a two-step process ❶ ❷.

1. Modify the font attribute specifying the fill pattern file in SYS\$FONT.
Use the logical name UIS\$FILL_PATTERN\$.
2. Modify the fill pattern file specifying the fill pattern to be used.

18-294 UIS Routine Descriptions
UIS\$SET_FILL_PATTERN

Screen Output



UIS\$SET_FONT

Specifies the fonts to be used in text drawing (UIS\$TEXT) and area filling (UIS\$PLOT).

Format

UIS\$SET_FONT *vd_id, iatb, oatb, font_id*

Returns

UIS\$SET_FONT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword value that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the argument *vd_id*.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a longword value that specifies the attribute block to be modified. The font attribute in the input attribute block is modified to reflect the new font file specified in the *font_id* argument. Either the attribute block 0 or a previously modified attribute block may be specified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

18-296 **UIS Routine Descriptions**
UIS\$SET_FONT

Output attribute block number. The **oatb** argument is the address of a longword value that specifies the newly modified attribute block.

font_id

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Font file name string. The **font_id** argument is the address of a character string descriptor pointing to a file specification that identifies the desired font. System font files are located in the SYS\$FONT directory. Fonts should be specified using only the file name. You do not need to specify the file type.

Description

See UIS\$SET_FILL_PATTERN.

UIS\$SET_GAIN_KB_AST

Specifies an AST routine to be executed when the specified virtual keyboard is attached to the physical keyboard.

Format

UIS\$SET_GAIN_KB_AST *kb_id* [,*astadr* [,*astprm*]]

Returns

UIS\$SET_GAIN_KB_AST signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The ***kb_id*** argument is the address of a longword value that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the ***kb_id*** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure mask**
access: **read only**
mechanism: **by reference**

AST routine. The ***astadr*** argument is the address of an entry mask to a procedure that is called at AST level whenever a specified virtual keyboard is attached to the physical keyboard.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The ***astprm*** argument is the address of a single argument or data structure, such as an array or record, to be used by the AST routine.

18-298 **UIS Routine Descriptions**
UIS\$SET_GAIN_KB_AST

Calls to UIS\$SET_GAIN_KB_AST in FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

Description

To disable UIS\$SET_GAIN_KB_AST, omit the **astadr** and **astprm** arguments.

UIS\$SET_INSERTION_POSITION

Positions the editing pointer in the display list.

Format

UIS\$SET_INSERTION_POSITION $\left\{ \begin{array}{l} \text{obj_id} \\ \text{seg_id} \\ \text{vd_id} \end{array} \right\} ,[\text{flags}]$

Returns

UIS\$SET_INSERTION_POSITION signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies an object. See the Description section for information about using this argument.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. When **seg_id** is specified as the first argument, the second argument is **not** specified. See the Description section for information about using this argument. See also UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

18-300 **UIS Routine Descriptions**
UIS\$SET_INSERTION_POSITION

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See the Description section for information about using this argument. See also UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Flags. The **flags** argument is the address of a longword mask whose bits define how entries are added to the display list.

The following table lists the flags and their functions.

Flags	Description
UIS\$M_DL_INSERT_AT_BEGIN	Inserts object before first object in the specified structure.
UIS\$M_DL_INSERT_AFTER_OBJECT	Inserts object before specified object in the same segment as the specified object.
UIS\$M_DL_INSERT_BEFORE_OBJECT	Inserts object after specified object in the same segment as the specified object.

See the DESCRIPTION section for more information about how these flags are evaluated.

Description

UIS\$SET_INSERTION_OBJECT examines different options in the **flags** argument depending on the type of object you specify in the first argument. The following table lists the effect of the flags on the different types of objects.

Flags Checked	Effect
Specifying the Virtual Display Identifier	
UIS\$M_DL_INSERT_AT_BEGIN ¹	If this bit is set, the editing pointer is placed at the beginning of the root segment and all new objects are inserted there. If this bit is not set, the editing pointer is placed at the end of the root segment and all new objects are appended to the end of the root segment.
Specifying the Segment Identifier	
All three bits ²	If any bit is set, UIS\$SET_INSERTION_POSITION sets the editing pointer at the place directed by that bit. If no bits are set, the editing pointer is placed at the end of the specified segment and any new objects are appended to the end of the specified segment.
Specifying the Object Identifier	
UIS\$M_DL_INSERT_AFTER_OBJECT ¹ UIS\$M_DL_INSERT_BEFORE_OBJECT	If any bit is set, UIS\$SET_INSERTION_POSITION sets the editing pointer at the place directed by that bit. If no bits are set, the editing pointer is placed at the specified object and any new objects are inserted before the specified object.
¹ If UIS\$M_DL_INSERT_BEFORE_OBJECT or UIS\$M_DL_INSERT_AFTER_OBJECT are set, the routine signals an error. ² If two bits are set, the routine signals an error.	

UIS\$SET_INTENSITIES

Loads one or more intensity values in the virtual color map.

Format

UIS\$SET_INTENSITIES *vd_id, index, count, i_vector*

Returns

UIS\$SET_INTENSITIES signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Starting color map index. The **index** argument is the address of a longword that identifies the starting color map index in the virtual color map.

If an index exceeds the maximum index for the virtual color map, an error is signaled.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of indices. The **count** argument is the address of a longword that defines the number of indices in the virtual color map (including the starting index) whose entries are to be loaded with intensity values.

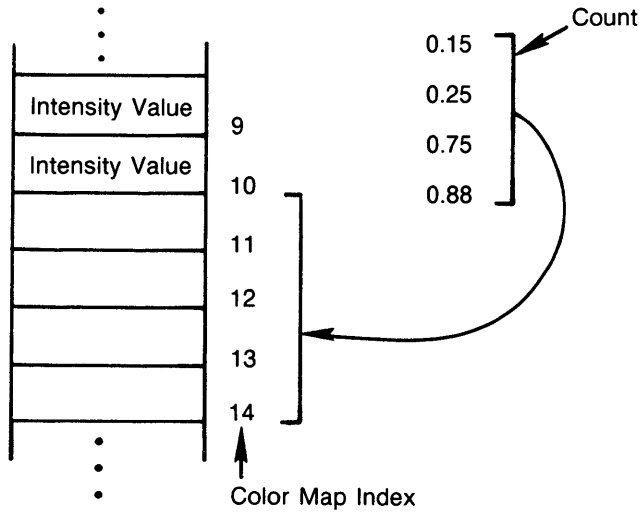
If **count** exceeds the maximum number of virtual color map entries, an error is signaled.

i_vector

VMS Usage: **vector_longword_signed**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Intensity values. The **i_vector** argument is the address of an array of **f_floating** point numbers that define the intensity values of the virtual color map entries.

Illustration



UIS\$SET_INTENSITY

Loads a single entry in the virtual color map with an intensity value.

Format

UIS\$SET_INTENSITY *vd_id, index, I*

Returns

UIS\$SET_INTENSITY signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword value that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

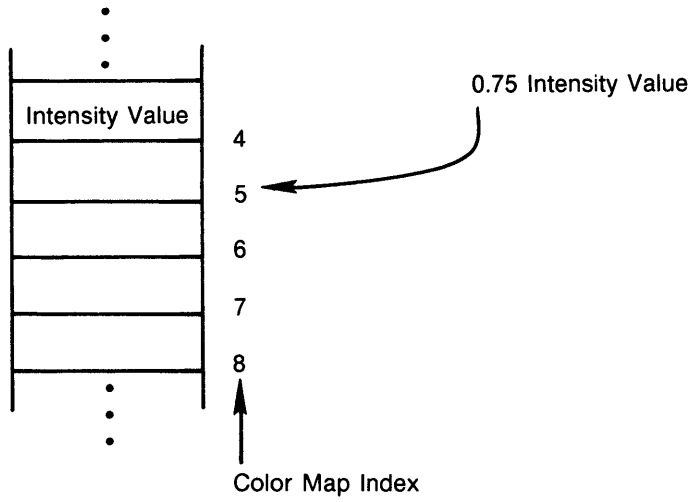
Color map index. The **index** argument is the address of a longword value that identifies an entry in the color map. If the index exceeds the maximum index for the associated color map, an error is signaled.

I

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Intensity value. The **I** argument is the address of an **f_floating** point number that defines the intensity. The intensity value is in the range of *0.0* to *1.0*, inclusive.

Illustration



UIS\$SET_KB_AST

Associates a key strike with the execution of a user-written AST routine.

Format

UIS\$SET_KB_AST *kb_id* [*,astadr* [*,astprm*],*keybuf*]

Returns

UIS\$SET_KB_AST signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The ***kb_id*** argument is the address of a longword value that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the ***kb_id*** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The ***astadr*** argument is the address of the entry mask to a procedure to be called at AST level whenever a key is struck. To cancel a previous AST-enabling request of UIS\$SET_KB_AST, specify 0 as the ***astadr*** argument.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The ***astprm*** argument is the address of a single argument or data structure, such as an array or record to be passed to the AST routine.

UIS\$SET_KB_AST

Calls to UIS\$SET_KB_AST in FORTRAN application programs that use this argument should be coded as follows: %REF(%LOC(astprm)).

keybuf

VMS Usage: **address**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Key buffer. The **keybuf** argument is the address of a longword buffer that receives the key information with the execution of each AST routine. The low two bytes are the key code. The key codes are based on the codes found in the module \$SMGDEF in SYS\$LIBRARY:STARLET.MLB. Bit <31> is set to 1 to indicate that the key is down. The AST routine is called only on the downstroke of the key. The buffer is not overwritten with subsequent keys until the AST routine completes.

The following table defines the bits in the high- and lower-order word.

Field	Symbol
1-16	UIS\$W_KEY_CODE
28	UIS\$V_KEY_SHIFT ¹
29	UIS\$V_KEY_CTRL ¹
30	UIS\$V_KEY_LOCK ¹
31	UIS\$V_KEY_DOWN ¹

¹This symbol is returned as SET if the corresponding key on the keyboard was down when the input event occurred.

Description

The terminal emulators use this routine to get all keyboard input. Other applications that perform asynchronous single character input can also use UIS\$SET_KB_AST.

To disable UIS\$SET_KB_AST, omit the **astadr** and **astprm** arguments.

Keyboard characteristics to be disabled. The **disable_items** argument is the address of a longword mask that identifies the keyboard characteristics to be disabled.

click_volume

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Click volume level. The **click_volume** argument is the address of a longword value that modifies the keyboard click volume for keyboard input to this window. The value is in the range 1 to 8, where the value 1 is the minimum volume level, and the value 8 is the maximum volume level. The default volume level is controlled by the workstation setup menu mechanism.

Description

All keyboard characteristics will be in effect only when the physical keyboard is attached to the specified virtual keyboard. Each virtual keyboard maintains its own keyboard characteristics and the human interface automatically switches the characteristics when the keyboard is associated with another virtual keyboard.

The enable and disable item lists are longword masks containing bits designating the characteristics to be enabled or disabled. The valid bits in the keyboard characteristics enable and disable masks are:

Symbol	Description ¹
UIS\$_M_KB_AUTORPT	Enable/disable keyboard autorepeat
UIS\$_M_KB_KEYCLICK	Enable/disable keyboard keyclick
UIS\$_M_KB_UDF6	Enable/disable up button transitions for F6 to F10 keys
UIS\$_M_KB_UDF11	Enable/disable up button transitions for F11 to F14 keys
UIS\$_M_KB_UDF17	Enable/disable up button transitions for F17 to F20 keys
UIS\$_M_KB_HELPDO	Enable/disable up button transitions for HELP and DO keys
UIS\$_M_KB_UDE1	Enable/disable up button transitions for E1 to E6 keys

¹By default down button transitions are enabled.

18-310 **UIS Routine Descriptions**
UIS\$SET_KB_ATTRIBUTES

Symbol	Description¹
UIS\$M_KB_ARROW	Enable/disable up button transitions for arrow keys
UIS\$M_KB_KEYPAD	Enable/disable up button transitions for numeric keypad keys

¹By default down button transitions are enabled.

Example

```
.  
. .  
enable_items=UIS$M_KB_HELPDO .OR. UIS$M_KB_UDE1 .OR. UIS$M_KB_ARROW  
disable_items=UIS$M_KB_AUTORPT .OR. UIS$M_KB_KEYCLICK  
CALL UIS$SET_KB_ATTRIBUTES(KB_ID, ENABLE_ITEMS,DISABLE_ITEMS)  
. .  
.
```

The preceding example describes how to enable and disable more than one keyboard characteristic at a time.

UIS\$SET_KB_COMPOSE2

Loads a two-stroke compose sequence table for the specified virtual keyboard.

Format

UIS\$SET_KB_COMPOSE2 *kb_id* [, *table*, *tablelen*]

Returns

UIS\$SET_KB_COMPOSE2 signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword value that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the **kb_id** argument.

table

VMS Usage: **vector_longword_unsigned**
type: **longword array**
access: **read only**
mechanism: **by reference**

Compose table. The **table** argument is the address of an array that identifies the compose table. If no table is specified, the system default table is reestablished.

tablelen

VMS Usage: **word_unsigned**
type: **word**
access: **read only**
mechanism: **by reference**

Length of the compose table in bytes. The **tablelen** argument is the address of word that defines the length of the compose table in bytes.

Description

You can use *compose sequences* to create characters that do not exist as standard keys on your keyboard.

Two-stroke sequences can be used on all keyboards except the North American keyboard. Two-stroke sequences do not use the `COMPOSE` key. Although faster to use than the three-stroke sequence, two-stroke sequences are limited to sequences starting with the following nonspacing diacritical marks: grave accent (`), acute accent (´), circumflex accent (^), tilde mark (~), diaeresis mark (¨), and the ring mark. Instead of using the `COMPOSE` key, as in a three-stroke sequence, you use a nonspacing diacritical mark to initiate the two-stroke sequence. You then enter a standard character that, together with that diacritical mark, results in a valid compose sequence.

Please refer to the *MicroVMS Workstation Video Device Driver Manual* for a description of this table and the macros to generate it. An application wishing to modify a table can use these macros to build a new table.

The MicroVMS Workstation contains a copy of the DIGITAL standard two-stroke compose table residing within the driver. This can be changed by performing a call to the SYS\$QIO system service to the QVSS device driver.

NOTE: DIGITAL standard two-stroke compose sequences are not supported on the North American keyboard.

UIS\$SET_KB_COMPOSE3

Loads a three-stroke compose sequence for the specified virtual keyboard.

Format

UIS\$SET_KB_COMPOSE3 *kb_id* [*,table, tablelen*]

Returns

UIS\$SET_KB_COMPOSE3 signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The *kb_id* argument is the address of a longword value that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the *kb_id* argument.

table

VMS Usage: **vector_longword_unsigned**
type: **longword array**
access: **read only**
mechanism: **by reference**

Compose table. The *table* argument is the address of an array that identifies the compose table.

tablelen

VMS Usage: **word_unsigned**
type: **word**
access: **read only**
mechanism: **by reference**

Length of the compose table in bytes. The *tablelen* argument is the address of a word that defines the length of the compose table in bytes.

Description

You can use *compose sequences* to create characters that do not exist as standard keys on your keyboard. There are two types of compose sequences: two-stroke sequences and three-stroke sequences.

Three-stroke sequences can be used on all keyboards. They are performed by first pressing the `COMPOSE` key and then pressing two standard keys.

Please refer to the *MicroVMS Workstation Video Device Driver Manual* for a description of this table and the macros to generate it. An application wishing to modify a table can use these macros to build a new table.

The MicroVMS Workstation contains a copy of the DIGITAL standard three-stroke compose tables residing within the driver. This can be changed by performing a call to the SYS\$QIO system service to the QVSS device driver.

UIS\$SET_KB_KEYTABLE

Loads a keyboard equivalence table for the specified virtual keyboard.

Format

UIS\$SET_KB_KEYTABLE *kb_id* [,*table*, *tablelen*]

Returns

UIS\$SET_KB_KEYTABLE signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the **kb_id** argument.

table

VMS Usage: **vector_longword_unsigned**
type: **longword array**
access: **read only**
mechanism: **by reference**

Keyboard table. The **table** argument is the address of an array that contains the keyboard table. If no table is specified, the system default table is reestablished.

tablelen

VMS Usage: **word_unsigned**
type: **word**
access: **read only**
mechanism: **by reference**

Length of the keyboard table. The **tablelen** argument is the address of a word that specifies the length of the keyboard table in bytes.

Description

UIS\$SET_KB_KEYTABLE lets you change the ASCII character returned by a key on the keyboard.

Keyboard Table Description and Macros

Please refer to the *MicroVMS Workstation Video Device Driver Manual* for a description of the table and the macro to build it. An application wishing to modify a table can use these macros to build a new table.

Keyboard Table Modification Using the Programming Interface

The MicroVMS Workstation contains a copy of the North American table established as the default keyboard table. You can modify the default keyboard table at the driver (QVSS) level by calling the SYS\$QIO system service.

Keyboard Table Modification Through the User Interface

If you want to create a keyboard table that any user can load using the Workstation Setup menus, see the command file DVORAK.COM in the directory SYS\$EXAMPLES. It provides an example of how to create, compile, and install the DVORAK simplified keyboard. The user interface can be used to modify the default key table.

UIS\$SET_LINE_STYLE

Sets the line style bit vector.

Format

UIS\$SET_LINE_STYLE *vd_id, iatb, oatb, style*

Returns

UIS\$SET_LINE_STYLE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword integer that specifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword integer that specifies the newly modified attribute block that controls the line style.

18-318 **UIS Routine Descriptions**
UIS\$SET_LINE_STYLE

style

VMS Usage: **mask_longword**

type: **longword**

access: **read only**

mechanism: **by reference**

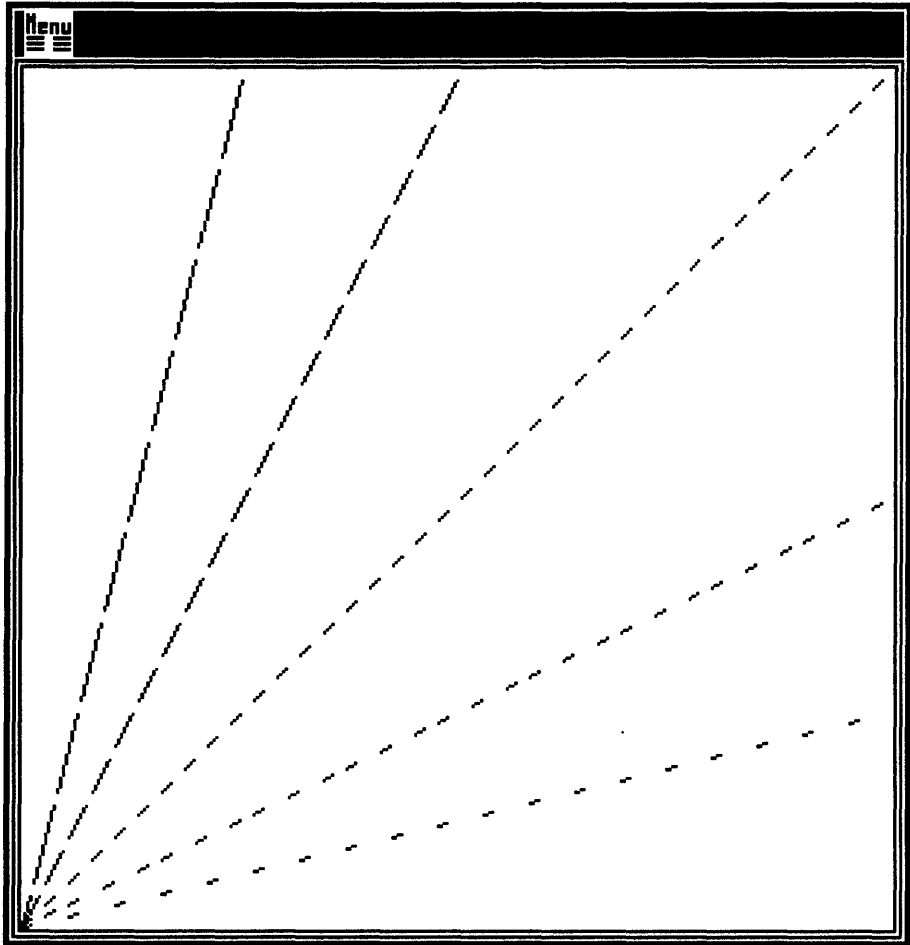
Line style bit vector. The **style** argument is the address of a longword bit vector that specifies whether to use foreground or background when drawing each pixel. It is repeated as many times as necessary to draw all the pixels in the line.

Example

```
.  
. .  
. .  
CALL UIS$SET_LINE_STYLE(VD_ID,0,1,'FFFFFFFO'x)  
CALL UIS$PLOT(VD_ID,1,0.0,0.0,5.0,20.0)  
  
CALL UIS$SET_LINE_STYLE(VD_ID,0,2,'FFFOFFFO'x)  
CALL UIS$PLOT(VD_ID,2,0.0,0.0,10.0,20.0)  
. .  
. .  
. .
```

The preceding example produces the first two dashed lines shown in the next section.

Screen Output



UIS\$SET_LINE_WIDTH

Sets the width of lines drawn on the screen.

Format

UIS\$SET_LINE_WIDTH *vd_id, iatb, oatb, width [,mode]*

Returns

UIS\$SET_LINE_WIDTH signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a longword that specifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The *oatb* argument is the address of a longword that specifies an attribute block that controls line width.

width

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Width of the line. The **width** argument is the address of an **f_floating** point number that defines the line width. See the DESCRIPTION section for more information about specifying the line width with **UIS\$C_WIDTH_WORLD**. The default value is 1.

mode

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mode. The **mode** argument is the address of a longword that indicates whether the line width should be interpreted as an absolute number of pixels or as an x world coordinate width. Specify the mode using one of the following constants:

- **UIS\$C_WIDTH_PIXELS**
- **UIS\$C_WIDTH_WORLD**

If **mode** is not specified, line width is interpreted as an absolute number of pixels (**UIS\$C_WIDTH_PIXELS**).

See DESCRIPTION for more information about the constant **UIS\$C_WIDTH_WORLD**.

Description

The line width is specified as a floating point number that is multiplied by the normal line width to produce line width actually drawn.

If you specify *0.0* in the **width** argument when the **mode** argument is **UIS\$C_WIDTH_WORLD**, the minimum line width is generated.

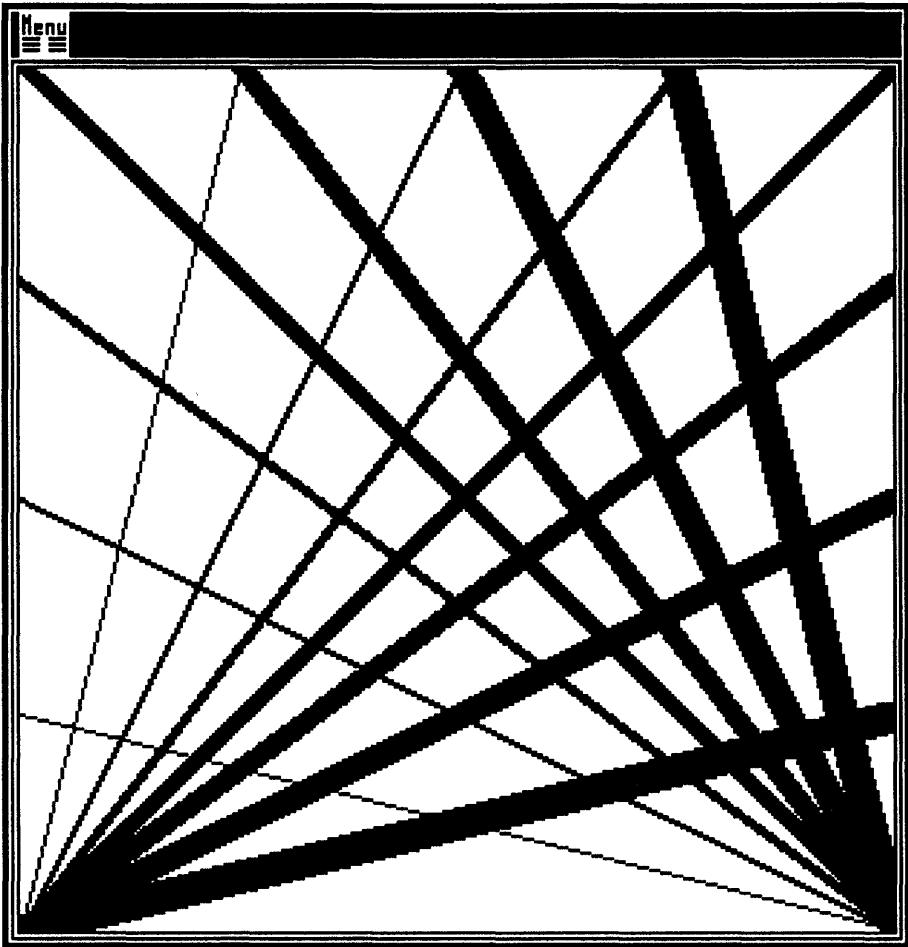
18-322 **UIS Routine Descriptions**
UIS\$SET_LINE_WIDTH

Example

```
.  
. .  
. .  
CALL UIS$SET_LINE_WIDTH(VD_ID,0,1,2.0,WDPL$C_WIDTH_WORLD)  
CALL UIS$PLOT(VD_ID,1,0.0,0.0,10.0,20.0)  
  
CALL UIS$SET_LINE_WIDTH(VD_ID,0,2,4.0,WDPL$C_WIDTH_WORLD)  
CALL UIS$PLOT(VD_ID,2,0.0,0.0,15.0,20.0)  
. .  
. .  
. .
```

The preceding example describes how to specify line width as x world coordinate width.

Screen Output



UIS\$SET_LOSE_KB_AST

Enables an AST routine that is executed when the specified virtual keyboard is detached from the physical keyboard.

Format

UIS\$SET_LOSE_KB_AST *kb_id* [*,astadr* [*,astprm*]]

Returns

UIS\$SET_LOSE_KB_AST signals all errors; no condition values are returned.

Arguments

kb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the **kb_id** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The **astadr** argument is the address of the entry mask to a procedure that is called at AST level whenever the virtual keyboard is disconnected from the physical keyboard.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine.

UIS\$SET_LOSE_KB_AST

Calls to `UIS$SET_LOSE_KB_AST` in VAX FORTRAN application programs should reference this argument as follows: `%REF(%LOC(astprm))`.

Description

To cancel the AST-enabling request of `UIS$SET_LOSE_KB_AST`, specify `0` in the **astadr** argument or omit the **astadr** and **astprm** arguments.

UIS\$SET_MOVE_INFO_AST

Enables an AST routine execution whenever the specified display viewport has been moved.

Format

UIS\$SET_MOVE_INFO_AST *wd_id*, [*astadr* [,*astprm*]]

Returns

UIS\$SET_MOVE_INFO_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The ***astadr*** argument is the address of an entry mask to a procedure that is called at AST level whenever the specified display viewport is moved.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The ***astprm*** argument is the address of a single argument or data structure, such as an array or record, that is passed to the AST routine.

Calls to UIS\$SET_MOVE_INFO_AST in VAX FORTRAN application programs should code this argument as follows: %REF(%LOC(astprm)).

Description

A MOVE notification AST can be used when an image needs to keep several display viewports in a particular arrangement. If one is moved, the AST routine can recreate the other display viewports in the correct positions around the moved viewport.

To cancel the AST-enabling request of UIS\$SET_MOVE_INFO_AST, perform any of the following actions:

- Specify the **wd_id** argument only.
- Specify *0* in the optional **astadr** argument.
- Omit the **astadr** and **astprm** arguments.

UIS\$SET_POINTER_AST

Allows an application to find out when the pointer is moved within, into, and out of a specified rectangle in the display window.

Format

UIS\$SET_POINTER_AST *vd_id, wd_id* [*astadr* [*astprm*]]
 [*x1, y1, x2, y2*] [*exitastadr*
 [*exitastprm*]]

Returns

UIS\$SET_POINTER_AST signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

UIS\$SET_POINTER_AST

AST routine. The **astadr** argument is the address of the entry mask to a procedure that is called at AST level whenever the pointer is moved within a rectangle in the virtual display.

To cancel the AST-enabling request of UIS\$SET_POINTER_AST for this argument only, specify 0 in the **astadr** argument and the coordinates of the rectangle.

astprm

VMS Usage: **user_arg**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as an array or record, passed to the AST routine. Calls to UIS\$SET_POINTER_AST in VAX FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

x₁, y₁, x₂, y₂

VMS Usage: **floating_point**
 type: **f_floating**
 access: **read only**
 mechanism: **by reference**

World coordinates of the rectangle. The **x₁** and **y₁** arguments are the addresses of **f_floating** point numbers that define the lower-left corner of the rectangle of the display window. The **x₂** and **y₂** arguments are the addresses of **f_floating** point numbers that define the upper-right corner of the rectangle of the display window.

If no rectangle is specified, the entire display window is assumed.

To cancel an AST-enabling request, specify 0 in either the **astadr** or the **exitastadr** arguments or both and the coordinates of the rectangle.

exitastadr

VMS Usage: **ast_procedure**
 type: **procedure entry mask**
 access: **read only**
 mechanism: **by reference**

Exit AST routine. The **exitastadr** argument is the address of the entry mask to a procedure that is called at AST level whenever the pointer leaves the rectangle.

To cancel the AST-enabling request of UIS\$SET_POINTER_AST for the EXIT AST routine only, specify 0 in the **exitastadr** argument and the coordinates of the rectangle.

18-330 UIS Routine Descriptions

UIS\$SET_POINTER_AST

exitastprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Exit AST parameter. The **exitastprm** argument is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine. Calls to **UIS\$SET_POINTER_AST** in FORTRAN application programs should be coded as follows: `%REF(%LOC(exitastprm))`.

Description

The Set Pointer AST routine also allows an application to keep track of the pointer in its own way. This routine can be called any number of times for different rectangles.

Note that an application need not enable both AST routines. It may specify one or the other.

UIS\$SET_POINTER_AST can be used by the application to highlight the display or some other application-specific function, as the user moves the pointer over specific areas of the display window. This might be used to define a number of regions within a menu, and execute an AST routine when the pointer enters or leaves any of these regions.

If both AST routines are enabled and the value 0 is specified in the **astadr** argument, the first AST routine is canceled.

To disable AST-enabling behavior for pointers entering a region, omit the **astadr** and **astprm** arguments.

To disable AST-enabling behavior for pointers leaving a region, omit the **exitastadr** and **exitastprm** arguments.

Pointer Region Priorities

UIS pointer regions are placed on the VAXstation screen in the order in which they are created. Therefore, if you create two overlapping viewports, and then use **UIS\$SET_POINTER_PATTERN**, **UIS\$SET_BUTTON_AST**, or **UIS\$SET_POINTER_AST** to define different pointer patterns for each viewport, the *correctness* of the result will depend on the order in which you both created the viewports and defined the cursor regions. For example, if you create the viewports and define the cursor patterns in the following manner, the viewport 1 cursor pattern will have a higher priority than viewport 2 cursor pattern in the overlapping region.

1. Create viewport 1
2. Create overlapping viewport 2
3. Define viewport 2 cursor pattern
4. Define viewport 1 cursor pattern

The preceding example causes the unexpected result that the viewport 1 cursor pattern will take priority over the viewport 2 cursor pattern in the overlapping region. This problem can be corrected by creating the viewports and defining the cursor patterns in the same order. To correct the problem, create the viewports and define cursor patterns in the following order:

1. Create viewport 1
2. Define viewport 1 cursor pattern
3. Create overlapping viewport 2
4. Define viewport 2 cursor pattern

The solution is for either UIS or your application to always pop the viewport before defining the cursor region for it.

UIS\$SET_POINTER_PATTERN

Allows an application to specify a special pointer cursor pattern for a specified rectangle in the virtual display.

Format

UIS\$SET_POINTER_PATTERN *vd_id, wd_id*
 [,pattern_array,
 pattern_count, activex,
 activey] [, x1, y1, x2, y2]
 [,flags]

Returns

UIS\$SET_POINTER_PATTERN signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

pattern_array

VMS Usage: **vector_word_unsigned**
type: **word_unsigned**
access: **read only**
mechanism: **by reference**

16- x 16-bit cursor pattern. The **pattern_array** argument is the address of one or more 16-bit arrays of 16 words that represents a bitmap image of the cursor pattern.

You can define two patterns that are executable on color and intensity systems using two arrays—a color plane and a mask plane. However, monochrome systems use a single array to specify the cursor pattern.

If two arrays are specified in an application running on a single-plane system, the first array is used.

NOTE: The bitmap image of the new pointer pattern is mapped in reverse order to the display screen.

pattern_count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of 16- x 16-bit cursor patterns defined. The **pattern_count** argument is the address of a longword that contains the number of cursor pattern arrays defined in the **pattern_array** argument.

activex, activey

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The **activex** and **activey** arguments are used to specify the actual bit in the cursor pattern that should be used to calculate the current pointer position. The arguments are expressed as bit offsets from the lower-left corner of the cursor pattern.

x₁, y₁, x₂, y₂

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

World coordinates of the rectangle in the virtual display. The **x₁** and **y₁** arguments are the addresses of **f_floating** point numbers that define the

18-334 **UIS Routine Descriptions** **UIS\$SET_POINTER_PATTERN**

lower-left corner of the rectangle in the display window. The x_2 and y_2 arguments are the addresses of *f_floating* point numbers that define the upper-right corner of the rectangle in the display window.

flags

VMS Usage: **longword_mask**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Flags. The **flags** argument is the address of a longword mask whose bits determine whether or not the cursor is confined to the display window rectangle.

When specified, **UIS\$M_BIND_POINTER** sets the appropriate bit in the mask.

Description

UIS\$SET_POINTER_PATTERN allows an application to specify a special pointer pattern to be used when the pointer is within the display window region specified by the optional rectangle. If no rectangle is given, then the entire display window is assumed. This function can be called any number of times for different rectangles.

To disable **UIS\$SET_POINTER_PATTERN**, omit the **pattern_array**, **pattern_count**, **activex**, **activey**, and **flags** arguments.

UIS\$SET_POINTER_POSITION

Specifies a new current pointer position in world coordinates. It is only effective if the new pointer position is within the specified display window and visible.

Format

status=UIS\$SET_POINTER_POSITION *vd_id*, *wd_id*, *x*, *y*

Returns

VMS Usage: **boolean**
type: **longword**
access: **write only**
mechanism: **by value**

Boolean value returned in a status variable or R0 (VAX MACRO). A status of 1 is returned, if the operation is successful, otherwise a 0 is returned.

UIS\$SET_POINTER_POSITION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

18-336 **UIS Routine Descriptions**
UIS\$SET_POINTER_POSITION

x, y

VMS Usage: **floating_point**

type: **f_floating**

access: **read only**

mechanism: **by reference**

World coordinates of the new pointer position. The **x** and **y** arguments are the addresses of **f_floating** point numbers that define the new pointer position.

UIS\$SET_POSITION

Sets the current position for text output. The current position is the point of alignment on the baseline of the next character to be output.

Format

UIS\$SET_POSITION *vd_id, x,y*

Returns

UIS\$SET_POSITION signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The ***vd_id*** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the ***vd_id*** argument.

x, y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

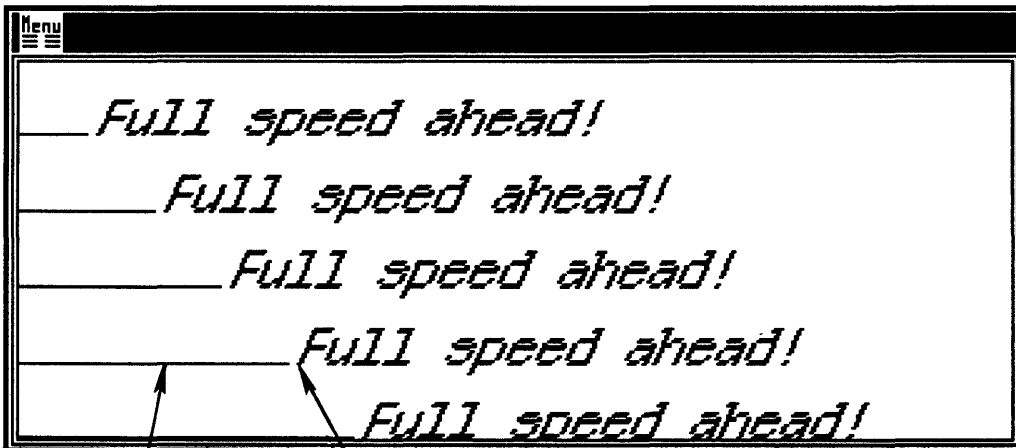
X and y world coordinate position. The ***x*** and ***y*** arguments are the addresses of ***f_floating*** point numbers that define the current position for text output.

18-338 UIS Routine Descriptions
UIS\$SET_POSITION

Example

```
REAL*4 Y  
DATA Y/4.0/  
  
DO I=1,5  
CALL UIS$SET_POSITION(VD_ID,FLOAT(I),Y)  
CALL UIS$PLOT(VD_ID,1,0.0,Y,FLOAT(I),Y)  
Y=Y-1.0  
CALL UIS$SET_FONT(VD_ID,1,1,'MY_FONT_11')  
CALL UIS$TEXT(VD_ID,1,'Full speed ahead!')  
ENDDO
```

Screen Output



Text Baseline

Current Text Position

UIS\$SET_RESIZE_AST

Specifies a user-requested AST routine to be executed when a display window has been resized using the user interface.

Format

```
UIS$SET_RESIZE_AST  vd_id, wd_id [,astadr [,astprm]]  
                    [,new_abs_x, new_abs_y]  
                    [,new_width, new_height]  
                    [,new_wc_x1, new_wc_y1,  
                    new_wc_x2, new_wc_y2]
```

Returns

UIS\$SET_RESIZE_AST signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

UIS\$SET_RESIZE_AST***astadr***

VMS Usage: **ast_procedure**
 type: **procedure entry mask**
 access: **read only**
 mechanism: **by reference**

AST routine. The **astadr** argument is the address of the entry mask of a procedure that is called at AST level whenever the "Change the size" item in the Window Options Menu is selected and a display window has been resized.

See the Description section for information about disabling UIS\$SET_RESIZE_AST.

astprm

VMS Usage: **user_arg**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine. Calls to UIS\$SET_RESIZE_AST in FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

new_abs_x, new_abs_y

VMS Usage: **floating_point**
 type: **f_floating**
 access: **write only**
 mechanism: **by reference**

Absolute device coordinate pair. The **new_abs_x** and **new_abs_y** arguments are the addresses of f_floating point longwords that receive the exact location of the newly resized display window in centimeters.

new_width, new_height

VMS Usage: **floating_point**
 type: **f_floating**
 access: **write only**
 mechanism: **by reference**

Width and height of the resized window. The **new_width** and **new_height** arguments are the addresses of f_floating point longwords that receive the dimensions of the newly resized display window in centimeters.

new_wc_x1, new_wc_y1, new_wc_x2, new_wc_y2

VMS Usage: **floating_point**

type: **f_floating**

access: **write only**

mechanism: **by reference**

World coordinates of the resized window. The **new_wc_x1** and **new_wc_y1** arguments are the addresses of **f_floating** point longwords that receive the world coordinates of the lower-left corner of the newly resized display window. The **new_wc_x2** and **new_wc_y2** arguments are the addresses of **f_floating** point longwords that receive the world coordinates of the upper-right corner of the newly resized display window.

Description

Typically, a call to **UIS\$SET_RESIZE_AST** in an application program indicates that the default resizing behavior is to be overridden.

By default, if a resize AST has not been enabled in an application program, **UIS** calls **UIS\$RESIZE_WINDOW**. If this behavior is not sufficient, the application program may call **UIS\$SET_RESIZE_AST** with its own resize routine.

To reenable the default behavior, specify **UIS\$_DEFAULT_RESIZE** as the **astadr** argument in a subsequent call to **UIS\$SET_RESIZE_AST**.

Resizing a window may be completely disabled in the following ways:

- By specifying the required **wd_id** argument and a value of 0 in the **astadr** argument
- By specifying only the required **wd_id** argument
- Omit the **astadr** and **astprm** arguments.

When window resizing is disabled, the option, "Change the size" displayed in the Window Options Menu changes from boldface to halftone.

The parameters for the resized window's new location, dimensions, and world coordinate range will not be overwritten with subsequent values until the AST has completed.

18-342 UIS Routine Descriptions

UIS\$SET_RESIZE_AST

Example

```
.
.
VD_ID=UIS$CREATE_DISPLAY(1.0,1.0,40.0,40.0,15.0,15.0) ❶
WD_ID=UIS$CREATE_WINDOW(VD_ID,'SYS$WORKSTATION','RESIZE',
2      5.0,5.0,25.0,25.0) ❷
.
.
CALL UIS$SET_RESIZE_AST(VD_ID,WD_ID,RESIZE_ME,0,NEW_ABS_X,NEW_ABS_Y,
2      NEW_WIDTH,NEW_HEIGHT,NEW_WC_X1,NEW_WC_Y1,
2      NEW_WC_X2,NEW_WC_Y2) ❸
CALL SYS$HIBER()
.
.
END      !end of main program
.
.
SUBROUTINE RESIZE_ME ❹
.
.
CALL UIS$RESIZE_WINDOW(VD_ID,WD_ID,NEW_ABS_X,NEW_ABS_Y,, ,
2      1.0,1.0,40.0,40.0) ❺
.
.
RETURN
END
.
.
```

In the preceding example, the call to `UIS$CREATE_DISPLAY` ❶ establishes the initial viewport size as a square.

The coordinate space of the initial display window is defined to be a subset of the virtual display ❷. When the original window is displayed it will show only a portion of the virtual display.

UIS\$SET_RESIZE_AST

The call to `UIS$SET_RESIZE_AST` ③ indicates that the program will override the default window resizing operation by enabling a user-written AST routine `RESIZE_ME` ④.

The parameter list of `UIS$RESIZE_WINDOW` ⑤ indicates how the resize operation is redefined. The absolute position and size of all viewports will default as usual to the final position and dimensions of the stretchy box.

However, the world coordinate range of the newly resized window is defined explicitly as the coordinate range of the virtual display. All newly resized windows will show the entire virtual display. If you tried to resize a previously resized window, you would still see the contents of the entire virtual display.

Distortion of objects displayed in the viewport will occur whenever the aspect ratio of the newly resized viewport does not equal the aspect ratio of the newly resized display window.

UIS\$SET_SHRINK_TO_ICON_AST

Specifies a user-requested AST routine to be executed whenever a display viewport is shrunk using the human interface.

Format

UIS\$SET_SHRINK_TO_ICON_AST *wd_id* [*,astadr* [*,astprm*]]

Returns

UIS\$SET_SHRINK_TO_ICON_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The **astadr** argument is the address of an entry mask of a procedure called at AST level whenever "Shrink to Icon" item in the Window Options Menu is selected.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine.

Calls to UIS\$SET_SHRINK_TO_ICON_AST in VAX FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

Description

The user interface for replacing a display viewport with an icon can be disabled by calling UIS\$SET_SHRINK_TO_ICON_AST with the **wd_id** only.

To reenble the default behavior of UIS\$SET_SHRINK_TO_ICON_AST, specify the constant UIS\$C_DEFAULT_SHRINK_TO_ICON in the **astadr** argument.

UIS\$SET_TB_AST

Specifies a user-requested AST routine to be executed whenever the digitizer lies within a specified rectangle on the tablet.

Format

UIS\$SET_TB_AST *tb_id*, [*data_astadr*,
 [*data_astprm*]], [*x_pos*, *y_pos*]
 [*data_x1*, *data_y1*, *data_x2*, *data_y2*]
 [*button_astadr*
 [*button_astprm*], *button_keybuf*]

Returns

UIS\$SET_TB_AST signals all errors; no condition values are returned.

Arguments

tb_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Tablet identifier. The ***tb_id*** argument is the address of a longword that uniquely identifies the tablet. See UIS\$CREATE_TB for more information about the ***tb_id*** argument.

data_astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The ***data_astadr*** argument is the address of an entry mask of a procedure that is called at AST level for each data point whenever the digitizer is moved within the specified active data region defined on the tablet.

See the Description section for information about disabling the digitizing region.

data_astprm

VMS Usage: **user_arg**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **reference**

AST parameter. The **data_astprm** is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine. Calls to UIS\$SET_TB_AST in VAX FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

x_pos,y_pos

VMS Usage: **floating_point**
 type: **f_floating**
 access: **write only**
 mechanism: **by reference**

Absolute device coordinate pair. The **x_pos**, **y_pos** arguments are the addresses of **f_floating** longwords that receive the current x and y tablet positions in centimeters relative to the lower-left corner of the tablet, when a data AST occurs.

data_x1,data_y1

VMS Usage: **floating_point**
 type: **f_floating**
 access: **read only**
 mechanism: **by reference**

Absolute device coordinate pair. The **data_x1**,**data_y1** arguments are the addresses of **f_floating** point numbers that define the lower-left corner of the data or digitizer region specified on the tablet. The data rectangle defines an area on the tablet in which data should be collected.

data_x2,data_y2

VMS Usage: **floating_point**
 type: **f_floating**
 access: **read only**
 mechanism: **by reference**

Absolute device coordinate pair. The **data_x2**,**data_y2** arguments are the addresses of **f_floating** point numbers that define the upper-right corner of the data or digitizer region specified on the tablet.

button_astadr

VMS Usage: **ast_procedure**
 type: **procedure entry mask**
 access: **read only**
 mechanism: **by reference**

18-348 UIS Routine Descriptions

UIS\$SET_TB_AST

AST routine. The **button_astadr** argument is the address of an entry mask of a procedure that is called at AST level whenever a button is depressed or released within the specified active data region defined on the tablet.

See the "DESCRIPTION" section for information about disabling the digitizing region.

button_astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **reference**

AST parameter. The **button_astprm** is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine. Calls to UIS\$SET_TB_AST in VAX FORTRAN application programs should be coded as follows: %REF(%LOC(astprm)).

button_keybuf

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Button information. The **button_keybuf** argument is the address of a longword that receives button information.

Description

The data rectangle specifies the active data region on the tablet. Only points within this rectangle are returned to the application. The data rectangle is specified using a centimeter coordinate system that is based at the lower-left corner of the tablet.

If no data rectangle is specified, the entire tablet is assumed.

Button AST Routines

To disable button AST routines, specify *0* in the **button_ast_rtn** argument.

UIS\$SET_TEXT_FORMATTING

Sets the text formatting justification mode.

Format

UIS\$SET_TEXT_FORMATTING *vd_id, iatb, oatb, mode*

Returns

UIS\$SET_TEXT_FORMATTING signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword that identifies a newly modified attribute block.

18-350 **UIS Routine Descriptions**
UIS\$SET_TEXT_FORMATTING

mode

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Text formatting mode. The **mode** argument is the address of a longword mask that sets the text formatting mode. The following table lists valid text formatting modes.

Formatting Mode	Function
UIS\$_TEXT_FORMAT_LEFT	Left justified, ragged right
UIS\$_TEXT_FORMAT_RIGHT	Right justified, left ragged
UIS\$_TEXT_FORMAT_CENTER	Centered line between left and right margin
UIS\$_TEXT_FORMAT_JUSTIFY	Justified lines, space filled to right margin
UIS\$_TEXT_FORMAT_NOJUSTIFY	No text justification (default)

All other values are reserved to DIGITAL for future use.

Description

Text justification occurs at the end of every UIS\$TEXT or UIS\$MEASURE_TEXT call. Text justification also occurs when a UIS\$_TEXT_NEW_LINE item is encountered in a UIS\$TEXT or UIS\$MEASURE_TEXT control list. The formatting mode and margins that are used are based on either the attribute block specified in the routine call or the last attribute block specified before the UIS\$_TEXT_NEW_LINE item code is encountered.

NOTE: Lines of text that do not fit completely within the margins will extend beyond the margin.

Example

```
.  
. .  
CALL UIS$SET_TEXT_MARGINS(VD_ID,0,1,3.0,27.0,24.0)  
CALL UIS$PLOT(VD_ID,0,3.0,30.0,3.0,0.0)  
CALL UIS$PLOT(VD_ID,0,27.0,30.0,27.0,0.0)  
  
CALL UIS$SET_TEXT_FORMATTING(VD_ID,1,1,UIS$C_TEXT_FORMAT_JUSTIFY)  
CALL UIS$SET_ALIGNED_POSITION(VD_ID,1,3.0,28.0)  
CALL UIS$SET_FONT(VD_ID,1,2,'MY_FONT_8')  
  
DO I= 1,4  
CALL UIS$TEXT(VD_ID,2,'What has been, may be')  
CALL UIS$NEW_TEXT_LINE(VD_ID,2)  
ENDDO  
  
. .  
. .
```

Screen Output

Menu	
left justified	
Sooner begun, sooner done	
Sooner begun, sooner done	
Sooner begun, sooner done	
Sooner begun, sooner done	

Menu	
right justified	
	The biter is sometimes bit
	The biter is sometimes bit
	The biter is sometimes bit
	The biter is sometimes bit

Menu	
centered	
A crowd is no company	
A crowd is no company	
A crowd is no company	
A crowd is no company	

Menu	
fully justified	
What	has been, may be
What	has been, may be
What	has been, may be
What	has been, may be

UIS\$SET_TEXT_MARGINS

Sets the text margins for a line of text.

Format

UIS\$SET_TEXT_MARGINS *vd_id ,iatb ,oatb ,x ,y
,margin_length*

Returns

UIS\$SET_TEXT_MARGINS signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword that identifies an attribute block.

18-354 **UIS Routine Descriptions**
UIS\$SET_TEXT_MARGINS

x,y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Starting margin position. The **x,y** arguments are the addresses of **f_floating** numbers that define a point on the margin. The margin is the minor text path when slope equals zero.

margin_length

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Ending margin position. The **margin_length** is the address of an **f_floating** number that defines the distance in world coordinates from the starting margin to the end margin.

Description

Lines of text do not automatically wrap to the next line.

UIS\$SET_TEXT_PATH

Sets the direction of text drawing and the direction of new text lines.

Format

UIS\$SET_TEXT_PATH *vd_id, iatb, oatb, major [,minor]*

Returns

UIS\$SET_TEXT_PATH signal all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a number that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The *oatb* argument is the address of a number that identifies a modified attribute block.

18-356 UIS Routine Descriptions

UIS\$SET_TEXT_PATH

major

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Major text path. The **major** argument is the address of a symbol that identifies the major text path type. The major path of text drawing is the direction of text drawing along a line. See the Description section for more information.

minor

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Minor text path. The **minor** argument is the address of a symbol that identifies the minor text path type. The minor path of text drawing refers to the direction of new text line creation. See the Description section for more information.

Description

The following table contains symbols for valid character drawing directions.

Path	Direction
UIS\$C_TEXT_PATH_RIGHT	Left to right (default major text path)
UIS\$C_TEXT_PATH_LEFT	Right to left
UIS\$C_TEXT_PATH_UP	Bottom to top
UIS\$C_TEXT_PATH_DOWN	Top to bottom (default minor text path)

Example

.
.
.

```
CALL UIS$SET_TEXT_PATH(VD_ID,0,1,UIS$C_TEXT_PATH_LEFT,  
2      UIS$C_TEXT_PATH_DOWN)  
CALL UIS$SET_FONT(VD_ID,1,1,'MY_FONT_5')  
CALL UIS$SET_ALIGNED_POSITION(VD_ID,1,38.0,38.0)
```

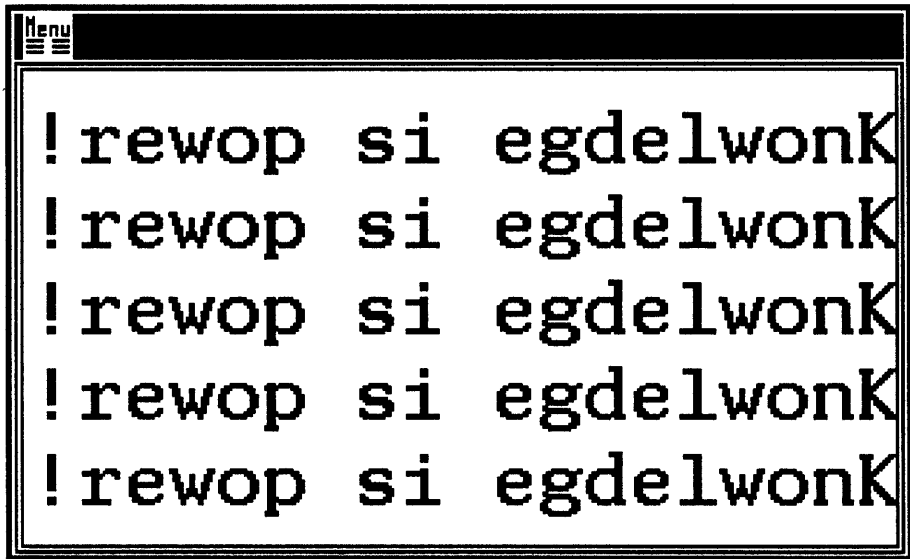


```
CALL UIS$TEXT(VD_ID,1,'Knowledge is power!')  
CALL UIS$NEW_TEXT_LINE(VD_ID,1)
```

·
·
·

The preceding example illustrates how to alter the default major text drawing path to produce the output shown in the next section.

Screen Output



UIS\$SET_TEXT_SLOPE

Sets the angle of the actual path of text drawing relative to the major path.

Format

UIS\$SET_TEXT_SLOPE *vd_id ,iatb ,oatb ,angle*

Returns

UIS\$SET_TEXT_SLOPE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a number that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a number that identifies an attribute block.

angle

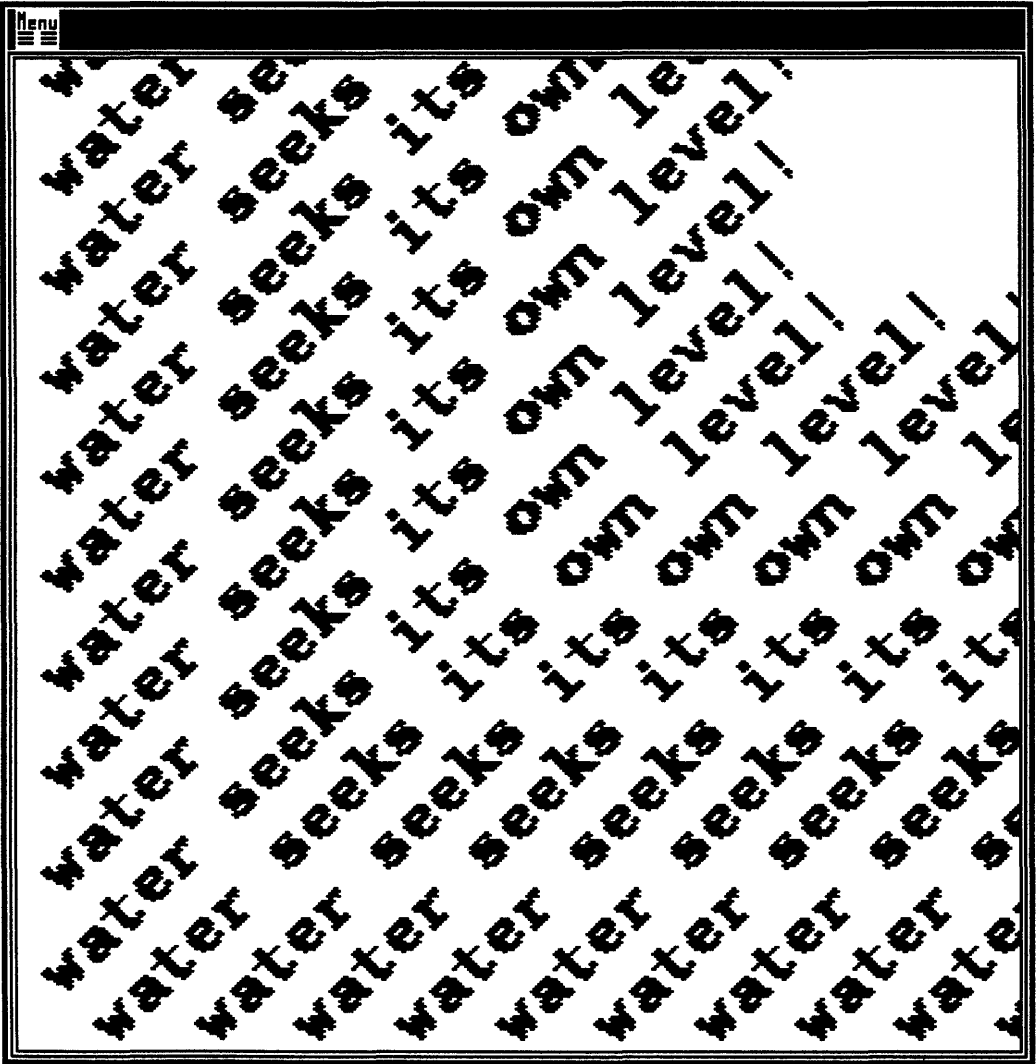
VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Angle of text slope. The **angle** argument is the address of an **f_floating** point number that defines the angle of the actual path of text drawing relative to the major path measured counterclockwise in degrees. The default angle of text slope is 0 degrees.

Example

```
.  
. .  
CALL UIS$SET_FONT(VD_ID,0,1,'MY_FONT_13')  
CALL UIS$SET_TEXT_SLOPE(VD_ID,1,2,45.0)  
  
DO I=1,10  
CALL UIS$SET_ALIGNED_POSITION(VD_ID,2,0.0,Y)  
CALL UIS$TEXT(VD_ID,2,'water seeks its own level!')  
Y=Y-2.0  
ENDDO  
  
PAUSE  
  
DO I=1,10  
CALL UIS$SET_ALIGNED_POSITION(VD_ID,2,X,1.0)  
CALL UIS$TEXT(VD_ID,2,'water seeks its own level!')  
X=X+2.0  
ENDDO  
  
.  
.  
.
```

Screen Output



UIS\$SET_WRITING_INDEX

Sets the writing color index for text and graphics output.

Format

UIS\$SET_WRITING_INDEX *vd_id, iatb, oatb, index*

Returns

UIS\$SET_WRITING_INDEX signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword integer that specifies the attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword integer that identifies the newly modified attribute block.

18-362 **UIS Routine Descriptions**
UIS\$SET_WRITING_INDEX

index

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Color map index. The **index** argument is the address of a longword integer that specifies a color map index. If the index exceeds the maximum index for the associated color map, an error is signaled.

UIS\$SET_WRITING_MODE

Sets the text and graphics mode.

Format

UIS\$SET_WRITING_MODE *vd_id, iatb, oatb, mode*

Returns

UIS\$SET_WRITING_MODE signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The *vd_id* argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the *vd_id* argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a longword integer that specifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The *oatb* argument is the address of a longword integer that specifies a newly modified attribute block that controls the writing mode.

18-364 **UIS Routine Descriptions**
UIS\$SET_WRITING_MODE

mode

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Writing mode. The **mode** argument is the address of a longword that specifies the writing mode (UIS\$C_MODE_XXXX). The default writing mode is overlay.

Description

Table 9-2 lists and describes all UIS writing modes.

UIS\$SHRINK_TO_ICON

Replaces a display viewport with its associated icon.

Format

UIS\$SHRINK_TO_ICON *wd_id* [,*icon_wd_id*] [,*icon_flags*]
[,*icon_name*] [,*attributes*]

Returns

UIS\$SHRINK_TO_ICON signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

icon_wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Icon window identifier. The *icon_wd_id* argument is the address of a longword that uniquely identifies an icon.

icon_flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Icon flags. The *icon_flags* is the address of a longword mask of flags that may be used to specify whether default icon behavior should be extended to an application-supplied icon. By default, no modifications are made to the application-supplied icon. The following table lists valid icon flags.

18-366 **UIS Routine Descriptions**
UIS\$SHRINK_TO_ICON

Flag	Function
UIS\$M_ICON_DEF_KB	UIS manages keyboard ownership. If the display window is enabled for keyboard ownership, UIS\$DISABLE_VIEWPORT_KB is called during window shrinking and UIS\$ENABLE_KB is called during icon expansion.
UIS\$M_ICON_DEF_BODY	UIS places a button AST region over the body of the icon window and uses that AST to trigger icon expansion.
All other bits	The remaining bits are set to 0 and are reserved to DIGITAL.

icon_name

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Icon name. The **icon_name** argument is the address of a descriptor of the text to be used as the icon name.

attributes

VMS Usage: **item_list_pair**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Window attributes list. The **attributes** argument is the address of data structure, such as an array or record. The **attributes** argument may be used to specify exact placement of the icon on the display screen.

The following figure describes the structure of the window attributes list.

UIS Routine Descriptions 18-367
UIS\$SHRINK_TO_ICON

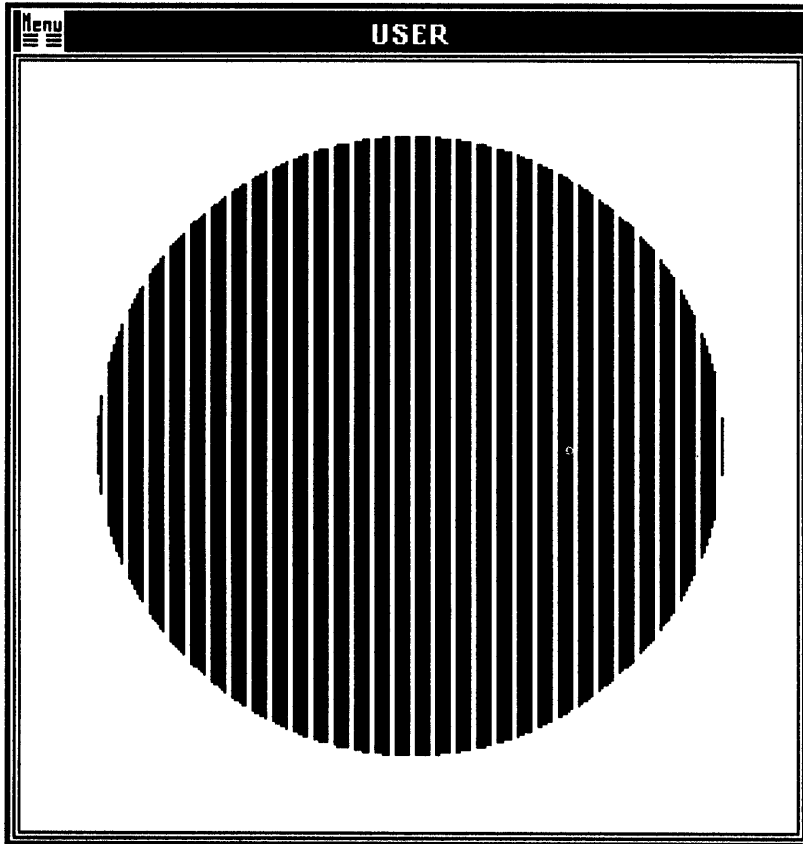
Attribute ID code (WDPL\$C__xxx)
Longword value for attribute identified in previous longword
2nd attribute ID code
2nd attribute value
• • •
End of list :- 0 (WDPL\$C__END__OF__LIST)

ZK-4581-85

See UIS\$CREATE_WINDOW for more information.

18-368 UIS Routine Descriptions
UIS\$SHRINK_TO_ICON

Screen Output



UIS\$SOUND_BELL

Actuates the keyboard bell to ring once.

Format

UIS\$SOUND_BELL *devnam* [,*bell_volume*]

Returns

UIS\$SOUND_BELL signals all errors; no condition values are returned.

Arguments

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name string. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify 'SYS\$WORKSTATION' as the default device name.

bell_volume

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Bell volume level. The **bell_volume** argument is the address of a longword that specifies the bell volume. The **bell_volume** argument can be supplied explicitly as a number from 0 to 8, where 0 is the most quiet; and 8 is the loudest. If the **bell_volume** argument is not specified, the default volume specified in the workstation setup menu is used.

Description

On the LK201 keyboard, the bell sound differs from a key click sound in the frequency and tone.

UIS\$SOUND_CLICK

Actuates the keyboard click sound once.

Format

UIS\$SOUND_CLICK *devnam* [,*click_volume*]

Returns

UIS\$SOUND_CLICK signals all errors; no condition values are returned.

Arguments

devnam

VMS Usage: **device_name**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Device name string. The **devnam** argument is the address of a character string descriptor of the workstation device name. Specify SYS\$WORKSTATION as the device name.

click_volume

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Key click volume level. The **click_volume** argument is the address of a longword that specifies the key click volume level. The **click_volume** argument is specified explicitly as a number from 0 to 8, where 0 is the most quiet and 8 is the loudest. If the **click_volume** argument is not specified, the default volume is used from the workstation setup menu mechanism.

Description

On the LK201 keyboard, the key click sound differs from a bell sound in the frequency and tone.

UIS\$TEST_KB

Returns a boolean value indicating whether the physical keyboard is currently bound to the specified virtual keyboard.

Format

status=**UIS\$TEST_KB** *kb_id*

Returns

VMS Usage: **boolean**
type: **longword**
access: **write only**
mechanism: **by value**

Boolean value returned in a status variable or R0 (VAX MACRO). The boolean value TRUE is returned if the physical keyboard is bound to the virtual keyboard, otherwise a boolean value FALSE is returned.

UIS\$TEST_KB signals all errors; no condition values are returned.

Arguments

kb_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual keyboard identifier. The **kb_id** argument is the address of a longword that uniquely identifies a virtual keyboard. See UIS\$CREATE_KB for more information about the **kb_id** argument.

UIS\$TEXT

Draws a series of characters.

Format

UIS\$TEXT *vd_id, atb, text_string* [,x,y] [,ctllist ,ctllen]

Returns

UIS\$TEXT signals all errors; no condition values are returned.

Arguments

vd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **vd_id** argument is the address of a longword that uniquely identifies a virtual display. See UIS\$CREATE_DISPLAY for more information about the **vd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that specifies an attribute block that modifies text output. When a control list is specified, the **atb** argument defines the initial attribute settings of the text string.

text_string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Text string. The **text_string** argument is the address of a character string descriptor of a text string.

x, y

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Starting point of text output. The **x** and **y** arguments are the addresses of **f_floating** point numbers that define in world coordinates of the starting point of text output. The starting point is the upper-left corner of the character cell of the next character to be drawn.

If this argument is not specified, the current text position is used. (See the **UIS\$SET_ALIGNED_POSITION** routine for more information.)

When a control list is specified, the **x,y** arguments specify the starting coordinate for the first character of the character string.

ctl

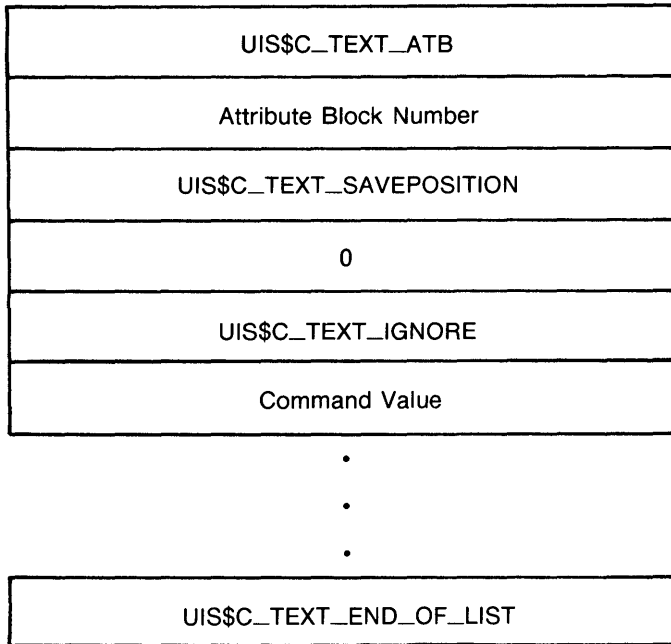
VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Text formatting control list. The **ctl** argument is the address of an array of longwords that define the font, text rendition, text formatting, and positioning of fragments of the text string. When a control list is specified, the **atb** argument defines the initial attribute settings of the text string

If **ctl** is not specified, text rendition and position are the values specified in the arguments **atb** and **x,y**.

The control list consists of a sequence of data elements, each two longwords in length. The first longword of each element is a tag. The second longword is either a value particular to the type of element specified or zero. Following is a diagram showing the structure of a text control list.

18-374 **UIS Routine Descriptions**
UIS\$TEXT



ZK-5426-86

The following table describes valid formatting commands.

Formatting Command	Function
Commands Without Values¹	
UIS\$C_TEXT_NOP	Nil operation
UIS\$C_TEXT_RESTORE_POSITION	Restores the current writing position
UIS\$C_TEXT_SAVE_POSITION	Saves the current writing position
Commands Requiring Values	
UIS\$C_TEXT_ATB	Specifies an attribute block number
UIS\$C_TEXT_HPOS_ABSOLUTE	Specifies a new current x position
UIS\$C_TEXT_HPOS_RELATIVE	Modifies the current x position by a delta
UIS\$C_TEXT_IGNORE	Skips <i>n</i> characters

¹Second longword must be zero

Formatting Command	Function
Commands Requiring Values	
UIS\$C_TEXT_NEW_LINE	Skips <i>n</i> new lines and positions at the left margin
UIS\$C_TEXT_TAB_ABSOLUTE	Writes white space to the new absolute position
UIS\$C_TEXT_TAB_RELATIVE	Writes white space to the new relative position
UIS\$C_TEXT_VPOS_ABSOLUTE	Writes a new current y position
UIS\$C_TEXT_VPOS_RELATIVE	Modifies the current y position by a delta
UIS\$C_TEXT_WRITE	Writes <i>n</i> characters
Commands Not Requiring a Second Longword	
UIS\$C_TEXT_END_OF_LIST	Terminates the control list

When UIS encounters illegal commands and values within the control list, it skips the invalid item and signals an error.

ctllen

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Length of formatting control list. The **ctllen** argument is the address of a longword that specifies the length of the formatting control list in longwords.

Description

Nonprinting characters such as tab and line feed are not handled in any special way. The character is obtained from the font and is displayed like any other character.

UIS\$TRANSFORM_OBJECT

Transforms the coordinates or attributes or both of the specified object within the display list.

Format

UIS\$TRANSFORM_OBJECT $\left\{ \begin{array}{l} \text{obj_id} \\ \text{seg_id} \end{array} \right\} [,matrix] [,atb]$

Returns

UIS\$TRANSFORM_OBJECT signals all errors; no condition values are returned.

Arguments

obj_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Object identifier. The **obj_id** argument is the address of a longword that uniquely identifies the object.

seg_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Segment identifier. The **seg_id** argument is the address of a longword that uniquely identifies the segment. See UIS\$BEGIN_SEGMENT for more information about the **seg_id** argument.

matrix

VMS Usage: **vector_longword_signed**
type: **F_floating**
access: **read only**
mechanism: **by reference**

Transformation matrix. The **matrix** argument is the address of an array of longword integers that define the values to be used for scaling, rotation,

and/or translation. A two-dimensional array declared as ARRAY(2,3) has the following structure.

1,1	1,2	1,3
2,1	2,2	2,3

ZK-5492-86

VAX FORTRAN allocates memory for the array elements. Memory addresses of array elements range from lowest to highest in the following order: (1,1),(2,1), (1,2),(2,2),(1,3), and (2,3). UIS assigns values to array elements in the order shown in the following illustration.

NOTE: For the purposes of assigning values to array elements, UIS treats all transformation matrices as VAX FORTRAN arrays regardless of the programming language of the application.

1	3	5
2	4	6

ZK-5493-86

Pairs of array elements govern how displayed objects are scaled, rotated, and translated. UIS computes the transformed coordinates in the following manner.

$$x_1 = A(1,1)*x + A(1,2)*y + A(1,3)$$

$$y_1 = A(2,1)*x + A(2,2)*y + A(2,3)$$

Translation

When translation alone is performed, the following array elements are assigned values. D_x and D_y represent distances between the original coordinates and the new coordinates.

18-378 **UIS Routine Descriptions**
UIS\$TRANSFORM_OBJECT

1	0	Dx
0	1	Dy

ZK-5494-86

Scaling

When scaling alone is performed, the following array elements are assigned values.

Sx	0	0
0	Sy	0

ZK-5495-86

Rotation

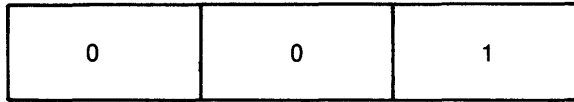
When rotation alone is performed, the following array elements are assigned values, where "@" is the desired angle of rotation. The values returned from the FORTRAN SIN and COS functions are stored in the appropriate array elements.

cos (@)	sin (@)	0
-sin (@)	cos (@)	0

ZK-5496-86

An unlimited number of transformations can be performed at one time by simply multiplying the matrices together into a single matrix using matrix multiplication.

In order to multiply two matrices together, you must add a row to the bottom of each matrix.



ZK-5461-86

After the multiplication is performed, remove the last row of the result.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block to override current attribute settings.

Description

Either the coordinates can be transformed, or the attributes can be overridden or both.

After a transformation, occluded objects may not appear correctly. This can be corrected by calling UIS\$EXECUTE to refresh the display screen.

Example

```
.  
. .  
. .  
REAL*4 MATRIX(2,3)  
. .  
. .  
CALL UIS$PLOT(VD_ID,0,5.0,5.0,15.0,5.0,10.0,15.0,5.0,5.0)  
CURRENT_ID=UIS$GET_CURRENT_OBJECT(VD_ID)  
OBJ_ID=CURRENT_ID
```

18-380 **UIS Routine Descriptions**
UIS\$TRANSFORM_OBJECT

```
CALL UIS$SET_FONT(VD_ID,0,1,'UIS$FILL_PATTERNS')  
CALL UIS$SET_FILL_PATTERN(VD_ID,1,1,PATT$C_HORIZ1_7)
```

PAUSE

```
MATRIX(1,1)=1.0  
MATRIX(2,1)=0.0  
MATRIX(1,2)=0.0  
MATRIX(2,2)=1.0  
MATRIX(1,3)=-10.0  
MATRIX(2,3)=-10.0  
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX,1)
```

PAUSE

```
MATRIX(1,1)=2.0  
MATRIX(2,1)=0.0  
MATRIX(1,2)=0.0  
MATRIX(2,2)=2.0  
MATRIX(1,3)=0.0  
MATRIX(2,3)=0.0  
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX,1)
```

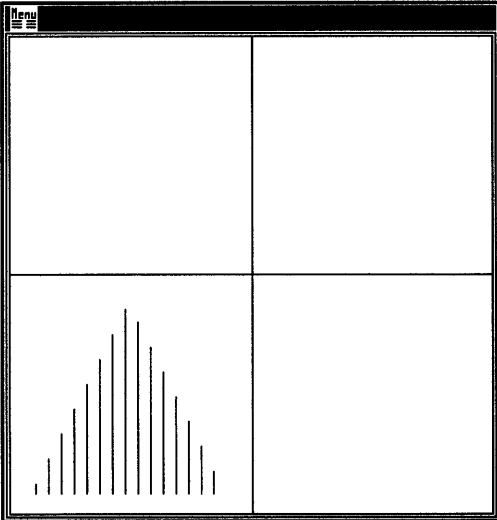
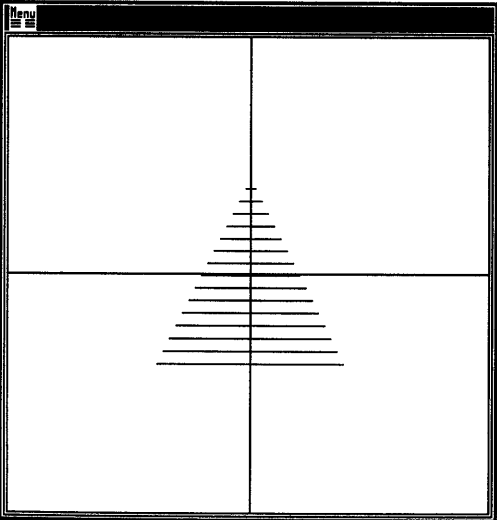
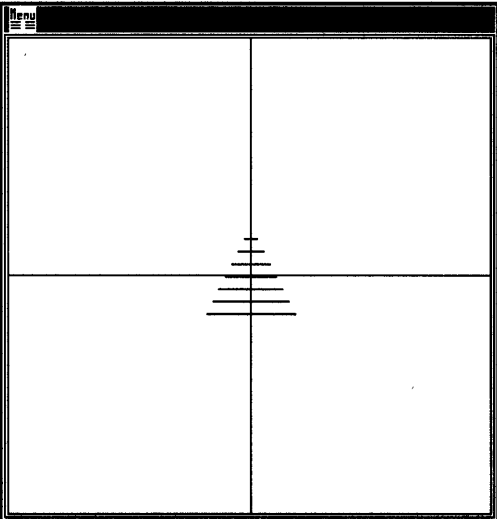
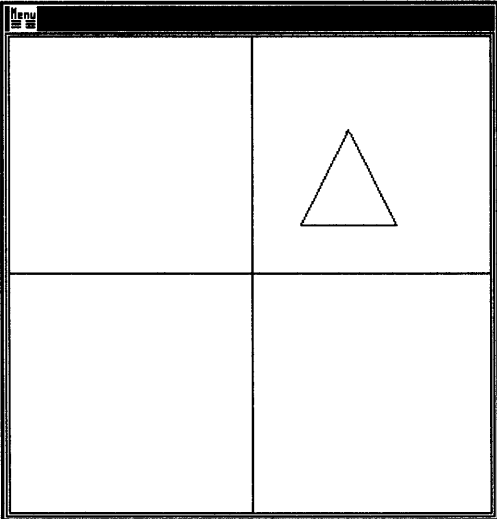
PAUSE

```
CALL UIS$SET_FONT(VD_ID,0,2,'UIS$FILL_PATTERNS')  
CALL UIS$SET_FILL_PATTERN(VD_ID,2,2,PATT$C_VERT1_7)
```

```
MATRIX(1,1)=1.0  
MATRIX(2,1)=0.0  
MATRIX(1,2)=0.0  
MATRIX(2,2)=1.0  
MATRIX(1,3)=-13.0  
MATRIX(2,3)=-13.0  
CALL UIS$TRANSFORM_OBJECT(OBJ_ID,MATRIX,2)
```

.
. .
.

Screen Output



PART IV UIS Device Coordinate (UISDC) Routines

Chapter 19

UIS Device Coordinate Graphics Routines

19.1 Overview

This section introduces the MicroVMS workstation UISDC (device coordinate) graphics system services. It contains a reference section of all UISDC routines and pertinent information on how they are used.

19.2 UISDC Routines—How to Use Them

In addition to the world coordinate interface (UIS), the MicroVMS workstation software provides a device-coordinate, or pixel-level, interface (UISDC) to the graphics system services. UISDC allows applications to create UIS windows, but manipulate the contents of those windows at the pixel level.

Programming in device coordinates requires that an application make mixed use of UIS and UISDC routines. Only those UIS routines that use or modify world coordinate positions have been duplicated as UISDC routines. Most informational, attribute, windowing, and display routines exist only in UIS format and are shared by the two programming levels.

The major differences between UISDC and UIS are:

- The UISDC drawing surface is a display window, as opposed to a virtual display as it is with UIS. Therefore, the UISDC output routines utilize display window identifiers instead of virtual display identifiers.
- Most UISDC positions are expressed in viewport-relative device coordinates.

The lower-left corner of the display viewport is pixel (0,0). The upper-right corner is (width multiplied by x resolution, height multiplied by y resolution), where width and height are expressed in centimeters and resolution is expressed in pixels per centimeter.

19-2 UIS Device Coordinate Graphics Routines

- UISDC does not maintain or manage a display list. Automatic zooming, panning, and playback of a display are not supported.

Mixed use of UIS and UISDC output routines is allowed. Therefore, it is possible to do the following UIS and UISDC operations simultaneously:

- Draw to a virtual display that contains a window, using world coordinates.
- Draw directly to the same window, using viewport-relative device coordinates.

Separate current text positions, character size, text margins, and clipping rectangles are maintained for both coordinate systems.

The following section of this chapter lists the UISDC routines and their arguments in alphabetical order.

UISDC\$ALLOCATE_DOP

Allocates a drawing operation primitive (DOP) for a particular display window in VAXstation color and intensity systems.

Format

dop=UISDC\$ALLOCATE_DOP *wd_id*,*size*,*atb*

Returns

VMS Usage: **address**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as the address of the drawing operation primitive in the variable *dop* or R0 (VAX MACRO).

UISDC\$ALLOCATE_DOP signals all errors; no condition values are returned.

Arguments

wd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

size
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

Size of the variable portion of the drawing operation primitive. The ***size*** argument is the address of a number that defines the size of the variable portion of the drawing operation primitive to allocated.

19-4 UISDC Routines

UISDC\$ALLOCATE_DOP

The size of the variable portion of the allocated DOP is returned in the size field. The size of the allocated DOP may be smaller than the requested size.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of an attribute block.

Description

UISDC\$ALLOCATE_DOP writes the following information from the specified attribute block into portions of the DOP data structure and returns the address of the DOP.

- Clipping rectangle
- Writing mode
- Writing mask

See the *MicroVMS Workstation Video Device Driver Manual* for more information.

UISDC\$CIRCLE

Draws an arc along the circumference of a circle.

Format

UISDC\$CIRCLE *wd_id, atb, center_x, center_y, xradius*
[,start_deg, end_deg]

Returns

UISDC\$CIRCLE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that specifies an attribute block that controls the appearance of the circle or arc.

center_x, center_y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Center position x and y viewport-relative device coordinates. The **center_x** and **center_y** arguments are the addresses of integers that define a point in the virtual display that is the center of the arc or circle.

19-6 UISDC Routines

UISDC\$CIRCLE

xradius

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Radius of the circle specified as an x viewport-relative device coordinate width. The **xradius** argument is the address of an integer that defines the distance from the center of the circle to the circumference of the circle.

start_deg, end_deg

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Degree at which the arc starts. The **start_deg** and **end_deg** arguments are the addresses of **f_floating** numbers that define the starting and ending point on the circumference of the circle where the arc or circle will be drawn. Degrees are measured clockwise from the top of the circle. If these arguments are not specified, 0.0 degrees and 360.0 degrees are assumed, respectively.

Description

UISDC\$CIRCLE draws an arc specified by a center position and a radius for the range of the degrees specified.

The arc is closed by drawing one or more lines between the endpoints. The arc type associated with the attribute block specifies the way in which the arc is closed. The arc is not closed off by default. See UIS\$SET_ARC_TYPE for details.

The points are drawn with the current line pattern and width, and filled with the current fill pattern, if enabled.

UISDC\$CIRCLE does not support the following combination of attributes:

- Line width not equal to 1 and line style not equal to **FFFFFFFF₁₆**
- Line width not equal to 1 and complement writing mode

Circles are distorted by virtual display/display window aspect ratio distortion.

UISDC\$ELLIPSE

Draws an arc at a starting position along the circumference of an ellipse.

Format

UISDC\$ELLIPSE *wd_id, atb, center_x, center_y, xradius, yradius [,start_deg ,end_deg]*

Returns

UISDC\$ELLIPSE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Virtual display identifier. The **wd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_DISPLAY for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies the attribute block that will modify the ellipse. If you specify 0 in the **atb** argument, the default settings of attribute block 0 are used.

center_x, center_y

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Center position x and y viewport-relative device coordinates. The **center_x** and **center_y** arguments are the addresses of integers that define a point in the display window that is the center of the ellipse or arc.

19-8 UISDC Routines

UISDC\$ELLIPSE

xradius

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Radius of the ellipse specified as an *x* device coordinate width. The **xradius** argument is the address of an integer that defines the distance from the center of the ellipse to the circumference of the ellipse or arc.

yradius

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Radius of the ellipse specified as a *y* device coordinate width. The **yradius** argument is the address of a integer that defines the distance from the center of the ellipse to the circumference of the ellipse or arc.

start_deg, end_deg

VMS Usage: **floating_point**
type: **f_floating**
access: **read only**
mechanism: **by reference**

Degree at which the arc starts and ends. The **start_deg** and **end_deg** arguments are the addresses of **f_floating** numbers that define the starting point and ending point in degrees on the circumference of the ellipse where the arc or ellipse will be drawn. Degrees are measured clockwise from the top of the ellipse.

If these arguments are not specified, *0.0* and *360.0* degrees are assumed. If both arguments are not specified, a complete ellipse is drawn.

Description

UISDC\$ELLIPSE uses center position coordinates and *x* and *y* radii to construct an ellipse. Along the circumference of this ellipse, UISDC\$ELLIPSE draws an arc for a specified range of degrees.

The arc is closed by drawing one or more lines between the endpoints. The type of arc associated with the attribute block specifies the way in which the arc is closed. See the UIS\$SET_ARC_TYPE routine for more information.

The points are drawn with the current line pattern and width, and filled with the current fill pattern, if enabled.

UISDC\$ELLIPSE does not create thick patterned ellipses and thick ellipses that are undefined in complement mode.

UISDC\$ELLIPSE does not support the following combination of attributes:

- Line width not equal to 1 and line style not equal to *FFFFFFFF*₁₆
- Line width not equal to 1 and complement writing mode

UISDC\$ERASE

Erases the specified rectangle in the display window.

Format

UISDC\$ERASE *wd_id* [*x₁*, *y₁*, *x₂*, *y₂*]

Returns

UISDC\$ERASE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies the display window containing the specified rectangle. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

x₁, *y₁*, *x₂*, *y₂*

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Viewport-relative device coordinate pairs. The *x₁* and *y₁* arguments are the addresses of integers that define the lower-left corner of the rectangle in the display window. The *x₂* and *y₂* arguments are the addresses of integers that define the upper-right corner of the rectangle in the display window. If no rectangle is specified, the entire display window is erased.

Description

Areas within display windows affected by this call are filled with the color specified by entry 0 in the virtual display color map.

UISDC\$EXECUTE_DOP_ASYNC

Starts the execution of the specified drawing operation primitive (DOP) in the specified display window of VAXstation color and intensity systems and returns control to the application immediately.

Format

UISDC\$EXECUTE_DOP_ASYNC *wd_id ,dop ,iosb*

Returns

UISDC\$EXECUTE_DOP_ASYNC signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

dop

VMS Usage: **vector_byte_unsigned**
type: **byte_unsigned**
access: **read only**
mechanism: **by reference**

Drawing operation primitive. The **dop** argument is the address of an array of bytes that comprise the drawing operation primitive.

iosb

VMS Usage: **io_status_block**
type: **quadword (unsigned)**
access: **write only**
mechanism: **by reference**

19-12 **UISDC Routines**
UISDC\$EXECUTE_DOP_ASYNCH

I/O status block. The **iosb** argument is the address of an I/O status block that receives a value indicating that the drawing operation primitive is queued for execution.

Description

UISDC\$EXECUTE_DOP_ASYNCH queues the specified DOP for execution in the specified window.

You may later use the SYS\$SYNCH system service to determine when the DOP has been drawn. See the *MicroVMS Workstation Video Device Driver Manual* for more details.

UISDC\$EXECUTE_DOP_SYNCH

Queues the drawing operation primitive (DOP), waits for the specified DOP to complete execution in the specified display window, and then returns control to the application.

Format

UISDC\$EXECUTE_DOP_SYNCH *wd_id ,dop*

Returns

UISDC\$EXECUTE_DOP_SYNCH signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

dop

VMS Usage: **vector_byte_unsigned**
type: **byte_unsigned**
access: **read only**
mechanism: **by reference**

Drawing operation primitive. The **dop** argument is the address of an array of bytes that comprises the drawing operation primitive.

Description

UISDC\$EXECUTE_DOP_SYNCH queues the specified drawing operation primitive for execution in the specified window and returns when the drawing operation is complete.

See *MicroVMS Workstation Video Device Driver Manual* for more information.

19-14 **UISDC Routines**
UISDC\$GET_ALIGNED_POSITION

UISDC\$GET_ALIGNED_POSITION

Returns the current position for text output—the upper-left corner of the next character cell.

Format

UISDC\$GET_ALIGNED_POSITION *wd_id, atb, retx, rety*

Returns

UISDC\$GET_ALIGNED_POSITION signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block. The **atb** argument is the address of a longword that identifies an attribute block that contains a modified font attribute.

retx, rety

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Viewport-relative device coordinate pair. The **retx** and **rety** arguments are the addresses of longwords that receive the current position as *x* and *y* viewport-relative device coordinate positions.

Description

UISDC\$GET_ALIGNED_POSITION differs from UISDC\$GET_POSITION in that the current position refers to the upper-left corner of the next character to be output using the specified attribute block. This is useful for applications that require the position of the upper-left corner, but do not have enough information about the font baseline to determine the proper alignment point. The position is converted into the proper alignment point using the font specified in the given attribute block. See UISDC\$SET_ALIGNED_POSITION.

UISDC\$GET_CHAR_SIZE

Returns both a value indicating whether or not character scaling is enabled and the character size used.

Format

boolean = **UISDC\$GET_CHAR_SIZE** *wd_id, atb*
,[char],[width],[height]

Returns

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword value returned as a Boolean to indicate the status of character scaling in the variable *boolean* or R0 (VAX MACRO).

UISDC\$GET_CHAR_SIZE signals all errors; no condition values are returned.

Arguments

wd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

atb
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The ***atb*** argument is the address of a longword that identifies an attribute block containing the character size attribute setting.

char

VMS Usage: **char_string**
type: **character_string**
access: **write only**
mechanism: **by descriptor**

Single character. The **char** argument is the address of a character string descriptor of a single char.

width

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Character width. The **width** argument is the address of a longword that receives the character width in viewport-relative device coordinates.

height

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Character height. The **height** argument is the address of a longword that receives the character height in viewport-relative device coordinates.

UISDC\$GET_CLIP

Returns the clipping mode.

Format

status=UISDC\$GET_CLIP *wd_id*, *atb* [*x*₁, *y*₁, *x*₂, *y*₂]

Returns

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Boolean value returned as the clipping mode in a status variable or R0 (VAX MACRO). If clipping is enabled, a Boolean TRUE is returned. If clipping is disabled, a Boolean FALSE is returned.

UISDC\$GET_CLIP signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The *atb* argument is the address of a longword that identifies the attribute block that modifies the clipping mode.

x_1, y_1, x_2, y_2

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Viewport-relative device coordinate pairs. The x_1 and y_1 arguments are the addresses of longwords that receive the viewport-relative device coordinates of the lower-left corner of the clipping rectangle. The x_2 and y_2 arguments are the addresses of longwords that receive the viewport-relative device coordinates of the upper-right corner of the clipping rectangle.

UISDC\$GET_POINTER_POSITION

Returns the current pointer position in viewport-relative device coordinates.

Format

status=UISDC\$GET_POINTER_POSITION *wd_id, retx, rety*

Returns

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Boolean value returned as the current position of the pointer in a status variable. UISDC\$GET_POINTER_POSITION returns the Boolean TRUE value 1 if the pointer is within the visible portion of the viewport, 0 is returned if the pointer is outside the visible portion of the viewport. In the latter case, the *x* and *y* values are returned as 0,0.

UISDC\$GET_POINTER_POSITION signals all errors; no condition values are returned.

Arguments

wd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

retx, rety
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Viewport-relative device coordinate pair. The *retx* and *rety* arguments are the addresses of longwords that receive the pointer position in viewport-relative device coordinates.

Description

Note that the returned status value should always be tested when using this routine, since it is always possible that the pointer could be outside the window when the service is called and the x,y values would be meaningless.

UISDC\$GET_POSITION

Returns the current baseline position for text output.

Format

UISDC\$GET_POSITION *wd_id, retx, rety*

Returns

UISDC\$GET_POSITION signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

retx, rety

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Viewport-relative device-coordinate pair. The **retx** and **rety** arguments are addresses of longwords that receive the current position of text output in viewport-relative device coordinate positions.

Description

UIS\$NEW_TEXT_LINE and UIS\$TEXT recognize the concept of current position. The position refers to the alignment point on the baseline of the next character to be output. (See the UIS\$SET_POSITION routine.)

UISDC\$GET_TEXT_MARGINS

Returns the text margins for a line of text. See UISDC\$SET_TEXT_MARGINS for more information.

Format

UISDC\$GET_TEXT_MARGINS *wd_id ,atb ,x ,y*
[,margin_length]

Returns

UISDC\$GET_TEXT_MARGINS signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Virtual display identifier. The **wd_id** argument is the address of a longword that uniquely identifies the virtual display. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block.

x,y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

19-24 **UISDC Routines**
UISDC\$GET_TEXT_MARGINS

Starting margin position. The *x,y* arguments are the addresses of longwords that receive the starting margin relative to the direction of text drawing in viewport-relative device coordinates.

margin_length

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Ending margin position. The **margin_length** is the address of a longword that receives the distance to the end margin in viewport-relative device coordinates. The margin is measured along the actual path of text drawing.

UISDC\$GET_VISIBILITY

Returns a Boolean value that indicates whether or not the specified rectangle in the display window is visible.

Format

Boolean=UISDC\$GET_VISIBILITY *wd_id* [*x*₁, *y*₁ [*x*₂, *y*₂]]

Returns

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Boolean value returned in a status variable or R0 (VAX MACRO). The returned value, the visibility status, is a Boolean TRUE only if the entire area is visible, and a Boolean FALSE if even a portion of the area is occluded or clipped.

UISDC\$GET_VISIBILITY signals all errors; no condition values are returned.

Arguments

wd_id
VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

*x*₁, *y*₁, *x*₂, *y*₂
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinates of a rectangle in the display window. The *x*₁ and *y*₁ arguments are addresses of integers that define the lower-left corner of a rectangle in the display window. The *x*₂ and *y*₂ arguments are

19-26 **UISDC Routines**
UISDC\$GET_VISIBILITY

addresses of integers that define the upper-right corner of a rectangle in the display window.

If the coordinates of the rectangle are not specified, the dimensions of the entire display window are used by default.

Description

UIS\$GET_VISIBILITY determines if a single position is visible by specifying the same coordinate for both minimum and maximum values.

UISDC\$IMAGE

Draws a raster image into a specified display rectangle.

Format

UISDC\$IMAGE *wd_id, atb, x₁, y₁, x₂, y₂, rasterwidth, rasterheight, bitsperpixel, rasteraddr*

Returns

UISDC\$IMAGE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block that modifies the writing mode.

x₁, y₁, x₂, y₂

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinates of the rectangle in the display window. The **x₁** and **y₁** arguments are the addresses of integers that define the lower-left corner of the rectangle in the display window. The **x₂** and **y₂** arguments

19-28 **UISDC Routines**
UISDC\$IMAGE

are the addresses of integer pixels that define the upper-right corner of the rectangle in the display window.

rasterwidth

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Width of the raster image. The **rasterwidth** argument is the address of a longword that defines the width of the raster image in pixels.

rasterheight

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Height of the raster image. The **rasterheight** is the address of a longword that defines the height of the raster image in pixels.

bitsperpixel

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of bits per pixel in the raster image. The **bitsperpixel** argument is the address of a longword that defines the number of bits per pixel in the raster image. The **bitsperpixel** argument is currently required to be either 1 or 8.

If **bitsperpixel** is specified as 8 on a single-plane system, the results are unpredictable.

rasteraddr

VMS Usage: **vector_longword_unsigned**
type: **longword_unsigned**
access: **read only**
mechanism: **by reference**

Raster image. The **rasteraddr** argument is the address of an array that defines a raster image.

Description

The raster dimensions are described by the width, height, and bits per pixel parameters. The width and height give the number of pixels in each dimension, and bits per pixel represents the number of bits that makes up each pixel. The raster is read from memory as "height" bit vectors each of which is "width" pixels long and each pixel is "bits/pixel" bits long.

UISDC\$IMAGE never scales. If the size of the destination rectangle is larger than the size of the raster, then the remaining space on the right and top will not be written.

The procedure for assignment of bits in the bitmap is as follows:

1. Each bit in the array is set from left-most bit to the right-most bit
2. Each row is filled from the top row to the bottom row.

NOTE: The bitmap is not byte- or word-aligned.

The following figure illustrates the setting of bits in the bitmap.

19-30 UISDC Routines
UISDC\$IMAGE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0	1	0	0	1	1	1	0	1

Bitmap
Image



1 →

2 ↓

1	0	1	1	1	0	0	1	0	1	0	1
1	0	1	0								

Raster
Image

UISDC\$LINE

Draws a line or series of unconnected lines.

Format

UISDC\$LINE *wd_id, atb, x₁, y₁ [,x₂,y₂ [,...x_n,y_n]]*

Returns

UISDC\$LINE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies line style and line width or both.

x, y

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Viewport-relative device coordinate pair. The **x** and **y** arguments are the addresses of integers that define a point in the display window.

If the arguments are repeated to specify a second position, a line is created.

19-32 **UISDC Routines**
UISDC\$LINE

If one coordinate pair is specified, a point is drawn. If any other odd number of coordinate pairs is specified, the final coordinate pair is ignored.

Up to 126 world coordinate pairs may be specified as arguments. See the "DESCRIPTION" section below for more information about this argument.

Description

If one position is specified, then a point is drawn. If two positions are specified, a single vector is drawn.

Up to 252 arguments can be specified, that is, 63 unconnected lines may be drawn. If a larger number of points must be specified in a single call, UIS\$LINE_ARRAY should be used.

The points or lines are drawn with the line pattern and width for the attribute block. Fill pattern attribute settings are ignored.

UISDC\$LINE_ARRAY

Draws an unfilled point, line, or a series of unconnected lines depending on the number of positions specified.

Format

UISDC\$LINE_ARRAY *wd_id, atb, count, x_vector, y_vector*

Returns

UISDC\$LINE_ARRAY signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword integer that identifies an attribute block that modifies line style or line width or both.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of points. The **count** argument is the address of longword integer that denotes the number of viewport-relative device coordinate pairs defined in the arguments **x_vector** and **y_vector**.

19-34 **UISDC Routines**
UISDC\$LINE_ARRAY

x_vector, y_vector

VMS Usage: **vector_longword_unsigned**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Array of *x* and *y* viewport-relative device coordinates. The **x_vector** and **y_vector** arguments are the addresses of arrays of integers whose elements together define in viewport-relative device coordinates the starting and end points of lines drawn in the display window.

Description

UISDC\$LINE_ARRAY performs the same functions as UISDC\$LINE except that *x* and *y* coordinates are stored in arrays.

A maximum of 32,767 points can be plotted in a single call. UISDC\$LINE_ARRAY is the same as UISDC\$LINE except that the *x* and *y* coordinates are specified using two arrays, each of length *count* points.

19-36 UISDC Routines
UISDC\$LOAD_BITMAP

bitmap_len

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Bitmap length. The **bitmap_len** argument is the address of the number that defines the length of the bitmap in bytes. The length must be a multiple of 2.

bitmap_width

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Width of the bitmap. The **bitmap_width** argument is the address of a number that defines the width of the bitmap in pixels. If the number of bits per pixel is 1, the specified width must be a multiple of 16.

If the width of the bitmap should not exceed 1024.

bits_per_pixel

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The **bits_per_pixel** argument is the address of a number that defines the number of bits per pixel. Currently, the values 1 and 8 are supported.

Description

See the *MicroVMS Workstation Video Device Driver Manual* for more information.

UISDC\$MEASURE_TEXT

Measures a text string as if it were output in a display window.

Format

UISDC\$MEASURE_TEXT *wd_id, atb, text_string, retwidth, retheight, [,ctllist, ctllen] [,posarray]*

Returns

UISDC\$MEASURE_TEXT signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block that modifies text output.

text_string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Text string. The **text_string** argument is the address of a character string descriptor of a text string.

19-38 **UISDC Routines**
UISDC\$MEASURE_TEXT

retwidth, retheight

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Dimensions of the text string. The **retwidth** and **retheight** arguments are the addresses of longwords that receive the width and height of the text in centimeters.

ctllist

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Text formatting list. The **ctllist** argument is the address of an array of longwords that describes the font, text rendition, format, and positioning of text string fragments. See **UIS\$TEXT** for a complete description of the formatting control list.

ctlten

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Length of the text formatting control list. The **ctlten** argument is the address of a longword that defines the length of the text formatting control list.

posarray

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Character position array. The **posarray** argument is the address of an array of longwords that receives character positions in pixels that are relative offsets at which each character would have been displayed. See **UIS\$MEASURE_TEXT** for a complete description of the character position array.

Description

UISDC\$MEASURE_TEXT is used in justification and text positioning applications. The routine returns the height and width of the text string in viewport-relative device coordinates.

UISDC\$MOVE_AREA

Shifts a portion of a display window to another position in the window.

Format

UISDC\$MOVE_AREA *wd_id, x₁, y₁, x₂, y₂, new_x, new_y*

Returns

UISDC\$MOVE_AREA signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

x₁, y₁, x₂, y₂

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinates of the source rectangle. The **x₁** and **y₁** arguments are the addresses of integers that define the lower-left corner of the source rectangle. The **x₂** and **y₂** are the addresses of integers that define the upper-right corner of the source rectangle.

new_x, new_y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinate pair. The **new_x** and **new_y** arguments are the addresses of integers that define the lower-left corner of the

19-40 **UISDC Routines**
 UISDC\$MOVE_AREA

destination rectangle. The height and width of the destination rectangle is implied from the height and width of the source rectangle.

Description

Note that display objects that are only partially contained within the specified source rectangle, though partially moved within existing display windows will be completely moved within the display list.

The nonoccluding portion of the source rectangle (if any) is erased after the operation.

UISDC\$NEW_TEXT_LINE

Moves the current text position along the actual path of text drawing to the starting margin and then along the margin in the direction of the minor text path. Depending on the minor text path, either the width or height of the character cell is used for spacing between characters and lines.

Format

UISDC\$NEW_TEXT_LINE *wd_id, atb*

Returns

UISDC\$NEW_TEXT_LINE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See **UIS\$CREATE_WINDOW** for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block.

UISDC\$PLOT

Draws a filled or unfilled point, line, or polygon depending on the number of positions specified.

Format

UISDC\$PLOT *wd_id, atb, x₁, y₁ [,x₂,y₂ [,...x_n,y_n]]*

Returns

UISDC\$PLOT signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block that modifies line style and line width.

x, y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinate pair. The **x** and **y** arguments are the addresses of integers that define a point in the display window. If the argument be used to specify a second position, a line is created. Up to 126

viewport-relative device coordinate pairs may be specified as arguments. See the DESCRIPTION section below for more information about this argument.

Description

If one position is specified, then a point is drawn. If two positions are specified, a single vector is drawn. If more than two positions are specified, a connected polygon is drawn. Up to 252 arguments can be specified, giving a maximum of a 126-point polygon using this routine. If a larger number of points must be specified in a single call, UISDC\$PLOT_ARRAY should be used.

The points or lines are drawn with the line pattern and width for the attribute block, and if FILL is enabled for the attribute block, the enclosed area is filled with the current fill pattern.

NOTE: VAX PASCAL application programs that draw lines and polygons should use UISDC\$PLOT_ARRAY.

UISDC\$PLOT_ARRAY

Draws an unfilled or filled point, line or polygon depending on the number of positions specified. This routine performs the same functions as UISDC\$PLOT.

Format

UISDC\$PLOT_ARRAY *wd_id, atb, count, x_vector, y_vector*

Returns

UISDC\$PLOT_ARRAY signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that identifies an attribute block that modifies line style or line width or both.

count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of points. The **count** argument is the address of longword that denotes the number of viewport-relative device coordinate pairs defined in the **x_vector** and **y_vector** arguments.

x_vector, y_vector

VMS Usage: **vector_longword_signed**

type: **longword_signed**

access: **read only**

mechanism: **by reference**

Array of x and y viewport-relative device coordinates. The **x_vector** argument is the address of an array of integers whose elements are the x viewport-relative device coordinate values of points defined in the window display. The **y_vector** argument is the address of an array of integers whose elements are the y viewport-relative device coordinate values of points defined in the display window.

Description

A maximum of 65,535 points can be plotted in a single call.

UISDC\$PLOT_ARRAY is the same as UISDC\$PLOT except that the x and y viewport-relative device coordinates are specified using two arrays, each of length n points.

UISDC\$QUEUE_DOP

Queues the specified drawing operation primitive (DOP) for execution in the specified window and then returns control to the application.

Format

UISDC\$QUEUE_DOP *wd_id ,dop*

Returns

UISDC\$QUEUE_DOP signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

dop

VMS Usage: **vector_byte_unsigned**
type: **byte_unsigned**
access: **read only**
mechanism: **by reference**

Drawing operation primitive. The **dop** argument is the address of an array of bytes that contains the drawing operation primitive.

Description

UISDC\$EXECUTE_DOP_ASYNCH queues the specified drawing operation primitive (DOP) for execution in the specified window. To obtain notification that the DOP has completed execution, see UISDC\$EXECUTE_DOP_ASYNCH and UISDC\$EXECUTE_DOP_SYNC. See the *MicroVMS Workstation Video Device Driver Manual* for more information about DOPs.

UISDC\$READ_IMAGE

Reads a raster image from within a specified rectangle contained by a display window.

Format

UISDC\$READ_IMAGE *wd_id, x₁, y₁, x₂, y₂, rasterwidth, rasterheight, bitsperpixel, [rasteraddr], rasterlen*

Returns

UISDC\$READ_IMAGE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **object_id**
 type: **longword**
 access: **read only**
 mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

x₁, y₁

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Viewport-relative device coordinates of lower-left corner of the specified rectangle. The **x₁,y₁** arguments are the addresses of integers that define the lower-left corner of the rectangle in the display window.

x₂, y₂

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

19-48 **UISDC Routines**
UISDC\$READ_IMAGE

Viewport-relative device coordinates of the upper-right corner of the rectangle. The x_2, y_2 arguments are the addresses of integers that define the upper-right corner of the specified rectangle in the display window.

rasterwidth

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Width of the raster image in pixels. The **rasterwidth** argument is the address of a longword that receives the width of the raster image in pixels.

rasterheight

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Height of the raster image in pixels. The **rasterheight** argument is the address of a longword that receives the height of the raster image in pixels.

bitsperpixel

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Number of bits per pixel in the raster image. The **bitsperpixel** argument is the address of a longword that receives the number of bits per pixel in the raster image.

rasteraddr

VMS Usage: **vector_byte_unsigned**
type: **byte**
access: **write only**
mechanism: **by reference**

Address of buffer in which to return the raster image. The **rasteraddr** argument is the address of an array of bytes that receives the raster image.

rasterlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Size in bytes of the buffer. The **rasterlen** argument is the address of a longword that specifies the size in bytes of the buffer.

Description

The raster image contained within the rectangle described by x_1 , y_1 and x_2 , y_2 is returned in the specified buffer. The actual dimensions, in pixels, of the returned buffer is written to **rasterwidth** and **rasterheight**. The number of bits per pixel is written to **bitsperpixel**. If the size of the buffer specified by **rasterlen** is not large enough to accept the entire bitmap raster, then **rasterwidth**, **rasterheight**, and **bitsperpixel** are returned as 0 and no data is written to the buffer.

If the buffer length is specified as 0, values are returned in **rasterwidth**, **rasterheight**, and **bitsperpixel**. These values can be used to calculate the size of the buffer needed to contain the raster image.

You should specify a buffer length of 0 to obtain the width, height, and bits per pixels. Use these returned values to do the following:

1. Calculate the correct buffer size
2. Reissue the call with the correct data

UISDC\$SET_ALIGNED_POSITION

Sets the current position for text output. This routine differs from `UISDC$SET_POSITION` in that the position refers to the upper-left corner of the next character to the output.

Format

UISDC\$SET_ALIGNED_POSITION *wd_id, atb, x, y*

Returns

`UISDC$SET_ALIGNED_POSITION` signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies a display window. See `UIS$CREATE_WINDOW` for more information about the ***wd_id*** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The ***atb*** is the address of a longword that identifies an attribute block that contains the appropriate font attribute text attribute setting.

x, y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinate pair. The *x* and *y* arguments are the addresses of integers that define the current position for text output.

Description

UISDC\$SET_ALIGNED_POSITION is useful in applications that know the position of the upper-left corner, but also don't know enough about the font baseline to determine the proper alignment point. The position is converted into the proper alignment point using the font specified in the given attribute block. The alignment point is stored internally.

UISDC\$SET_BUTTON_AST

Allows an application to find out when a button on the pointing device is depressed or released in a given rectangle of the display window.

Format

UISDC\$SET_BUTTON_AST *wd_id* [*astadr*, [*astprm*], *keybuf*]
[*x*₁, *y*₁, *x*₂, *y*₂]

Returns

UISDC\$SET_BUTTON_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The *astadr* argument is the address of an entry mask to a procedure that is called at AST level whenever a pointer button is depressed or released. To cancel the AST-enabling request of UISDC\$SET_BUTTON_AST, specify 0 in the *astadr* argument. To disable UISDC\$SET_BUTTON_AST, omit the *astadr* argument.

astprm

VMS Usage: **user_arg**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as a record or an array, to be passed to the AST routine. Calls to **UISDC\$SET_BUTTON_AST** in FORTRAN application programs should be coded as follows: **%REF(%LOC(astprm))**.

keybuf

VMS Usage: **address**
 type: **longword (unsigned)**
 access: **write only**
 mechanism: **by reference**

Key buffer. The **keybuf** argument is the address of a longword buffer that receives button information whenever a pointer button is depressed or released. The low two bytes are the key code. The buttons are located on the left, center and right of the pointing device and are defined as **UIS\$_POINTER_BUTTON_1**, **UIS\$_POINTER_BUTTON_2**, **UIS\$_POINTER_BUTTON_3**, and **UIS\$_POINTER_BUTTON_4** respectively. The bit **<31>** is set to **1** if the button has been pressed, and **0** if the button has been released. The buffer is not overwritten with subsequent button transitions until the AST routine completes.

x₁, y₁, x₂, y₂

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Viewport-relative device coordinates of a rectangle in the display window. The **x₁** and **y₁** arguments are the addresses of integers that define the lower-left corner of a rectangle in the display window. The **x₂** and **y₂** arguments are the addresses of integer pixels that define the upper-right corner of a rectangle in the display window.

If no rectangle is specified, the entire display window is assumed.

Description

This function can be called any number of times for different rectangles within the same display window or many display windows.

See the **DESCRIPTION** section of **UIS\$SET_BUTTON_AST** for information about pointer region priorities.

UISDC\$SET_CHAR_SIZE

Sets the viewport-relative device coordinate size of the specified character.

Format

UISDC\$SET_CHAR_SIZE *wd_id, iatb, oatb, [char],
[width][,height]*

Returns

UISDC\$SET_CHAR_SIZE signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies the display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The **iatb** argument is the address of a longword that identifies an attribute block to be modified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The **oatb** argument is the address of a longword that identifies a modified attribute block.

char

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Single character. The **char** argument is the address of a descriptor of a single character. You may specify any character in the font. Specify this argument when you are using proportionally spaced fonts to establish spacing and scaling factors among character within the font. The **char** has no effect on monospaced fonts.

If **char** is not specified or if the specified character is invalid, the widest character in the font is chosen.

width

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Character width. The **width** argument is the address of an integer that defines the character width in viewport-relative device coordinates.

height

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Character height. The **height** argument is the address of an integer that defines the character height in viewport-relative device coordinates.

Description

To disable character scaling, omit all of the following arguments: **char**, **width**, and **height**.

To scale characters to their nominal size as specified in the font, do not specify **width** or **height**. Scaling is only visible when you use a window that does not have same proportions as the virtual display.

If you specify either **width** or **height** only, characters are scaled to the size you specify and in the direction you specify. In the unspecified direction, characters are scaled so as to maintain the same ratio of height and width as the unscaled character.

UISDC\$SET_CLIP

Sets a clipping rectangle within the display window.

Format

UISDC\$SET_CLIP *wd_id, iatb, oatb* [*x1, y1, x2, y2*]

Returns

UISDC\$SET_CLIP signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the *wd_id* argument.

iatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Input attribute block number. The *iatb* argument is the address of a longword value that identifies an attribute block to be modified. Either the attribute block 0 or a previously modified attribute block can be specified.

oatb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Output attribute block number. The *oatb* argument is the address of a longword value that identifies a newly modified attribute block that controls the dimensions of the clipping rectangle.

x_1, y_1, x_2, y_2

VMS Usage: **longword_unsigned**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Viewport-relative device coordinates of the clipping rectangle. The x_1 and y_1 arguments are the addresses of integers that define the lower-left corner of the clipping rectangle in viewport-relative device coordinates. The x_2 and y_2 arguments are the addresses of integers that define the upper-right corner of the clipping rectangle in viewport-relative device coordinates. Only graphic objects and portions of graphic objects drawn **within** the clipping rectangle are seen.

If the device coordinates of the clipping rectangle corners are not specified, then clipping is disabled for this attribute block.

UISDC\$SET_POINTER_AST

Allows an application to find out when the pointer is moved in a given rectangle of the display window.

Format

UISDC\$SET_POINTER_AST *wd_id* [*,astadr* [*,astprm*]]
[,x₁, y₁, x₂, y₂] [*,exitastadr*
[,exitastprm]]

Returns

UISDC\$SET_POINTER_AST signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

AST routine. The **astadr** argument is the address of the entry mask to a procedure that is called at AST level whenever the pointer is moved within a rectangle in the display window.

To cancel the AST-enabling request of UISDC\$SET_POINTER_AST for this argument only, specify 0 in the **astadr** argument and the coordinates of the rectangle.

astprm

VMS Usage: **user_arg**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

AST parameter. The **astprm** argument is the address of a single argument or data structure, such as an array or record, passed to the AST routine. Calls to **UISDC\$SET_POINTER_AST** in VAX FORTRAN application programs should be coded as follows: **%REF(%LOC(astprm))**.

x₁, y₁, x₂, y₂

VMS Usage: **longword_unsigned**
 type: **longword (unsigned)**
 access: **read only**
 mechanism: **by reference**

Viewport-relative device coordinates of the rectangle of the display window. The **x₁** and **y₁** arguments are the addresses of integers that define the lower-left corner of the rectangle of the display window. The **x₂** and **y₂** arguments are the addresses of integer pixels that define the upper-right corner of the rectangle of the display window.

If no rectangle is specified, the entire display window is assumed.

To cancel an AST-enabling request, specify **0** in either the **astadr** or the **exitastadr** arguments or both and the coordinates of the rectangle.

exitastadr

VMS Usage: **ast_procedure**
 type: **procedure entry mask**
 access: **read only**
 mechanism: **by reference**

Exit AST routine. The **exitastadr** argument is the address of the entry mask to a procedure that is called at AST level whenever the pointer leaves the rectangle.

To cancel the AST-enabling request of **UISDC\$SET_POINTER_AST** for the EXIT AST routine only, specify **0** in the **exitastadr** argument and the coordinates of the rectangle.

exitastprm

VMS Usage: **user_arg**
 type: **longword**
 access: **read only**
 mechanism: **by reference**

19-60 UISDC Routines

UISDC\$SET_POINTER_AST

Exit AST parameter. The **exitastprm** argument is the address of a single argument or data structure, such as an array or record, to be passed to the AST routine. Calls to `UISDC$SET_POINTER_AST` in VAX FORTRAN application programs should be coded as follows: `%REF(%LOC(exitastprm))`.

Description

`UISDC$SET_POINTER_AST` also allows an application to keep track of the pointer in its own way. This routine can be called any number of times for different rectangles.

Note that an application need not enable both AST routines. It may specify one or the other.

`UISDC$SET_POINTER_AST` can be used by the application to highlight the display or some other application-specific function, as the user moves the pointer over specific areas of the display window. This might be used to define a number of regions within a menu, and execute an AST when the pointer enters or leaves any of these regions.

If both AST routines are enabled and the value `0` is specified in the **astadr** argument, the first AST routine is canceled.

See the DESCRIPTION section of `UIS$SET_BUTTON_AST` for information about pointer region priorities.

UISDC\$SET_POINTER_PATTERN

Allows an application to specify a special pointer cursor pattern.

Format

UISDC\$SET_POINTER_PATTERN *wd_id* [*pattern_array*,
pattern_count, *activex*,
activey] [*x1*, *y1*, *x2*, *y2*]
[flags]

Returns

UISDC\$SET_POINTER_PATTERN signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

pattern_array

VMS Usage: **vector_word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

16 x 16 bit cursor pattern. The **pattern_array** argument is the address of one or more arrays of 16 words that represents a bitmap image of the cursor.

Color and intensity applications can define two patterns that are also executable on monochrome systems.

19-62 **UISDC Routines**
UISDC\$SET_POINTER_PATTERN

If two arrays are specified in an application running on a single-plane system, the first array is used.

NOTE: The bitmap image of the new pointer pattern is mapped in reverse order to display screen.

pattern_count

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Number of 16- x 16-bit cursor pattern. The **pattern_count** argument is the address of a longword that contains the number of cursor pattern arrays defined in the **pattern_array** argument.

activex, activey

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The **activex** and **activey** arguments are used to specify the actual bit in the cursor pattern that should be used to calculate the current pointer position. The arguments are expressed as bit offsets from the lower-left corner of the cursor pattern.

x₁, y₁, x₂, y₂

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinates of the rectangle in the display window. The **x₁** and **y₁** arguments are the addresses of integers that define the lower-left corner of the rectangle in the display window. The **x₂** and **y₂** arguments are the addresses of integer pixels that define the upper-right corner of the rectangle in the display window.

flags

VMS Usage: **longword_mask**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Flags. The **flags** argument is the address of a longword mask whose bits determine whether or not the cursor is confined to the display window rectangle.

When specified, UISM_BIND_POINTER sets the appropriate bit in the mask.

Description

UISDC\$SET_POINTER_PATTERN allows an application to specify a special pointer pattern to be used when the pointer is within the display window region specified by the optional rectangle. If no rectangle is given, then the entire display window is assumed. This function can be called any number of times for different rectangles.

To disable UISDC\$SET_POINTER_PATTERN, omit the **pattern_array**, **pattern_count**, **activex**, and **activey** arguments.

See the DESCRIPTION section of UIS\$SET_BUTTON_AST for information about pointer region priorities.

UISDC\$SET_POINTER_POSITION

Specifies a new current pointer position in device coordinates. It is only effective if the new pointer position is within the specified display window and visible.

Format

status=UISDC\$SET_POINTER_POSITION *wd_id*, *x*, *y*

Returns

Longword value returned as boolean in the variable *status* or R0 (VAX MACRO) to indicate that the position is set.

UISDC\$SET_POINTER_POSITION signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The *wd_id* argument is the address of a longword that uniquely identifies a display window. See UISDC\$CREATE_WINDOW for more information about the *wd_id* argument.

x, *y*

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinates of the new pointer position. The *x* and *y* arguments are the addresses of integers that define the new pointer position.

UISDC\$SET_POSITION

Sets the current position for text output. The current position is the point of alignment on the baseline of the next character to be output.

Format

UISDC\$SET_POSITION *wd_id, x, y*

Returns

UISDC\$SET_POSITION signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The ***wd_id*** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the ***wd_id*** argument.

x, y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinate pair. The ***x*** and ***y*** arguments are the addresses of integers that define the current position for text output.

x,y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Starting margin position. The *x,y* arguments are the addresses of integers that define a point on the starting margin in viewport-relative device coordinates. The starting margin is the minor text path when the angle of text slope equals 0 degrees.

margin_length

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Ending margin position. The **margin_length** is the address of a number that defines the distance from the starting margin to the end margin in viewport-relative device coordinates.

UISDC\$TEXT

Draws a series of encoded characters.

Format

UISDC\$TEXT *wd_id, atb, text_string [,x,y] [,ctllist ,ctllen]*

Returns

UISDC\$TEXT signals all errors; no condition values are returned.

Arguments

wd_id

VMS Usage: **identifier**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display window identifier. The **wd_id** argument is the address of a longword that uniquely identifies a display window. See UIS\$CREATE_WINDOW for more information about the **wd_id** argument.

atb

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute block number. The **atb** argument is the address of a longword that specifies an attribute that modifies text output.

text_string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Text string. The **text_string** argument is the address of a character string descriptor of a text string.

x, y

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Viewport-relative device coordinates pair. The *x* and *y* arguments are the addresses of integers that define the viewport-relative device coordinates of the starting point of text output at the upper-left corner of the character cell.

If this argument is not specified, the current text position is used. (See the `UISDC$SET_ALIGNED_POSITION` routine for more information.)

ctl

VMS Usage: **vector_longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Text control formatting list. The *ctl* argument is the address of an array of longwords that describe the font, text rendition, format, and positioning of text string fragments. See `UIS$TEXT` for a complete description of the text formatting control list.

ctl

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Length of the text formatting control list. The *ctl* argument is the address of an integer that defines the length of the text formatting control list in longwords.

Description

Nonprinting characters such as tab and linefeed are not handled in any special way. The character is obtained from the font and is displayed like any other character.

Appendix A

Summary of UIS Calling Sequences

A.1 UIS Calling Sequences

Table A-1 lists return values, entry point names, and parameter lists of all UIS routines.

Table A-1 Summary of UIS Calling Sequences

Return Value	Routine	Argument List
seg_id	UIS\$BEGIN_SEGMENT UIS\$CIRCLE	vd_id vd_id, atb, center_x, center_y, xradius, [,start_deg] [,end_deg]
copy_id	UIS\$CLOSE_WINDOW UIS\$COPY_OBJECT ¹	wd_id { obj_id } [,matrix] [,atb] { seg_id }
vcm_id	UIS\$CREATE_COLOR_MAP	vcm_size [,vcm_name] [,vcm_attributes]
cms_id	UIS\$CREATE_COLOR_MAP_SEG	vcm_id, [,devnam] [,place_mode] [,place_data]
vd_id	UIS\$CREATE_DISPLAY	x1, y1, x2, y2, width, height [,vcm_id]
kb_id	UIS\$CREATE_KB UIS\$CREATE_TERMINAL	devnam termtype [,title] [,attributes] [,devnam] [,devlen]
tb_id	UIS\$CREATE_TB	devname
tr_id	UIS\$CREATE_TRANSFORMATION	vd_id, x1, y1, x2, y2 [vdx1, vdy1, vdx2, vdy2]

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

A-2 Summary of UIS Calling Sequences

UIS Calling Sequences

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
wd_id	UIS\$CREATE_WINDOW	vd_id, devnam [,title] [,x ₁ , y ₁ , x ₂ , y ₂] [, width,height] [,attributes]
	UIS\$DELETE_COLOR_MAP	vcm_id
	UIS\$DELETE_COLOR_MAP_SEG	cms_id
	UIS\$DELETE_DISPLAY	vd_id
	UIS\$DELETE_KB	kb_id
	UIS\$DELETE_OBJECT ¹	{ obj_id } { seg_id }
	UIS\$DELETE_PRIVATE	obj_id
	UIS\$DELETE_TB	tb_id
	UIS\$DELETE_TRANSFORMATION	tr_id
	UIS\$DELETE_WINDOW	wd_id
	UIS\$DISABLE_DISPLAY_LIST	vd_id [,display_flags]
	UIS\$DISABLE_KB	kb_id
	UIS\$DISABLE_TB	tb_id
	UIS\$DISABLE_VIEWPORT_KB	wd_id
	UIS\$ELLIPSE	vd_id, atb, center_x, center_y, xradius, yradius, [,start_deg] [,end_deg]
	UIS\$ENABLE_DISPLAY_LIST	vd_id [,display_flags]
	UIS\$ENABLE_KB	kb_id [,wd_id]
	UIS\$ENABLE_TB	tb_id
	UIS\$ENABLE_VIEWPORT_KB	kb_id, wd_id
	UIS\$END_SEGMENT	vd_id
	UIS\$ERASE	vd_id [,x ₁ y ₁ , x ₂ , y ₂]
	UIS\$EXECUTE	vd_id [,buflen] [,bufaddr]
	vd_id	UIS\$EXECUTE_DISPLAY
UIS\$EXPAND_ICON		wd_id [,icon_wd_id] [,attributes]
UIS\$EXTRACT_HEADER		vd_id [,buflen, bufaddr] [,retlen]
UIS\$EXTRACT_OBJECT ¹		{ obj_id } { seg_id } [,buflen ,bufaddr] [,retlen]

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
	UIS\$EXTRACT_PRIVATE ¹	{ obj_id } { seg_id } [,buflen ,bufaddr] [,retlen]
	UIS\$EXTRACT_REGION	vd_id [,x1,y1,x2,y2] [,buflen ,bufaddr] [,retlen]
	UIS\$EXTRACT_TRAILER	vd_id [,buflen, bufaddr] [,retlen]
obj_id	UIS\$FIND_PRIMITIVE	vd_id, x1, y1,x2,y2 [,context] [,extent]
seg_id	UIS\$FIND_SEGMENT	vd_id, x1, y1, x2, y2, [,context] [,extent]
	UIS\$GET_ABS_POINTER_POS	devnam, retx, rety
	UIS\$GET_ALIGNED_POSITION	vd_id, atb, retx, rety
arc_type	UIS\$GET_ARC_TYPE	vd_id,*atb
index	UIS\$GET_BACKGROUND_INDEX	vd_id, atb
status	UIS\$GET_BUTTONS	wd_id, retstate
angle	UIS\$GET_CHAR_ROTATION	vd_id, atb
boolean	UIS\$GET_CHAR_SIZE	vd_id, atb [,char], [width, height]
angle	UIS\$GET_CHAR_SLANT	vd_id, atb
	UIS\$GET_CHAR_SPACING	vd_id, atb, dx, dy
status	UIS\$GET_CLIP	vd_id, atb [,x1, y1, x2, y2]
	UIS\$GET_COLOR	vd_id, index, retr, retg, retb [,wd_id]
	UIS\$GET_COLORS	vd_id, index, count, retr_vector, retg_vector, retb_vector [,wd_id]
current_id	UIS\$GET_CURRENT_OBJECT	vd_id
	UIS\$GET_DISPLAY_SIZE	devnam, retwidth, retheight [,retresolx, retresoly] [,retpwidth retpheight]
status	UIS\$GET_FILL_PATTERN	vd_id, atb [,index]
	UIS\$GET_FONT	vd_id, atb, bufferdesc [,length]
	UIS\$GET_FONT_ATTRIBUTES	vd_id, ascender, descender, height, [,maximum_width] [,item_list]
	UIS\$GET_FONT_SIZE	fontid, text_string, retwidth, retheight

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

A-4 Summary of UIS Calling Sequences

UIS Calling Sequences

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
	UIS\$GET_HW_COLOR_INFO	devnam [,type] [,indices] [,colors] [,maps] [,rbits] [,gbits] [,bbits] [,ibits] [,res_indices] [,regen]
	UIS\$GET_INTENSITIES	vd_id, index, count, reti_vector [,wd_id]
	UIS\$GET_INTENSITY	vd_id, index, reti [,wd_id]
	UIS\$GET_KB_ATTRIBUTES	kb_id [,enable_items] [,disable_items] [,click_volume]
style	UIS\$GET_LINE_STYLE	vd_id, atb
width	UIS\$GET_LINE_WIDTH	vd_id, atb [,mode]
next_id	UIS\$GET_NEXT_OBJECT ¹	{ obj_id } [,flags] { seg_id }
type	UIS\$GET_OBJECT_ATTRIBUTES ¹	{ obj_id } [,extent] { seg_id }
parent_id	UIS\$GET_PARENT_SEGMENT ¹	{ obj_id } { seg_id }
status	UIS\$GET_POINTER_POSITION	vd_id, wd_id, retx, rety
	UIS\$GET_POSITION	vd_id, retx, rety
prev_id	UIS\$GET_PREVIOUS_OBJECT ¹	{ obj_id } [,flags] { seg_id }
root_id	UIS\$GET_ROOT_SEGMENT	vd_id
	UIS\$GET_TB_INFO	devnam, retwidth, retheight, retresolx, retresoly [,retpwidth, retpheight]
	UIS\$GET_TB_POSITION	wd_id, retx, rety
formatting	UIS\$GET_TEXT_FORMATTING	vd_id, atb
	UIS\$GET_TEXT_MARGINS	vd_id, atb, x, y [,margin_length]
	UIS\$GET_TEXT_PATH	vd_id, atb [,major][,minor]
angle	UIS\$GET_TEXT_SLOPE	vd_id, atb
vcm_id	UIS\$GET_VCM_ID	vd_id
boolean	UIS\$GET_VIEWPORT_ICON	wd_id [icon_wd_id]
	UIS\$GET_VIEWPORT_POSITION	wd_id, retx, rety
	UIS\$GET_VIEWPORT_SIZE	wd_id, retwidth, retheight
status	UIS\$GET_VISIBILITY	vd_id, wd_id [,x ₁ , y ₁ [,x ₂ , y ₂]]

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

Summary of UIS Calling Sequences A-5

UIS Calling Sequences

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
attributes	UIS\$GET_WINDOW_ATTRIBUTES	wd_id
	UIS\$GET_WINDOW_SIZE	vd_id, wd_id, x ₁ , y ₁ , x ₂ , y ₂
index	UIS\$GET_WRITING_INDEX	vd_id, atb
mode	UIS\$GET_WRITING_MODE	vd_id, atb
	UIS\$GET_WS_COLOR	vd_id, color_id, retr, retg, retb [,wd_id]
	UIS\$GET_WS_INTENSITY	vd_id, color_id, reti [,wd_id]
	UIS\$HLS_TO_RGB	H, L, S, retr, retg, retb
	UIS\$HSV_TO_RGB	H, S, V, retr, retg, retb
	UIS\$IMAGE	vd_id, atb, x ₁ , y ₁ , x ₂ , y ₂ , rasterwidth, rasterheight, bitsperpixel, rasteraddr
	UIS\$INSERT_OBJECT ¹	{ obj_id } { seg_id }
	UIS\$LINE	vd_id, atb, x ₁ , y ₁ [,x ₂ , y ₂ [...x _n , y _n]]
	UIS\$LINE_ARRAY	vd_id, atb, count, x_vector, y_vector
	UIS\$MEASURE_TEXT	vd_id, atb, text_string, retwidth, retheight [,ctllist, ctllen] [,posarray]
	UIS\$MOVE_AREA	vd_id, x ₁ , y ₁ , x ₂ , y ₂ , new_x, new_y
	UIS\$MOVE_VIEWPORT	wd_id, attributes
	UIS\$MOVE_WINDOW	vd_id, wd_id, x ₁ , y ₁ , x ₂ , y ₂
	UIS\$NEW_TEXT_LINE	vd_id, atb
UIS\$PLOT	vd_id, atb, x ₁ , y ₁ [,x ₂ , y ₂ [...x _n , y _n]]	
UIS\$PLOT_ARRAY	vd_id, atb, count, x_vector, y_vector	
	UIS\$POP_VIEWPORT	wd_id
status	UIS\$PRESENT	[major_version], [minor_version]
	UIS\$PRIVATE ¹	{ obj_id } { vd_id }, facnum, buffer
	UIS\$PUSH_VIEWPORT	wd_id
keybuf	UIS\$READ_CHAR	kb_id [,flags]

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

A-6 Summary of UIS Calling Sequences

UIS Calling Sequences

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
	UIS\$RESIZE_WINDOW	vd_id, wd_id [,new_abs_x, new_abs_y] [,new_width new_height] [,new_wc_x1, new_wc_y1, new_wc_x2, new_wc_y2]
	UIS\$RESTORE_CMS_COLORS	cms_id
	UIS\$RGB_TO_HLS	R, G, B, reth, retl, rets
	UIS\$RGB_TO_HSV	R, G, B, reth, rets, retv
	UIS\$SET_ADDOPT_AST	vd_id [,astadr [,astprm]]
	UIS\$SET_ALIGNED_POSITION	vd_id, atb, x, y
	UIS\$SET_ARC_TYPE	vd_id, iatb, oatb, arc_type
	UIS\$SET_BACKGROUND_INDEX	vd_id, iatb, oatb, index
	UIS\$SET_BUTTON_AST	vd_id, wd_id [,astadr [,astprm] ,keybuf] [,x1, y1, x2, y2]
	UIS\$SET_CHAR_ROTATION	vd_id, iatb, oatb, angle
	UIS\$SET_CHAR_SIZE	vd_id, iatb, oatb [,char] [,width][,height]
	UIS\$SET_CHAR_SLANT	vd_id, iatb, oatb, angle
	UIS\$SET_CHAR_SPACING	vd_id, iatb, oatb, dx, dy
	UIS\$SET_CLIP	vd_id, iatb, oatb [,x1, y1, x2, y2]
	UIS\$SET_CLOSE_AST	wd_id [,astadr [,astprm]]
	UIS\$SET_COLOR	vd_id, index, R, G, B
	UIS\$SET_COLORS	vd_id, index, count, r_vector, g_vector, b_vector
	UIS\$SET_EXPAND_ICON_AST	wd_id [,astadr [,astprm]]
	UIS\$SET_FILL_PATTERN	vd_id, iatb, oatb [,index]
	UIS\$SET_FONT	vd_id, iatb, oatb, font_id
	UIS\$SET_GAIN_KB_AST	kb_id [,astadr [,astprm]]
	UIS\$SET_INSERTION_POSITION ¹	$\left\{ \begin{array}{l} \text{obj_id} \\ \text{seg_id} \\ \text{vd_id} \end{array} \right\} [\text{flags}]$
	UIS\$SET_INTENSITIES	vd_id, index, count, i_vector
	UIS\$SET_INTENSITY	vd_id, index, I
	UIS\$SET_KB_AST	kb_id [,astadr [,astprm], keybuf]

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

Summary of UIS Calling Sequences A-7

UIS Calling Sequences

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
	UIS\$SET_KB_ATTRIBUTES	kb_id [,enable_items] [.disable_items] [click_volume]
	UIS\$SET_KB_COMPOSE2	kb_id [,table, tablelen]
	UIS\$SET_KB_COMPOSE3	kb_id [,table, tablelen]
	UIS\$SET_KB_KEYTABLE	kb_id [,table, tablelen]
	UIS\$SET_LINE_STYLE	vd_id, iatb, oatb, style
	UIS\$SET_LINE_WIDTH	vd_id, itab, oatb, width [,mode]
	UIS\$SET_LOSE_KB_AST	kb_id [,astadr [,astprm]]
	UIS\$SET_MOVE_INFO_AST	wd_id [,astadr [,astprm]]
	UIS\$SET_POINTER_AST	vd_id, wd_id [,astadr [,astprm]] [,x1, y1, x2, y2] [exitastadr [,exitastprm]]
	UIS\$SET_POINTER_PATTERN	vd_id, wd_id [,pattern_array, pattern_count, activex, activey] [,x1, y1, x2, y2] [,flags]
status	UIS\$SET_POINTER_POSITION	vd_id, wd_id, x, y
	UIS\$SET_POSITION	vd_id, x, y
	UIS\$SET_RESIZE_AST	vd_id, wd_id [,astadr [,astprm]] [.new_abs_x, new_abs_y] [.new_ width, new_height] [.new_wc_x1, new_wc_y1, new_wc_x2, new_wc_y2]
	UIS\$SET_SHRINK_TO_ICON_AST	wd_id [,astadr [,astprm]]
	UIS\$SET_TB_AST	tb_id, [,data_astadr, [data_astprm]], [.x_pos,y_pos] [,data_x1, data_y1, data_x2, data_y2] [,button_astadr [,button_astprm],button_keybuf]
	UIS\$SET_TEXT_FORMATTING	vd_id, iatb, oatb,mode
	UIS\$SET_TEXT_MARGINS	vd_id, iatb, oatb, x, y, margin_length
	UIS\$SET_TEXT_PATH	vd_id, iatb, oatb, major[,minor]
	UIS\$SET_TEXT_SLOPE	vd_id, iatb, oatb, angle
	UIS\$SET_WRITING_INDEX	vd_id, iatb, oatb, index
	UIS\$SET_WRITING_MODE	vd_id, iatb, oatb, mode

A-8 Summary of UIS Calling Sequences

UIS Calling Sequences

Table A-1 (Cont.) Summary of UIS Calling Sequences

Return Value	Routine	Argument List
	UIS\$SHRINK_TO_ICON	wd_id [,icon_wd_id] [,icon_flags] [,icon_name] [,attributes]
	UIS\$SOUND_BELL	devnam [,bell_volume]
	UIS\$SOUND_CLICK	devnam [,click_volume]
status	UIS\$TEST_KB	kb_id
	UIS\$TEXT	vd_id, atb, text_string [,x, y], [ctlist, ctllen]
	UIS\$TRANSFORM_OBJECT ¹	{ obj_id } [,matrix] [,atb] { seg_id }

¹VAX PASCAL and VAX PL/I applications must specify the **obj_id** argument.

Appendix B

Summary of UISDC Calling Sequences

B.1 UISDC Calling Sequences

The following table summarizes UISDC calling sequences.

Table B-1 Summary of UISDC Calling Sequences

Return Value	Routine	Argument List
dop	UISDC\$ALLOCATE_DOP	wd_id, size, atb
	UISDC\$CIRCLE	wd_id, atb, center_x, center_y, xradius [,start_deg] [,end_deg]
	UISDC\$ELLIPSE	wd_id, atb, center_x, center_y, xradius, yradius, [,start_deg] [,end_deg]
	UISDC\$ERASE	wd_id [,x1,y1,x2, y2]
	UISDC\$EXECUTE_DOP_ASYNCH	wd_id, dop, iosb
	UISDC\$EXECUTE_DOP_SYNCH	wd_id, dop
	UISDC\$GET_ALIGNED_POSITION	wd_id, atb, retx, rety
boolean	UISDC\$GET_CHAR_SIZE	wd_id, atb [,char],[width][,height]
status	UISDC\$GET_CLIP	wd_id, atb [,x1,y1, x2,y2]
status	UISDC\$GET_POINTER_POSITION	wd_id, retx, rety
	UISDC\$GET_POSITION	wd_id, retx, rety
	UISDC\$GET_TEXT_MARGINS	wd_id, atb, x, y [,margin_length]
status	UISDC\$GET_VISIBILITY	wd_id [,x1,y1 [,x2,y2]]
	UISDC\$IMAGE	wd_id, atb, x1, y1, x2, y2, rasterwidth, rasterheight, bitsperpixel, rasteraddr
	UISDC\$LINE	wd_id, atb, x1,y1, [,x2,y2 [,...xn, yn]]

B-2 Summary of UISDC Calling Sequences

UISDC Calling Sequences

Table B-1 (Cont.) Summary of UISDC Calling Sequences

Return Value	Routine	Argument List
	UISDC\$LINE_ARRAY	wd_id, atb, count, x_vector, y_vector
bitmap_id	UISDC\$LOAD_BITMAP	wd_id, bitmap_adr, bitmap_len, bitmap_width, bits_per_pixel
	UISDC\$MEASURE_TEXT	wd_id, atb, text_string, retwidth, retheight [,ctllist ,ctllen] [,posarray]
	UISDC\$MOVE_AREA	wd_id, x1,y1,x2, y2, new_x, new_y
	UISDC\$NEW_TEXT_LINE	wd_id, atb
	UISDC\$PLOT	wd_id, atb, x1,y1, [,x2,y2 [,...xn, yn]]
	UISDC\$PLOT_ARRAY	wd_id, atb, count, x_vector, y_vector
	UISDC\$QUEUE_DOP	wd_id, dop
	UISDC\$READ_IMAGE	wd_id, x1, y1, x2, y2, rasterwidth, rasterheight, bitsperpixel, rasteraddr, rasterlen
	UISDC\$SET_ALIGNED_POSITION	wd_id, atb, x, y
	UISDC\$SET_BUTTON_AST	wd_id [,astadr, [astprm], keybuf] [,x1, y1, x2, y2]
	UISDC\$SET_CHAR_SIZE	wd_id, iatb, oatb [,char][,width][,height]
	UISDC\$SET_CLIP	wd_id, iatb, oatb [,x1, y1, x2, y2]
	UISDC\$SET_POINTER_AST	wd_id [,astadr [astprm]] [,x1, y1, x2, y2] [,exitastadr [,exitastprm]]
	UISDC\$SET_POINTER_PATTERN	wd_id [,pattern_array, pattern_count, activex, activey] [,x1, y1, x2, y2][,flags]
status	UISDC\$SET_POINTER_POSITION	wd_id, x, y
	UISDC\$SET_POSITION	wd_id, x, y
	UISDC\$SET_TEXT_MARGINS	wd_id, iatb, oatb, x, y, margin_length
	UISDC\$TEXT	wd_id atb, text_string [,x, y] [,ctllist, ctllen]

Appendix C

UIS Fonts

C.1 Overview

This appendix contains figures and tables illustrating the UIS multinational character and technical fonts and font names contained in the directory SYS\$FONT.

C.2 UIS Multinational Character Set Fonts

There are 14 multinational character set font files in the directory SYS\$FONT. The figure captions below identify each UIS font with an arbitrarily assigned font number. For more information about font characteristics, match this number with the appropriate table in Section C.2.1.

Figure C-1 Font 1

**ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890-=!@#%&^*()_+
<>,./?'";:\|[]{}**

ZK-4565-85

Figure C-2 Font 2

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890-=!@#%&^*()_+
<>,./?;:'"\|[]{}

ZK-4566-85

C-2 UIS Fonts
UIS Multinational Character Set Fonts

Figure C-3 Font 3

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890-=!@#%&*()_+
<>,./?;:'"\|[]{}

ZK-4567-85

Figure C-4 Font 4

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890-=!@#%&*()_+
<>,./?;:'"\|[]{}

ZK-4568-85

Figure C-5 Font 5

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890-=!@#%&*()_+
<>,./?;:'"\|[]{}

ZK-4569-85

UIS Multinational Character Set Fonts

Figure C-6 Font 6

ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz
 1234567890-!@#%&^*()_+
 <>.,/?;:'"\|[]{}

ZK-4570-85

Figure C-7 Font 7

ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz
 1234567890-!@#%&^*()_+
 <>.,/?;:'"\|[]{}

ZK-4571-85

Figure C-8 Font 8

ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz
 1234567890-!@#%&^*()_+
 <>.,/?;:'"\|[]{}

ZK-4572-85

C-4 UIS Fonts
UIS Multinational Character Set Fonts

Figure C-9 Font 9

```
ABCDEFGHIJKLMNPOQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz  
1234567890-=!@#%&^*()_+  
<>,./?;:'"\|[]{}
```

ZK-4573-85

Figure C-10 Font 10

```
ABCDEFGHIJKLMNPOQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz  
1234567890-=!@#%&^*()_+  
<>,./?;:'"\|[]{}
```

ZK-4574-85

Figure C-11 Font 11

```
ABCDEFGHIJKLMNPOQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz  
1234567890-=!@#%&^*()_+  
<>,./?;:'"\|[]{}
```

ZK-4575-85

Figure C-12 Font 12

**ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890- = ! @ # \$ % ^ & * () _ +
< > , . / ? ; : ' " \ | [] { }**

ZK-4576-85

Figure C-13 Font 13

**ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890- = ! @ # \$ % ^ & * () _ +
< > , . / ? ; : ' " \ | [] { }**

ZK-4577-85

Figure C-14 Font 14

**ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890- = ! @ # \$ % ^ & * () _ +
< > , . / ? ; : ' " \ | [] { }**

ZK-4578-85

C.2.1 UIS Multinational Character Set Font Specifications

Each font file name, included in the following table captions contains typographical information about a UIS font. The accompanying tables analyzes the first 16 characters of the font file name.

C-6 UIS Fonts
UIS Multinational Character Set Fonts

Table C-1 Font 1—DTABER0003WK00PG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	0	Proportionally spaced
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-2 Font 2—DTABER0I03WK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	I	9 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-3 Font 3—DTABER0M03CK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	M	13 pitch (monospaced)
9-11	Type size	03C ₃₆	12 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

UIS Multinational Character Set Fonts

Table C-4 Font 4—DTABER0R03WK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-5 Font 5—DTABER0R07SK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	07S ₃₆	28 points (280 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-6 Font 6—DTERMING03CK00PG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TERMIN	Terminal
8	Spacing	G	7 pitch (monospaced)
9-11	Type size	03C ₃₆	12 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

C-8 UIS Fonts
UIS Multinational Character Set Fonts

Table C-7 Font 7—DTERMINM06OK00PG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TERMIN	Terminal
8	Spacing	M	13 pitch (monospaced)
9-11	Type size	06O ₃₆	24 points (240 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-8 Font 8—DTABER0003WK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	0	proportionally spaced
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-9 Font 9—DTABER0G03CK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	G	7 pitch (monospaced)
9-11	Type size	03C ₃₆	12 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

UIS Multinational Character Set Fonts

Table C-10 Font 10—DTABER0I03WK00PG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	I	9 (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-11 Font 11—DTABER0M06OK00GG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	M	13 pitch (monospaced)
9-11	Type size	06O ₃₆	24 points (240 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-12 Font 12—DTABER0R03WK00PG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

C-10 UIS Fonts

UIS Multinational Character Set Fonts

Table C-13 Font 13—DTABER0R07SK00PG0001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TABER0	Taber
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	07S ₃₆	28 points (280 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-14 Font 14—DTERMINM03CK00PG001UZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	TERMIN	Terminal
8	Spacing	M	13 pitch (monospaced)
9-11	Type size	03C ₃₆	12 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

C.3 UIS Technical Character Set Fonts

There are 12 technical character set font files in SYS\$FONT. The figure captions below identify each UIS font with an arbitrarily assigned font number. For more information about font characteristics, match this number with the appropriate table in Section C.3.1.

Figure C-15 Font 15

α δ χ ε ε ø γ η ι θ κ λ μ ν ο π ρ σ τ ϕ ω ξ υ ζ
 < < \ / - - > } | ≠ [] { | n d u

Figure C-16 Font 16

α β χ δ ε ϑ γ λ η θ κ λ ϑ υ ρ π ψ ρ σ τ ϑ ω ξ υ ζ
< < \ / - - > } | ≠ [] { [n ɔ u

ZK-5375-86

Figure C-17 Font 17

α β χ δ ε ϑ γ η λ θ κ λ ϑ υ ρ π ψ ρ σ τ ϑ
ω ξ υ ζ < < \ / - - > } | ≠ [] { [n ɔ u

ZK-5374-86

Figure C-18 Font 18

α β χ δ ε ϑ γ η λ θ κ λ ϑ υ ρ π ψ ρ
σ τ ϑ ω ξ υ ζ < < \ / - - > } | ≠ []
] { [n ɔ u

ZK-5373-86

Figure C-19 Font 19

α β χ δ ε ϑ γ η λ θ κ λ ϑ υ
ϑ π ρ σ τ ϑ γ λ η θ κ λ ϑ υ
/ | | > < | | ≠ [] { [n ɔ u

ZK-5372-86

C-12 **UIS Fonts**
UIS Technical Character Set Fonts

Figure C-20 Font 20

R R R R R R R R R R R R R R R R
 N N N N N N N N N N N N N N N N
 M M M M M M M M M M M M M M M M
 L L L L L L L L L L L L L L L L
 P P P P P P P P P P P P P P P P
 Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q
 S S S S S S S S S S S S S S S S
 T T T T T T T T T T T T T T T T
 U U U U U U U U U U U U U U U U
 V V V V V V V V V V V V V V V V
 W W W W W W W W W W W W W W W W
 X X X X X X X X X X X X X X X X
 Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
 Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z

ZK-5382-86

Figure C-21 Font 21

a a x o e ø y n l o k h
 e v b p f r o t f w e u
 z \ / \ / \ / \ / \ / \ / \ / \ / \ /
 j { | n u u u u u u u u u u u u u u u u

ZK-5381-86

Figure C-22 Font 22

a a x o e ø y n l o k h
 e v b p f r o t f w e u
 z \ / \ / \ / \ / \ / \ / \ / \ / \ /
 j { | n u u u u u u u u u u u u u u u u

ZK-5383-86

Figure C-23 Font 23

α	β	χ	δ	ε	ø	γ	η	ι	θ	κ
λ	⇌	ν	ρ	σ	τ	ϕ	ζ	υ	ϕ	ξ
ω	~	ζ	ν	λ	π	ϕ	π	υ	ϕ	ξ
ι	#	ι	ι	ι	ι	ι	ι	ι	ι	ι

ZK-5380-86

Figure C-24 Font 24

α	β	χ	δ	ε	ø	γ	η	ι	θ	κ
λ	⇌	ν	ρ	σ	τ	ϕ	ζ	υ	ϕ	ξ
ω	~	ζ	ν	λ	π	ϕ	π	υ	ϕ	ξ
ι	#	ι	ι	ι	ι	ι	ι	ι	ι	ι

ZK-5379-86

Figure C-25 Font 25

α	β	χ	δ	ε	ø	γ	η	ι	θ
κ	λ	⇌	ν	ρ	π	ψ	φ	σ	τ
φ	ω	ε	υ	ζ	ν	λ	ϕ	σ	ι
ι	~	ι	ι	#	ι	ι	ι	ι	ι
υ	υ								

ZK-5378-86

UIS Technical Character Set Fonts

Table C-16 Font 16—DVWSVT0G03CK00PG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	G	7 pitch (monospaced)
9-11	Type size	03C ₃₆	12 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-17 Font 17—DVWSVT0I03WK00GG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	I	9 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-18 Font 18—DVWSVT0I03WK00PG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	I	9 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

C-16 UIS Fonts
UIS Technical Character Set Fonts

Table C-19 Font 19—DVWSVT0N03CK00GG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	N	14 pitch (monospaced)
9-11	Type size	03C ₃₆	120 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-20 Font 20—DVWSVT0N03CK00PG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	N	14 pitch (monospaced)
9-11	Type size	03C ₃₆	12 points (120 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-21 Font 21—DVWSVT0N06OK00GG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	N	14 pitch (monospaced)
9-11	Type size	06O ₃₆	24 points (240 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

UIS Technical Character Set Fonts

Table C-22 Font 22—DVWSVT0N06OK00PG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	N	14 pitch (monospaced)
9-11	Type size	06O ₃₆	24 points (240 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

Table C-23 Font 23—DVWSVT0R03WK00GG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-24 Font 24—DVWSVT0R03WK00PG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	03W ₃₆	14 points (140 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	P	Bold
16	Proportion	G	Regular

C-18 UIS Fonts
UIS Technical Character Set Fonts

Table C-25 Font 25—DVWSVT0R07SK00GG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	07S ₃₆	28 points (280 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Table C-26 Font 26—DVWSVT0R07SK00GG0001QZZZZ02A000

Field	Field Name	Value	Meaning
1	Registration code	D	Registered by DIGITAL
2-7	Type Family ID	VWSVT0	VAXstation Technical Character Set
8	Spacing	R	18 pitch (monospaced)
9-11	Type size	07S ₃₆	28 points (280 decipoints)
12	Scale factor	K	1 (normal)
13-14	Style	00 ₃₆	Roman
15	Weight	G	Regular
16	Proportion	G	Regular

Appendix D

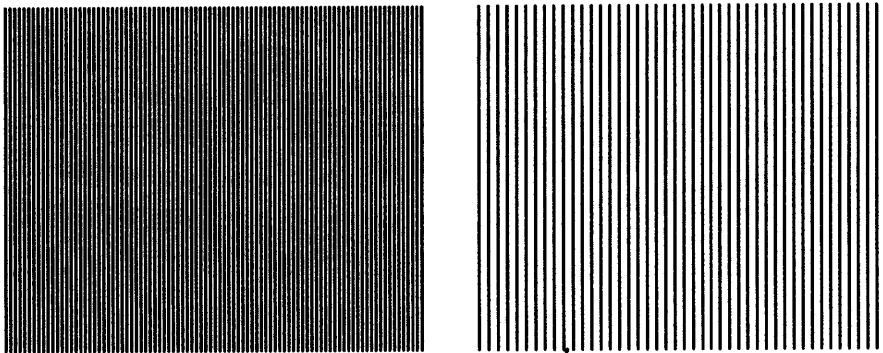
UIS Fill Patterns

All fill patterns are located together in SYS\$FONT in a separate file named DEUI\$PATAAAAAAAAAF000000000DA.VWS\$FONT. This file can be accessed with the logical name UIS\$FILL_PATTERNS.

The pairs of fill patterns shown in the following figures were drawn in OVERLAY writing mode on a white background. The figure caption contains the symbol name for each fill pattern. The symbol name represents an index to the appropriate fill pattern.

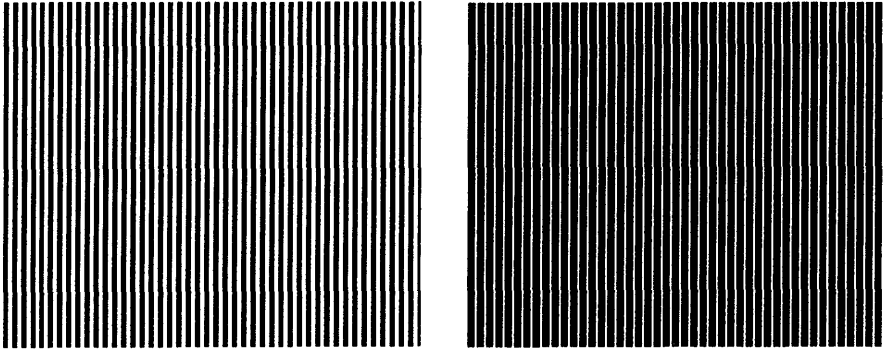
Symbol names are located in language-specific symbol definition files in SYS\$LIBRARY. Refer to Table 6-2 for a list of symbol definition files.

Figure D-1 PATT\$C_VERT1_1 and PATT\$C_VERT1_3



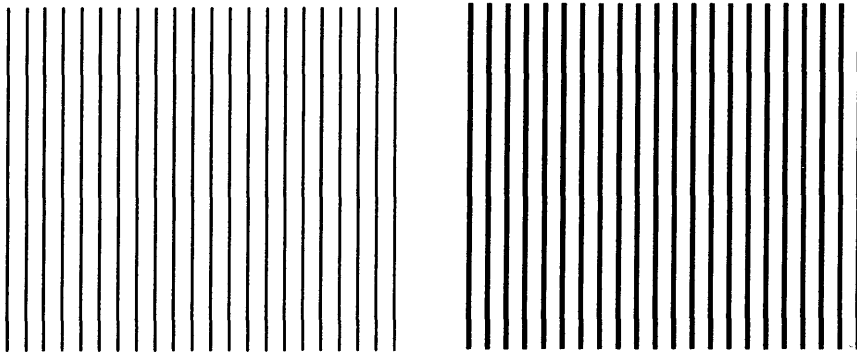
D-2 UIS Fill Patterns

Figure D-2 PATT\$C_VERT2_2 and PATT\$C_VERT3_1



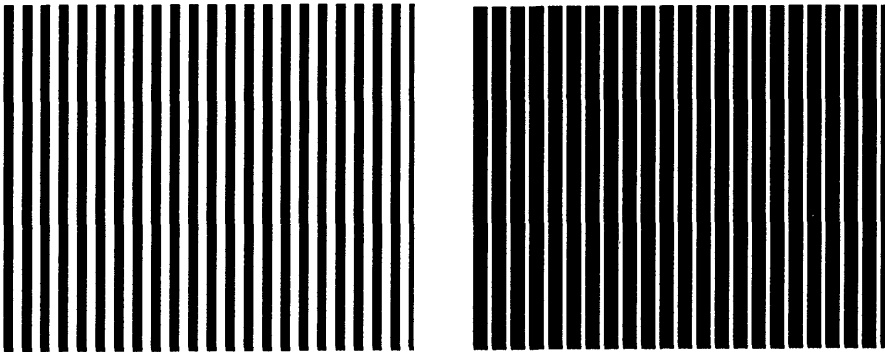
ZK-4585-85

Figure D-3 PATT\$C_VERT1_7 and PATT\$C_VERT2_6



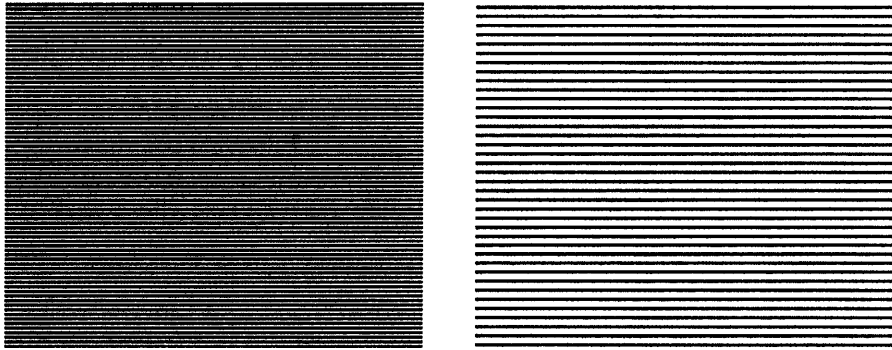
ZK-4586-85

Figure D-4 PATT\$C_VERT4_4 and PATT\$C_VERT6_2



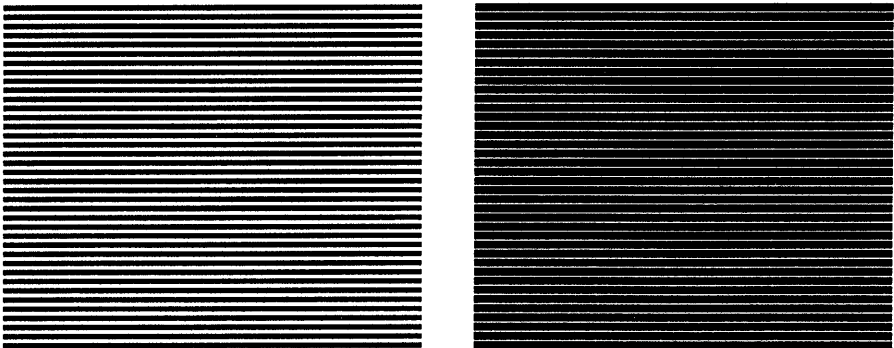
ZK-4587-85

Figure D-5 PATT\$C_HORIZ1_1 and PATT\$C_HORIZ1_3



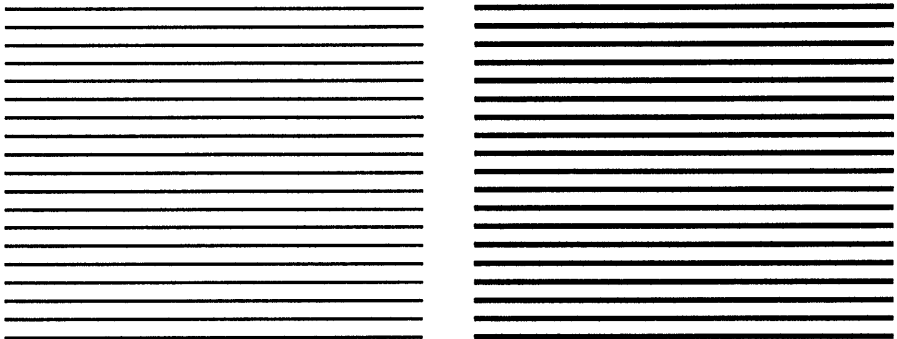
ZK-4588-85

Figure D-6 PATT\$C_HORIZ2_2 and PATT\$C_HORIZ3_1



ZK-4589-85

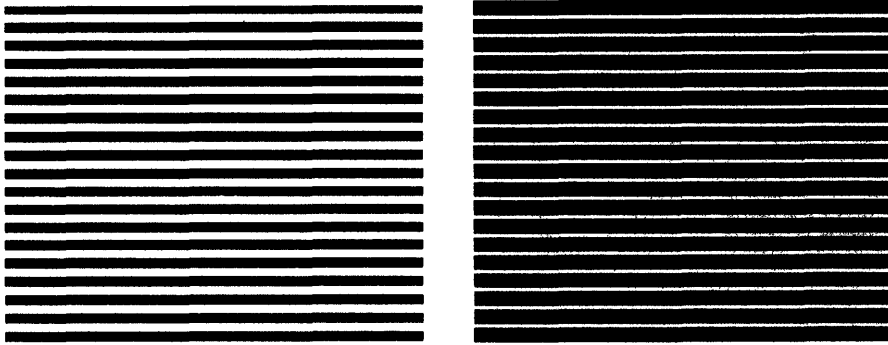
Figure D-7 PATT\$C_HORIZ1_7 and PATT\$C_HORIZ2_6



ZK-4590-85

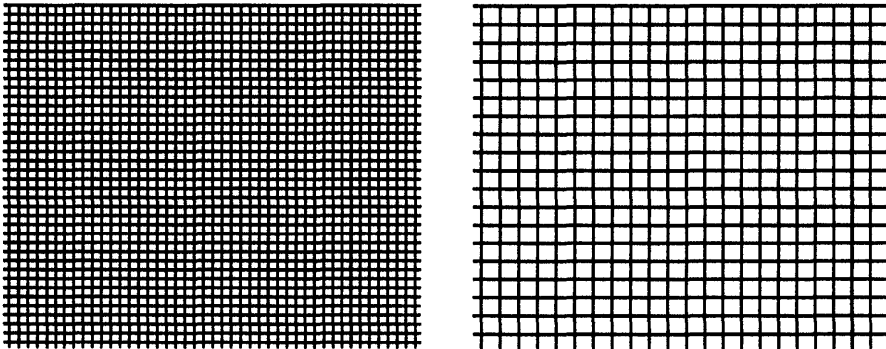
D-4 UIS Fill Patterns

Figure D-8 PATT\$C_HORIZ4_4 and PATT\$C_HORIZ6_2



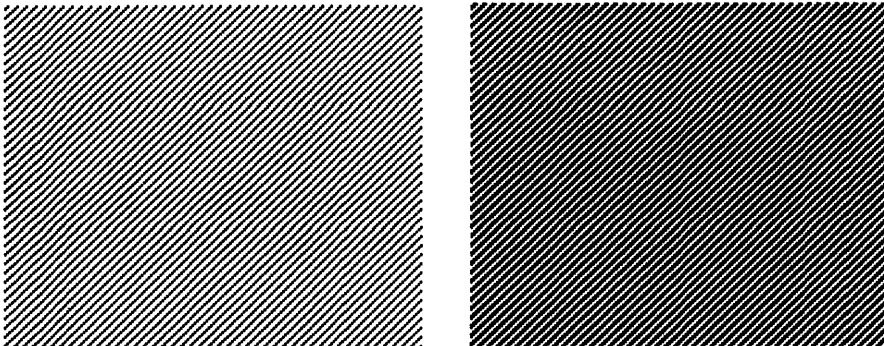
ZK-4591-85

Figure D-9 PATT\$C_GRID4 and PATT\$C_GRID8



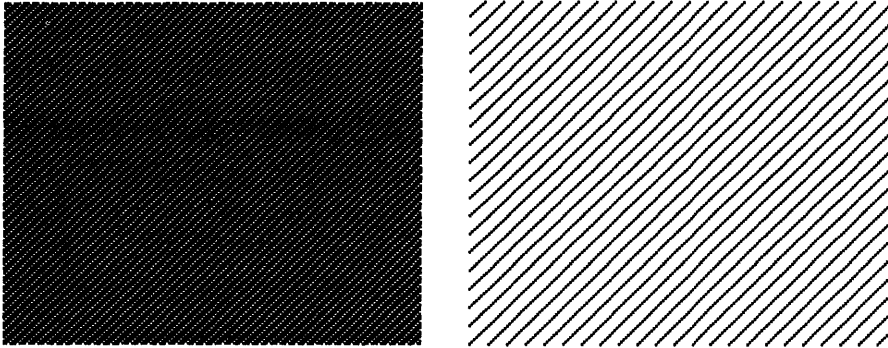
ZK-4592-85

Figure D-10 PATT\$C_UPDIAG1_3 and PATT\$C_UPDIAG2_2



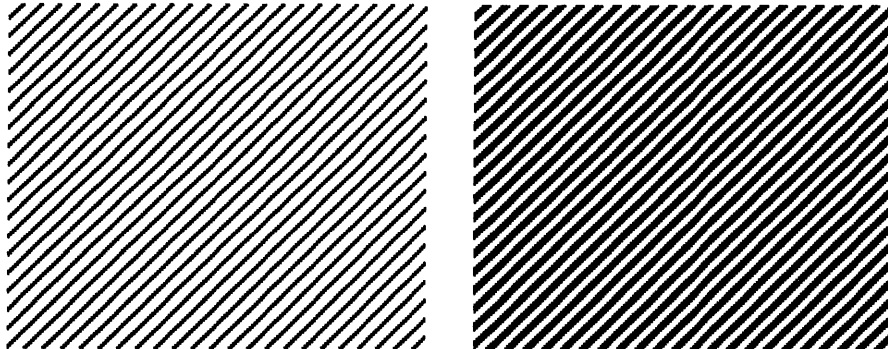
ZK-4593-85

Figure D-11 PATT\$_C_UPDIAG3_1 and PATT\$_C_UPDIAG1_7



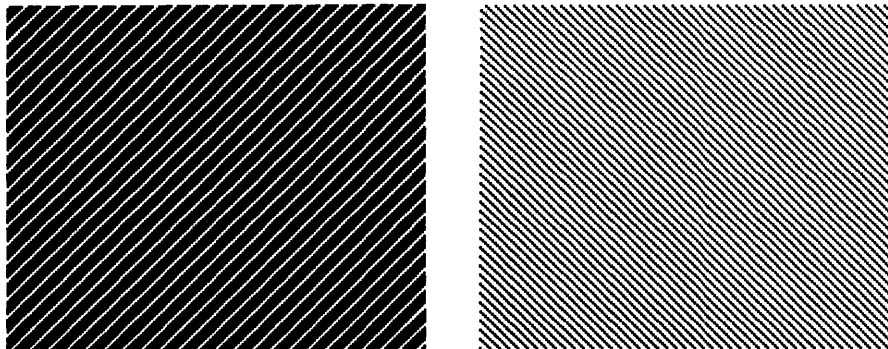
ZK-4594-85

Figure D-12 PATT\$_C_UPDIAG2_6 and PATT\$_C_UPDIAG4_4



ZK-4595-85

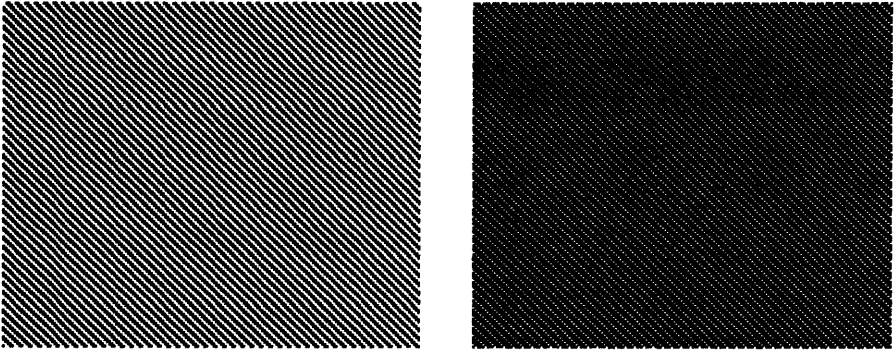
Figure D-13 PATT\$_C_UPDIAG6_2 and PATT\$_C_DOWNDIAG1_3



ZK-4596-85

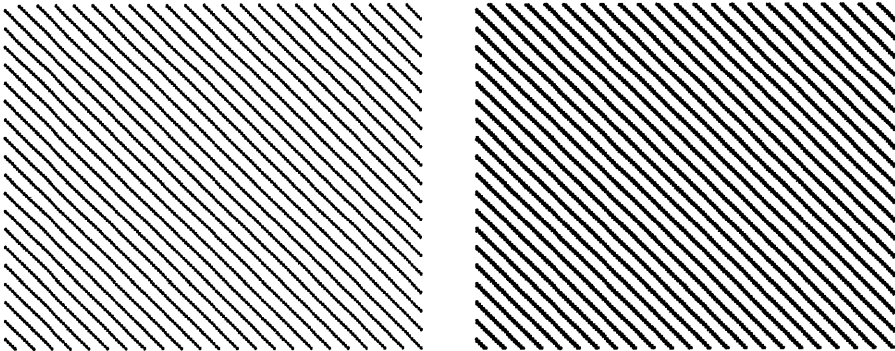
D-6 UIS Fill Patterns

Figure D-14 PATT\$C_DOWNDIAG2_2 and PATT\$C_DOWNDIAG3_1



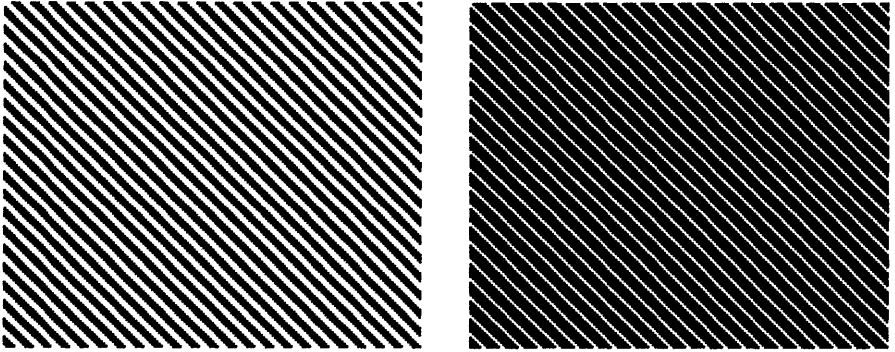
ZK-4597-85

Figure D-15 PATT\$C_DOWNDIAG1_7 and PATT\$C_DOWNDIAG2_6



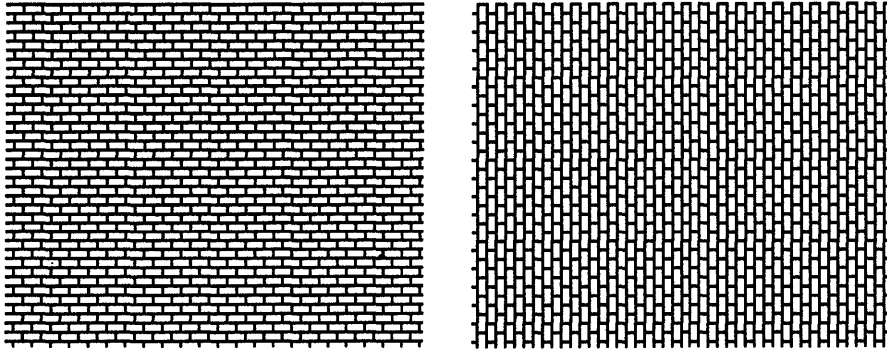
ZK-4598-85

Figure D-16 PATT\$C_DOWNDIAG4_4 and PATT\$C_DOWNDIAG6_2



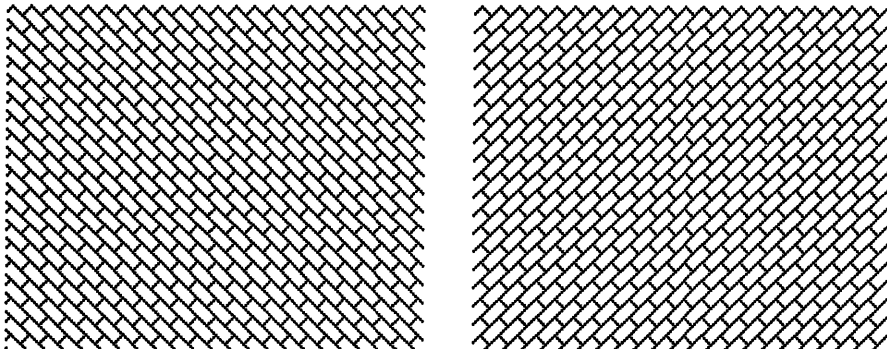
ZK-4599-85

Figure D-17 PATT\$C_BRICK_HORIZ and PATT\$C_BRICK_VERT



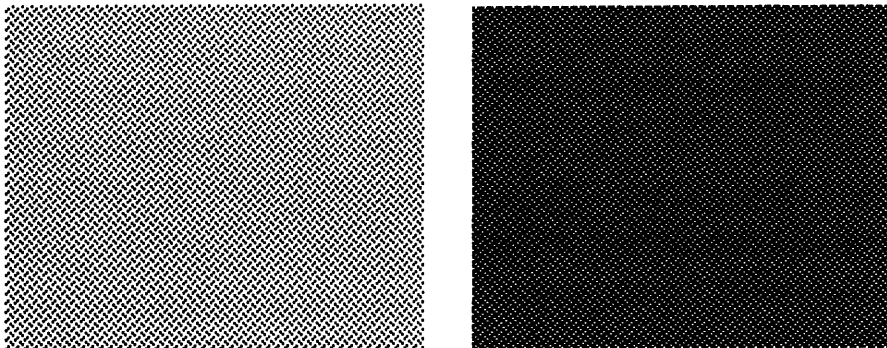
ZK-4600-85

Figure D-18 PATT\$C_BRICK_DOWNDIAG and PATT\$C_BRICK_UPDIAG



ZK-4601-85

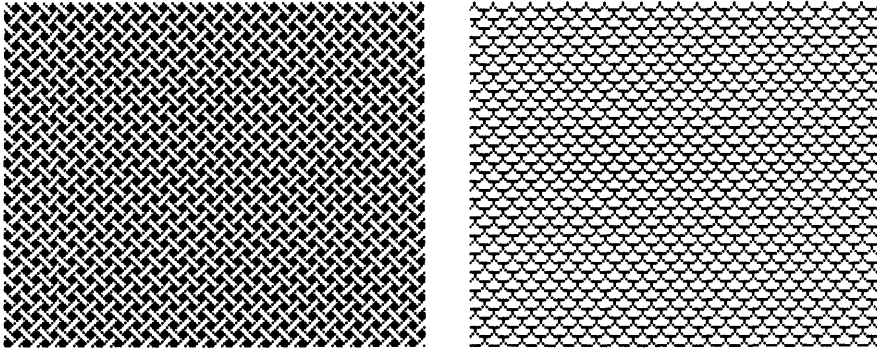
Figure D-19 PATT\$C_GREY4_16D and PATT\$C_GREY12_16D



ZK-4602-85

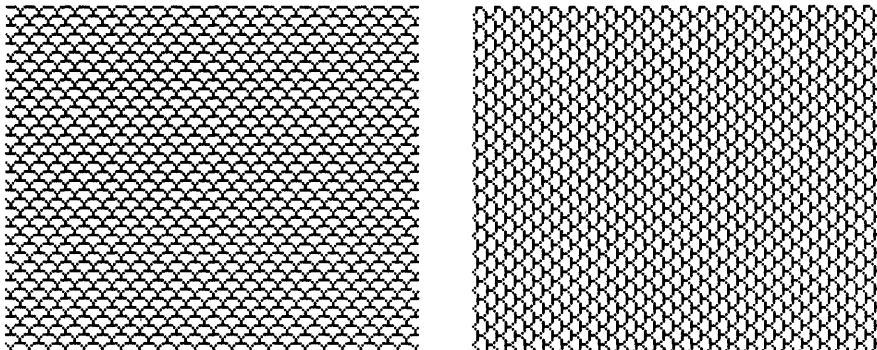
D-8 UIS Fill Patterns

Figure D-20 PATT\$C_BASKET_WEAVE and PATT\$C_SCALE_DOWN



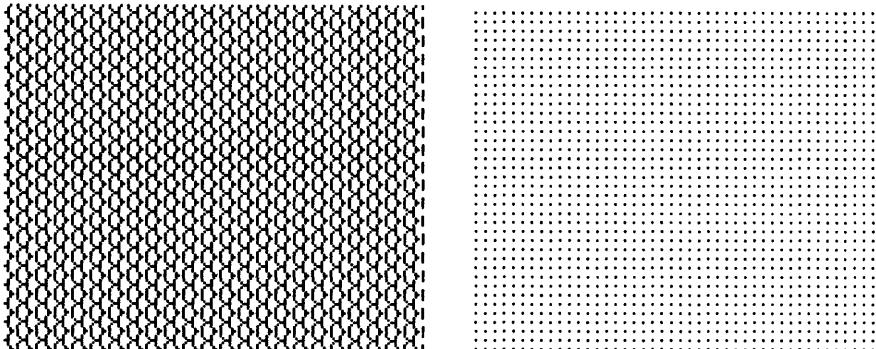
ZK-4603-85

Figure D-21 PATT\$C_SCALE_UP and PATT\$C_SCALE_RIGHT



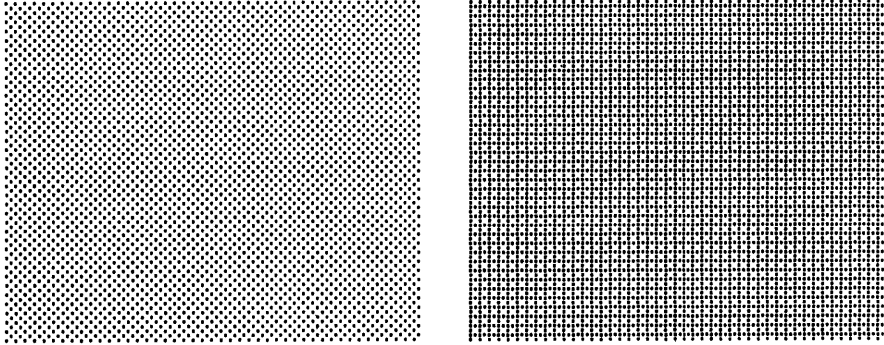
ZK-4604-85

Figure D-22 PATT\$C_SCALE_LEFT and PATT\$C_GREY1_16



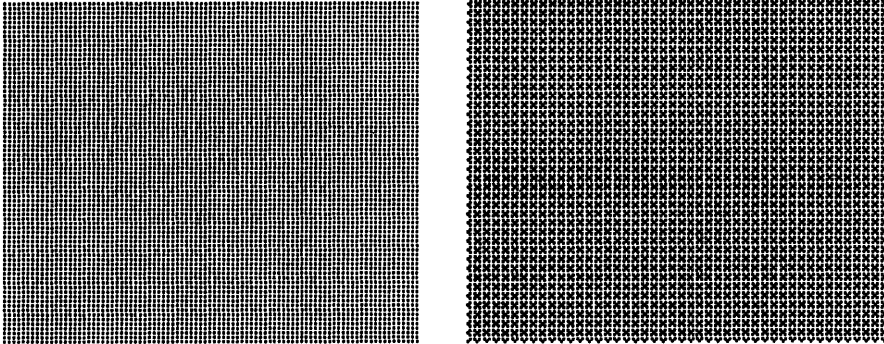
ZK-4606-85

Figure D-23 PATT\$C_GREY2_16 and PATT\$C_GREY3_16



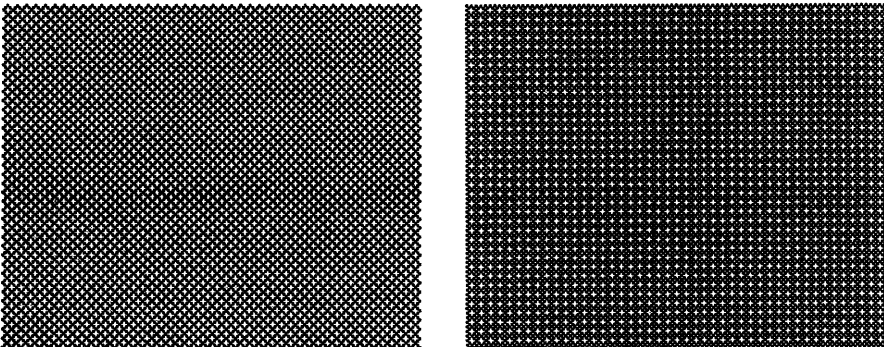
ZK-4607-85

Figure D-24 PATT\$C_GREY4_16 and PATT\$C_GREY5_16



ZK-4608-85

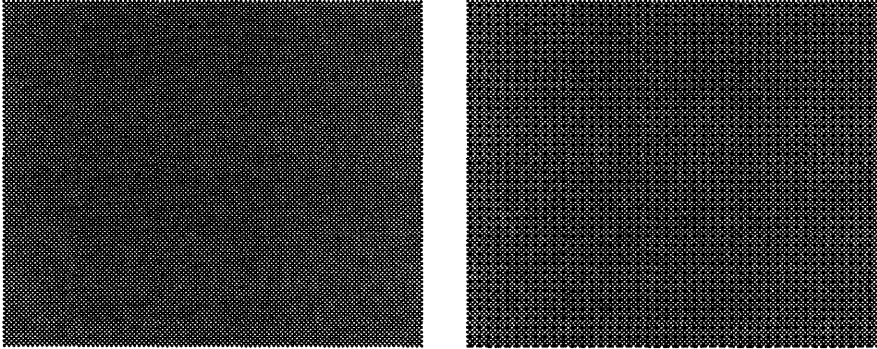
Figure D-25 PATT\$C_GREY6_16 and PATT\$C_GREY7_16



ZK-4609-85

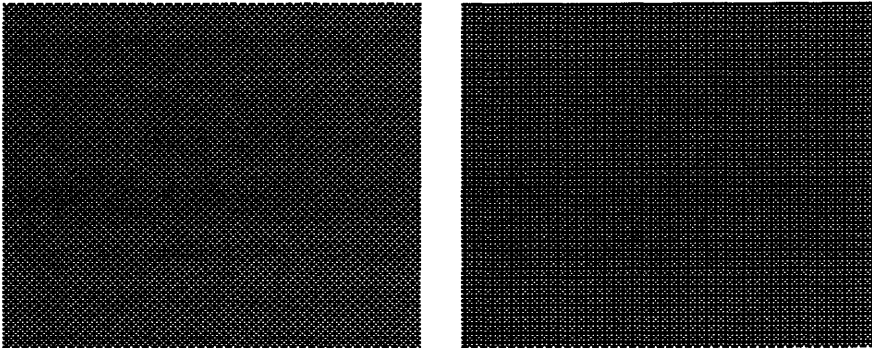
D-10 UIS Fill Patterns

Figure D-26 PATT\$C_GREY8_16 and PATT\$C_GREY9_16



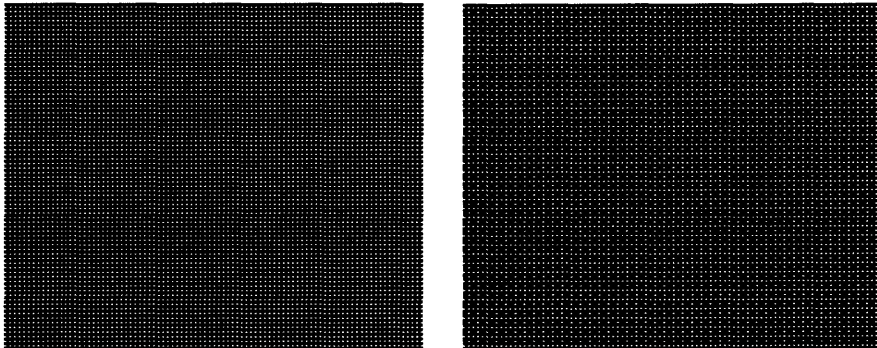
ZK-4610-85

Figure D-27 PATT\$C_GREY10_16 and PATT\$C_GREY11_16



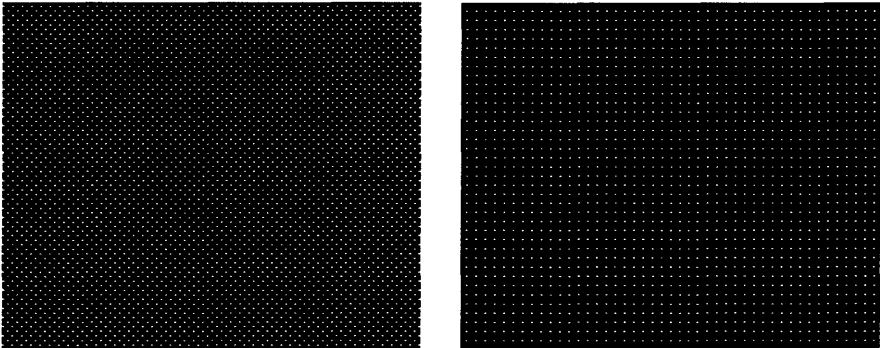
ZK-4611-85

Figure D-28 PATT\$C_GREY12_16 and PATT\$C_GREY13_16



ZK-4612-85

Figure D-29 PATT\$_GREY14_16 and PATT\$_GREY15_16



ZK-4613-85

Appendix E

Error Messages

This appendix contains the messages which may be generated by the MicroVMS workstation graphics software. Each message description consists of the message text, a brief explanation of the message, and the possible remedy.

BAD_ATB, Illegal attempt to modify attribute block 0 (read-only).

Explanation: An attempt was made to modify an attribute in attribute block #0, which is defined to be read-only. The modification request is ignored.

User Action: Check for a programming error.

BAD_DISP, Display list has been corrupted.

Explanation: An illegal display list type code has been encountered while traversing a display list.

User Action: Check the validity of the UIS metafile you are executing.

BAD_KB, Illegal virtual keyboard identifier.

Explanation: An illegal virtual keyboard identifier was given to a UIS routine as an argument.

User Action: Check for a programming error.

BADTITLE, Illegal window title string.

Explanation: An illegal window title string was passed when attempting to create a window.

User Action: Shorten the title.

BAD_TR, Illegal transformation identifier.

Explanation: An illegal transformation identifier was given to a UIS routine as an argument.

User Action: Check for a programming error.

E-2 Error Messages

BAD_VD, Illegal virtual display identifier.

Explanation: An illegal virtual display identifier was given to a UIS routine as an argument.

User Action: Check for a programming error.

BAD_VOLUME, Illegal volume level specified.

Explanation: An illegal volume level was given to the UIS\$SOUND routine. The volume must be in the range of 1 to 8.

User Action: Check for a programming error.

BAD_WD, Illegal display window identifier.

Explanation: An illegal display window identifier was given to a UIS routine as an argument.

User Action: Check for a programming error.

BADWDPL, Window placement attribute list has an invalid format.

Explanation: An illegal window attribute list was passed when attempting to create a window.

User Action: Check for illegal item types in the window attribute list.

INSFARG, Insufficient arguments.

Explanation: A required argument was not specified.

User Action: Check for a programming error.

NO_FONT, The font cannot be found.

Explanation: An attempt was made to reference a font which could not be satisfied, even by looking for other fonts which might be similar. All references to the attribute block specifying this font will produce this same error. The program may continue after this error.

User Action: Specify font contained in the SYS\$FONT directory.

NOURG, Cannot disable region AST because no matching region can be found.

Explanation: An attempt was made to disable a user region AST by using an ASTADR=0 and the region boundary used in the original enable request. However, no entry can be found with matching boundary coordinates. The program must ensure that the boundary coordinates match exactly in order to disable an existing request.

User Action: Check for a programming error.

VPTOOSMALL, Requested size of the viewport is too small.

Explanation: The desired size of the viewport is too small to be displayed on the screen.

User Action: Request larger viewport.

Appendix F

Obsolete Routines

The following routines are obsolete and will no longer be documented.

- `UIS$GET_LEFT_MARGIN`
- `UIS$SET_LEFT_MARGIN`
- `UISDC$GET_LEFT_MARGIN`
- `UISDC$SET_LEFT_MARGIN`

Glossary

array: Any organized arrangement of related elements.

address: A 32-bit VAX address positioned in a longword item.

argument list: A vector of longwords that represents a procedure parameter list and possibly a function value.

aspect ratio: The ratio between the height and width of a graphic object. In reference to a virtual display, the aspect ratio is a comparison of the relative proportions of the vertical and horizontal components of objects in the virtual display.

attribute: A quality or characteristic that determines the appearance of an object displayed on the screen. For example, the attributes of a line are its width, style, and color.

baseline: The side of a geometric object or drawing from which the object is constructed or drawn.

call: The transfer of processing control to a specified subroutine.

Cartesian coordinate system: A system of measuring distances in which the location of a point is defined as its distance from two straight lines that intersect at right angles. It is used as the basis of coordinate measurements in computer graphics systems.

clipping: Any graphic data outside a specified boundary that are removed from the display or the file. It is often used in mapping applications to remove data that would otherwise confuse the image being represented.

clipping rectangle: The physical limit in a graphics file beyond which data are either not visible or automatically deleted.

2-Glossary

condition value: A 32-bit value used to identify uniquely an exception condition. A condition value may be returned to a calling program as a function value or signaled using the VAX signaling mechanism.

current text position: The world coordinate position that defines the current drawing location for UIS text routines.

cursor: A position indicator used on a display screen to pinpoint where data will be displayed. The cursor is often represented by a blinking block character.

data tablet: The name for a variety of data entry devices consisting of a stylus (pen) or puck, and a board with a coordinate grid superimposed on its surface. When the input object (pen or puck) touches the board, graphic information describing the location of the point touched is transmitted as input information. The data tablet is an absolute pointing device.

descriptor: A mechanism for passing parameters in which the address of a descriptor is provided in the longword argument list entry. The descriptor contains the address of the parameter, the data type, size, and additional information needed to describe fully the data passed.

device coordinates: The device-dependent Cartesian coordinates that specify positions on the MicroVMS display screen. Sometimes referred to as physical device coordinates, these coordinates are involved in mapping of the display window to the display screen.

display viewport: The area of the physical display screen into which a display window is mapped. It is the physical region on the terminal screen that is created by the MicroVMS workstation and controlled by the user.

display window: The portion of world coordinate space mapped to the graphics viewport. The display window is used to control how much of the virtual display is potentially available for the user to view.

emulated terminal: A virtual I/O device whose programming interface matches the programming interface of a specific physical terminal and whose appearance on the MicroVMS workstation screen is similar to the appearance of the physical terminal.

exception condition: A hardware- or software-detected event that alters the normal flow of instruction execution.

- font:** A specific representation of a text character. The attributes of a font are family (type face), type size, and rendition.
- function:** A procedure that returns a single value according to standard conventions. If additional values are returned, they are returned by means of the argument list.
- graphics data tablet:** An optional input device that consists of a rigid tablet, and a puck containing a crosshair cursor and a number of buttons, or a pen. The position of the cursor can be read by application programs. The tablet is an absolute pointing device.
- graphics display:** Describes any graphics data output device that can present an image of graphic data derived from a computer graphics system. An example of a graphics display is a display screen or a printer.
- graphic object:** The graphic image constructed by an application program using UIS routines. A graphic object could be a simple line or a complex drawing.
- graphics text:** Text output primitives displayed using the UIS routines.
- grey scale:** The level of brightness that describes the illumination of a cathode-ray tube screen.
- image:** The output form of on-line graphics data. That is, a displayed or drawn representation of a graphics file.
- language-support procedures:** Procedures called implicitly to implement high-level language constructs. They are not intended to be called explicitly from user programs.
- library procedures:** Procedures called explicitly using the equivalent of a CALL statement or function reference. They are usually language independent.
- mapping:** Any process by which a graphics system translates graphic data from one coordinate system into a form useful on another coordinate system.
- mouse:** A data entry device consisting of a small control box, on rollers, that is pushed along a surface and transmits its changing position to the workstation. Often, function keys or buttons are mounted on the device and can be used to enter information or make selections. This device is the user's means for pointing to and selecting objects on the screen. The mouse is a relative pointing device.

4-Glossary

output primitive: A part of an image created with UIS procedures, such as a graphics object or a text string, that has a specific appearance. Values of attributes determine some aspects of this appearance.

physical device coordinates: Device-dependent Cartesian coordinates that specify the addressable points on a physical device.

pixel: The density of one picture element. The smallest displayable unit on a display screen.

pointer: The cursor on the screen that tracks movements of the mouse. The shape of the pointer depends upon its current use.

primitives: The most basic graphic entities available on a graphics system, such as points, line segments, or characters.

procedure: A closed sequence of instructions that is entered from, and returns control to, the calling program.

puck: A hand-held graphics device with a cross hair sight used to pinpoint coordinates on a data tablet or digitizer.

raster: A pattern of scanning lines in a cathode-ray tube which divide the display area into addressable points.

reference: A mechanism for passing parameters in which the address of the parameter is provided in the longword argument list by the calling program.

resizing: The process of scaling or changing the size of a graphics viewport according to predetermined data.

stretchy box: The outline of a clipping rectangle used in the UIS functions PRINT SCREEN and RESIZE WINDOW. This rectangle can be manipulated to assume practically any rectangular dimensions and is limited only by the display screen size.

subroutine: A procedure that does not return a value according to the standard conventions. If values are returned, they are returned by means of the argument list.

transformations: The ability of the UIS graphics system to manipulate coordinate data. Transformations occur when mapping one coordinate system into another coordinate system.

tablet: A device which can convert a stylus position into Cartesian coordinates. When connected to a graphic display screen, it can control the real-time positioning of a cursor or pointer.

UIS: The graphics software called User Interface Services.

value: A mechanism for passing input parameters in which the actual value is provided in the longword argument list entry by the calling program.

viewport: A rectangle that maps the image defined by a window into a virtual display onto the display screen. The user controls the visibility and placement of viewports on the physical screen.

viewing transformation: The viewing transformation is the process of mapping the world coordinates of a graphic object in a display window to the device coordinates of a display viewport on a physical display device.

virtual display: The world coordinate space defined by an application program. An application program uses a virtual display as a place in which to build graphic images. It can be thought of as a virtual output device that has the properties of a physical screen, but is not necessarily visible on a physical screen.

virtual keyboard: A virtual input device associated with a window. When users select a window into a virtual display with a virtual keyboard, input from the physical keyboard is directed to the virtual keyboard and can be read by an application program.

window: A defined area within a virtual display that can be used for viewing the virtual display. A window is the area of the virtual display that is to be mapped to a viewport.

world coordinates: Device-independent Cartesian coordinates defined by the application program in order to describe objects to UIS.

x axis: The reference line of a rectangular coordinate system used to determine horizontal distance and positions.

x-height: The height of lowercase characters excluding descenders and ascenders.

6-Glossary

y axis: The reference line of a rectangular coordinate system used to determine vertical distance and positions.

zooming: The process by which the perspective on a displayed graphics file moves rapidly closer or farther from the operator.

Index

A

Arc type
 See Attribute

Argument
 characteristics of
 passing mechanism, 6-3

Argument passing mechanism
 %DESCR, 6-7
 %LOC, 6-7
 %REF, 6-7
 %VAL, 6-7

Aspect ratio, 8-3

AST-enabling routine, 17-2

AST routine, 17-1

Asynchronous system trap (AST) routine,
 17-1

Attribute, 3-2, 9-1
 See also Attribute block
 See also Attribute block 0
 See also Segment
 description of, 3-2
 general, 3-2, 3-3, 9-8
 background color index, 3-3, 9-8
 modifying, 9-3, 11-1
 writing color index, 3-3, 9-9
 writing mode, 3-3, 9-9
 graphics, 3-2, 3-5, 11-1
 arc type, 3-5, 11-3
 fill patterns, 3-5, 11-2
 line style, 3-5, 11-4
 line width, 3-5, 11-4
 text, 3-2, 3-3, 10-21
 centering, 10-24
 character rotation, 10-24
 character scaling, 10-25

Attribute
 text (cont'd.)
 character slant, 10-25
 character spacing, 3-3, 10-24
 fonts, 3-3, 10-23
 formatting mode, 10-24
 justification, 10-24
 kerning, 10-24
 leading, 10-24
 left margin, 3-3
 line spacing, 10-24
 modifying, 10-22
 path, 10-25
 slope, 10-25
 text margin, 10-24
 window
 clipping rectangle, 3-5
 windowing
 clipping rectangle, 11-14

Attribute block, 3-6, 9-2
 See also Attribute
Attribute block 0, 3-6, 9-2
 See also Attribute
Attribute routine, 9-3, 10-22, 11-1

B

Background color index
 See Attribute

Baseline
 See Text output

Built-in function
 See Argument passing mechanism

Index-2

C

- Callable routine, 6-1
- Calling sequence, 6-2
 - argument characteristics, 6-3
 - argument list, 6-2
 - call type, 6-2
 - entry point name, 6-2
 - routine name, 6-2
 - summary
 - UIS, A-1
 - UISDC, B-1
- CALL statement, 6-1
- Character rotation
 - See Attribute
- Character scaling
 - See Attribute
- Character slant
 - See Attribute
- Character spacing
 - See Attribute
- Clipping rectangle
 - see Attribute
- Color
 - See color system
- Color map
 - See Color system
- Color map segment
 - See Color system
- Color system
 - color, 4-1
 - preferred, 4-11
 - standard, 4-11
 - color map
 - hardware, 4-3
 - segment, 4-11
 - virtual, 4-7
 - color regeneration
 - characteristics, 4-13
 - color value conversion, 4-12
 - compatibility feature
 - color, 4-12
 - intensity, 4-12
 - monochrome, 4-12
 - Color system (cont'd.)
 - hardware color map
 - reserved entries, 4-9
 - intensity, 4-1
 - model
 - color, 4-6
 - HLS, 4-6
 - HSV, 4-6
 - RGB, 4-6
 - monochrome, 4-1
 - palette, 4-6
 - palette size
 - direct color, 4-6
 - mapped color, 4-7
 - pixels, 4-1
 - planes, 4-2
 - realized color, 4-13
 - set color, 4-13
 - value
 - color, 4-6
 - direct color, 4-3
 - intensity, 4-6
 - mapped color, 4-3
 - pixel, 4-2
 - virtual color map, 4-7
 - characteristics, 4-9
 - initialization, 4-9
 - private, 4-11
 - shareable, 4-11
 - swapping, 4-7
- Communication tool, 1-6
 - keyboard, 1-7
 - pointer
 - mouse, 1-6
 - tablet, 1-6
- Condition value signaled, 6-9
- Constant, 6-9
- Coordinate
 - device-dependent
 - absolute, 2-6
 - viewport-relative, 2-7
 - device-independent
 - normalized, 2-5
 - world, 2-4
 - types of, 7-2

Coordinate system

- Cartesian, 2-3
- device-dependent, 2-6
 - absolute, 2-3
 - viewport-relative, 2-3
- device-independent, 2-3
 - normalized, 2-3
 - world, 2-3

D

Data definition file

See Data description file

Data description file

- entry point, 6-9
- message, 6-10
- symbol definition, 6-9

Display list, 2-13, 13-1

- disabling, 13-5
- editing, 13-21
- enabling, 13-5
- generic encoding, 2-13
 - metafile, 2-13
- private data, 15-19
- root segment, 13-2
- segment, 13-1
 - creating, 13-5
 - modifying attributes, 13-21
- walking, 13-6

Display list routine, 13-1

Display viewport, 2-10, 7-6

- banner, 8-4
- creating, 7-6, 8-2 to 8-5
- mapping windows to, 2-10
- number, 8-3
- placement, 8-4, 8-12 to 8-25
- popping, 8-12
- pushing, 8-12
- scaling, 2-11
- shrinking, 17-13
- size, 8-3

Display window, 2-9, 7-6

- clipping rectangle, 2-9
- closing, 17-13
- creating, 7-6, 8-2 to 8-5
- deletion, 8-7

Display window (cont'd.)

- distortion, 8-3
- magnification, 8-3
- number, 8-3
- placement, 8-12 to 8-25
- resizing, 17-13
- scaling, 2-11
- size, 8-3
- viewing objects, 2-9

Distortion

See Distortion of graphic objects

Distortion of graphic objects, 2-12

- cause of, 2-12
- correction of, 2-12
- transformations, 2-12

E

Error messages, E-1 to E-3

F

Fill pattern, 11-2, D-1

See also Attribute

Font

- See also Attribute
- font file names, 10-23
- multinational, 10-23
- multinational character, C-1
- SYS\$FONT, 10-23
- technical, 10-23, C-10

Format heading

See Routine format

FORTRAN built-in function, 6-7

Function reference, 6-1

G

General attribute

See Attribute

Generic encoding

See Display list

Graphic object, 3-2, 7-4 to 7-6

attributes, 3-2

Index-4

Graphic object (cont'd.)

- creating, 7-4
- geometric shapes
 - circle, 7-4
 - ellipse, 7-4
 - line, 7-4
 - point, 7-4
 - polygon, 7-4
- raster image, 7-5
- text, 7-5
- viewing transformation, 3-7

Graphics attribute

- See Attribute

Graphics capability, 1-7

- Graphics routine, 7-4
- description of, 3-2

H

Hardware color map

- See Color system

Human interface, 1-5

- See also Terminal emulation capabilities, 1-5
- interaction with user, 1-5

I

Inquiry routine, 12-1

- invoking, 12-7

K

KB icon

- See Virtual keyboard

Keyboard

- See Physical keyboard
- See Virtual keyboard

Keyboard routine, 17-3

L

Line spacing

- See Attribute

Line style

- See Attribute

Line width

- See Attribute

M

Mapping

- See Display viewport

Mapping display window

- See Display viewport

Margin setting

- See Attribute

Message definition file

- See data description file

Metafile

- See Display list

Mouse, 5-2

- menu selection, 5-2

P

Physical keyboard, 5-4

Pixel

- See Color system

Pointer, 5-2, 17-8

- See also Mouse

- See also Tablet

- alternate pattern, 17-10

Pointer routine, 17-8

Preferred color

- See Color system

Private data

- See display list

Program execution, 6-13

- compiling, 6-13

- invoking the editor, 6-13

- linking, 6-14

- running, 6-14

Programming example, 6-12

Puck, 5-3

R

- Routine
 - inquiry
 - AST-enabling, 17-2
 - attribute, 9-3, 10-22, 11-1
 - display list, 13-1
 - graphics, 7-4
 - keyboard, 17-3
 - pointer, 17-8
 - windowing, 8-1
- Routine format
 - format heading, 18-3

S

- Scaling
 - See display viewport
 - See Display window
- Segment, 3-7
 - See also Attribute
 - See Display list
- Segmentation
 - See Display list
- Standard color
 - See Color system
- Stylus, 5-3
- Swapping color map
 - See Color system
- Symbol definition file
 - See data description file

T

- Tablet, 5-3
 - puck, 5-3
 - stylus, 5-3
- Terminal emulation, 1-6
 - TEK4014, 1-6
 - VT220, 1-6
- Text attributes
 - See Attribute
- Text centering
 - See Attribute

- Text justification
 - See Attribute
- Text output, 7-4 to 7-6
 - alignment, 10-25
 - baseline, 10-25
 - creating, 7-4
- Text path
 - See Attribute
- Text routine
 - decription of, 3-2
- Text slope
 - See Attribute
- Transformation
 - attribute, 14-17
 - geometric, 3-7, 14-1
 - complex rotation, 14-9
 - complex scaling, 14-5
 - COPY, 14-10
 - differential scaling, 14-7
 - MOVE, 14-10
 - rotation, 14-8
 - scaling, 14-3
 - simple rotation, 14-8
 - simple scaling, 14-3
 - translation, 14-1
 - uniform scaling, 14-6
 - viewing, 2-1, 3-7
 - world coordinate, 8-29

U

- UIS\$BEGIN_SEGMENT, 18-9
- UIS\$CIRCLE, 18-11
- UIS\$CLOSE_WINDOW, 18-14
- UIS\$COPY_OBJECT, 18-15
- UIS\$CREATE_COLOR_MAP, 18-20
- UIS\$CREATE_COLOR_MAP_SEG, 18-23
- UIS\$CREATE_DISPLAY, 18-26
- UIS\$CREATE_KB, 18-28
- UIS\$CREATE_TB, 18-31
- UIS\$CREATE_TERMINAL, 18-32
- UIS\$CREATE_TRANSFORMATION, 18-34
- UIS\$CREATE_WINDOW, 18-37
- UIS\$DELETE_COLOR_MAP, 18-46
- UIS\$DELETE_COLOR_MAP_SEG, 18-47

Index-6

UIS\$DELETE_DISPLAY, 18-48
UIS\$DELETE_KB, 18-49
UIS\$DELETE_OBJECT, 18-50
UIS\$DELETE_PRIVATE, 18-51
UIS\$DELETE_TB, 18-52
UIS\$DELETE_TRANSFORMATION, 18-53
UIS\$DELETE_WINDOW, 18-54
UIS\$DISABLE_DISPLAY_LIST, 18-55
UIS\$DISABLE_KB, 18-58
UIS\$DISABLE_TB, 18-59
UIS\$DISABLE_VIEWPORT_KB, 18-60
UIS\$ELLIPSE, 18-61
UIS\$ENABLE_DISPLAY_LIST, 18-65
UIS\$ENABLE_KB, 18-68
UIS\$ENABLE_TB, 18-70
UIS\$ENABLE_VIEWPORT_KB, 18-71
UIS\$END_SEGMENT, 18-72
UIS\$ERASE, 18-73
UIS\$EXECUTE, 18-75
UIS\$EXECUTE_DISPLAY, 18-77
UIS\$EXPAND_ICON, 18-78
UIS\$EXTRACT_HEADER, 18-81
UIS\$EXTRACT_OBJECT, 18-83
UIS\$EXTRACT_PRIVATE, 18-85
UIS\$EXTRACT_REGION, 18-88
UIS\$EXTRACT_TRAILER, 18-91
UIS\$FIND_PRIMITIVE, 18-93
UIS\$FIND_SEGMENT, 18-95
UIS\$GET_ABS_POINTER_POINTER_POS, 18-97
UIS\$GET_ALIGNED_POSITION, 18-98
UIS\$GET_ARC_TYPE, 18-100
UIS\$GET_BACKGROUND_INDEX, 18-102
UIS\$GET_BUTTONS, 18-103
UIS\$GET_CHAR_ROTATION, 18-105
UIS\$GET_CHAR_SIZE, 18-106
UIS\$GET_CHAR_SLANT, 18-108
UIS\$GET_CLIP, 18-112
UIS\$GET_COLOR, 18-115
UIS\$GET_COLORS, 18-118
UIS\$GET_CURRENT_OBJECT, 18-121
UIS\$GET_DISPLAY_SIZE, 18-123
UIS\$GET_FILL_PATTERN, 18-126
UIS\$GET_FONT, 18-129
UIS\$GET_FONT_ATTRIBUTES, 18-131
UIS\$GET_FONT_SIZE, 18-135
UIS\$GET_HW_COLOR_INFO, 18-137
UIS\$GET_INTENSITY, 18-141
UIS\$GET_INTENSITY, 18-144
UIS\$GET_KB_ATTRIBUTES, 18-146
UIS\$GET_LINE_STYLE, 18-148
UIS\$GET_LINE_WIDTH, 18-150
UIS\$GET_NEXT_OBJECT, 18-153
UIS\$GET_OBJECT_ATTRIBUTES, 18-155
UIS\$GET_PARENT_SEGMENT, 18-158
UIS\$GET_POINTER_POSITION, 18-160
UIS\$GET_POSITION, 18-162
UIS\$GET_PREVIOUS_OBJECT, 18-164
UIS\$GET_ROOT_SEGMENT, 18-167
UIS\$GET_TB_INFO, 18-169
UIS\$GET_TB_POSITION, 18-172
UIS\$GET_TEXT_FORMATTING, 18-173
UIS\$GET_TEXT_MARGINS, 18-175
UIS\$GET_TEXT_PATH, 18-177
UIS\$GET_TEXT_SLOPE, 18-179
UIS\$GET_VCM_ID, 18-181
UIS\$GET_VIEWPORT_ICON, 18-182
UIS\$GET_VIEWPORT_POSITION, 18-184
UIS\$GET_VIEWPORT_SIZE, 18-186
UIS\$GET_VISIBILITY, 18-188
UIS\$GET_WINDOW_ATTRIBUTES, 18-190
UIS\$GET_WINDOW_SIZE, 18-191
UIS\$GET_WRITING_INDEX, 18-192
UIS\$GET_WRITING_MODE, 18-194
UIS\$GET_WS_COLOR, 18-195
UIS\$GET_WS_INTENSITY, 18-198
UIS\$HLS_TO_RGB, 18-200
UIS\$HSV_TO_RGB, 18-202
UIS\$IMAGE, 18-204
UIS\$INSERT_OBJECT, 18-209
UIS\$LINE, 18-210
UIS\$LINE_ARRAY, 18-213
UIS\$MEASURE_TEXT, 18-215
UIS\$MOVE_AREA, 18-221
UIS\$MOVE_VIEWPORT, 18-224
UIS\$MOVE_WINDOW, 18-226
UIS\$NEW_TEXT_LINE, 18-228
UIS\$PLOT, 18-229
UIS\$PLOT_ARRAY, 18-232
UIS\$POP_VIEWPORT, 18-234
UIS\$PRESENT, 18-236

UIS\$PRIVATE, 18-237
 UIS\$PUSH_VIEWPORT, 18-239
 UIS\$READ_CHAR, 18-241
 UIS\$RESIZE_WINDOW, 18-243
 UIS\$RESTORE_CMS_COLORS, 18-246
 UIS\$RGB_TO_HLS, 18-247
 UIS\$RGB_TO_HSV, 18-249
 UIS\$SET_ADDOPT_AST, 18-251
 UIS\$SET_ALIGNED_POSITION, 18-253
 UIS\$SET_ARC_TYPE, 18-255
 UIS\$SET_BACKGROUND_INDEX, 18-258
 UIS\$SET_BUTTON_AST, 18-260
 UIS\$SET_CHAR_ROTATION, 18-264
 UIS\$SET_CHAR_SIZE, 18-267
 UIS\$SET_CHAR_SLANT, 18-271
 UIS\$SET_CHAR_SPACING, 18-273
 UIS\$SET_CLIP, 18-278
 UIS\$SET_CLOSE_AST, 18-281
 UIS\$SET_COLOR, 18-283
 UIS\$SET_COLORS, 18-286
 UIS\$SET_EXPAND_ICON_AST, 18-289
 UIS\$SET_FILL_PATTERN, 18-291
 UIS\$SET_FONT, 18-295
 UIS\$SET_GAIN_KB_AST, 18-297
 UIS\$SET_INSERTION_POSITION, 18-299
 UIS\$SET_INTENSITIES, 18-302
 UIS\$SET_INTENSITY, 18-304
 UIS\$SET_KB_AST, 18-306
 UIS\$SET_KB_ATTRIBUTES, 18-308
 UIS\$SET_KB_COMPOSE2, 18-311
 UIS\$SET_KB_COMPOSE3, 18-313
 UIS\$SET_KB_KEYTABLE, 18-315
 UIS\$SET_LINE_STYLE, 18-317
 UIS\$SET_LINE_WIDTH, 18-320
 UIS\$SET_LOSE_KB_AST, 18-324
 UIS\$SET_MOVE_INFO_AST, 18-326
 UIS\$SET_POINTER_AST, 18-328
 UIS\$SET_POINTER_PATTERN, 18-332
 UIS\$SET_POINTER_POSITION, 18-335
 UIS\$SET_POSITION, 18-337
 UIS\$SET_RESIZE_AST, 18-339
 UIS\$SET_SHRINK_TO_ICON_AST,
 18-344
 UIS\$SET_TB_AST, 18-346
 UIS\$SET_TEXT_FORMATTING, 18-349
 UIS\$SET_TEXT_MARGINS, 18-353
 UIS\$SET_TEXT_PATH, 18-355
 UIS\$SET_TEXT_SLOPE, 18-358
 UIS\$SET_WRITING_INDEX, 18-361
 UIS\$SET_WRITING_MODE, 18-363
 UIS\$SHRINK_TO_ICON, 18-365
 UIS\$SOUND_BELL, 18-369
 UIS\$SOUND_CLICK, 18-370
 UIS\$TEST_KB, 18-371
 UIS\$TEXT, 18-372
 UIS\$TRANSFORM_OBJECT, 18-376
 UISDC\$ALLOCATE_DOP, 19-3
 UISDC\$CIRCLE, 19-5
 UISDC\$ELLIPSE, 19-7
 UISDC\$ERASE, 19-10
 UISDC\$EXECUTE_DOP_ASYNC, 19-11
 UISDC\$EXECUTE_DOP_SYNC, 19-13
 UISDC\$GET_ALIGNED_POSITION, 19-14
 UISDC\$GET_CHAR_SIZE, 19-16
 UISDC\$GET_CLIP, 19-18
 UISDC\$GET_POINTER_POSITION, 19-20
 UISDC\$GET_POSITION, 19-22
 UISDC\$GET_TEXT_MARGINS, 19-23
 UISDC\$GET_VISIBILITY, 19-25
 UISDC\$IMAGE, 19-27
 UISDC\$LINE, 19-31
 UISDC\$LINE_ARRAY, 19-33
 UISDC\$LOAD_BITMAP, 19-35
 UISDC\$MEASURE_TEXT, 19-37
 UISDC\$MOVE_AREA, 19-39
 UISDC\$NEW_TEXT_LINE, 19-41
 UISDC\$PLOT, 19-42
 UISDC\$PLOT_ARRAY, 19-44
 UISDC\$QUEUE_DOP, 19-46
 UISDC\$READ_IMAGE, 19-47
 UISDC\$SET_ALIGNED_POSITION, 19-50
 UISDC\$SET_BUTTON_AST, 19-52
 UISDC\$SET_CHAR_SIZE, 19-54
 UISDC\$SET_CLIP, 19-56
 UISDC\$SET_POINTER_AST, 19-58
 UISDC\$SET_POINTER_PATTERN, 19-61
 UISDC\$SET_POINTER_POSITION, 19-64
 UISDC\$SET_POSITION, 19-65
 UISDC\$SET_TEXT_MARGINS, 19-66
 UISDC\$TEXT, 19-68

Index-8

V

VAX Procedure Calling Standard, 6-1

Viewing object

See Display window

Viewport

See Display viewport

Virtual display, 2-8, 7-1

aspect ratio, 2-8

creating, 2-8, 7-3

description of, 2-8

panning, 8-12

world coordinates, 2-8

zooming, 8-12

Virtual keyboard, 5-4

assignment list, 17-3

binding, 17-3

creating, 17-3

KB icon, 5-4

VMS usage, 6-3

W

Window

See Display window

Windowing

See Display window

Windowing feature, 1-7

Windowing routine, 8-1

Workstation hardware, 1-1

communications board, 1-4

keyboard, 1-3

monitor, 1-2

mouse, 1-3

printer, 1-4

processor, 1-2

tablet, 1-3

Workstation standard color

See Color system

World coordinate transformation, 8-29

scaling, 8-29

two-dimensional, 8-29

Writing color index

See Attribute

Writing mode

See Attribute

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital



No Postage
Necessary
if Mailed in the
United States



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698



Do Not Tear - Fold Here

Cut Along Dotted Line

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital



No Postage
Necessary
if Mailed in the
United States



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698



Do Not Tear - Fold Here

Cut Along Dotted Line